Escuela Superior de Cómputo

dESCOMprimidos.zip

Alberto Silva , Erick Santillan, Ulises Reyes

ICPC Regional Finals 2022

2023-09-29

Algorithm Competition Template Library
new ESCOM version 2023-09-29
'Latin America Regional Finals' Edition

Contents

contest (1)

1.1 compliec++.sh

compliec++.sh — 7 lines

```bash
green=$(tput setaf 71);
red=$(tput setaf 12);
blue=$(tput setaf 32);
orange=$(tput setaf 178);
bold=$(tput bold);
reset=$(tput sgr0);
g++ $1.cpp -o a -std=c++17 -Wall -Wextra -pedantic -O2 -Wshadow -Wformat=2 -Wfloat-equal -Wconversion -Wlogical-op -Wshift-overflow=2 -Wduplicated-cond -Wcast-qual -Wcast-align -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC -D_FORTIFY_SOURCE=2 -fsanitize=address -fsanitize=undefined -fno-sanitize-recover -fstack-protector -g3|| { echo ${bold}${orange}Compilation Error in ${reset} $1.cpp; exit 1; }
```

1.2 hash.sh

hash.sh — 3 lines

```bash
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp $1 -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

1.3 largeTemplate.cpp

largeTemplate.cpp — 82 lines

```cpp
#define _GLIBCXX_DEBUG 1
#define _GLIBCXX_DEBUG_PEDANTIC 1
#define _FORTIFY_SOURCE 2
#pragma GCC target ("avx2")
#pragma GCC optimization ("O3")
#pragma GCC optimization ("unroll-loops")
#pragma GCC optimize("Ofast")
#pragma GCC target("avx,avx2,fma")
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
typedef long long int lli;
#define __ ios_base::sync_with_stdio(false);cin.tie(NULL);
#define print(A) for(auto c:A)cout<<c<<" ";cout<<endl;
#define printM(A) for(auto c:A){print(c);}
#define x first
#define y second
#define printP(A)for(auto c:A)cout<<"("<<c.x<<","<<c.y<<") ";cout<<endl;
#define printMP(A)for(auto c:A){printP(c);}
#define MOD(n,k) ( ( ((n) % (k)) + (k) ) % (k))
void Madd(int &res, int val) { if ((res += val) >= MOD) res -= MOD; }
void Mres(int &res, int val) { if ((res -= val) < 0 ) res += MOD; }
//DEBUG
#define error(args...) { string _s = #args; replace(_s.begin(), _s.end(), ',', ' ');\
stringstream _ss(_s); istream_iterator<string> _it(_ss); err(_it, args); }
#define cerr(s) cerr << "\033[48;5;196m\033[38;5;15m" << s << "\033[0m"
void err(istream_iterator<string> it) {}
template<typename T, typename... Args>
void err(istream_iterator<string> it, T a, Args... args) {
 cerr << *it << " = " << a << endl;
 err(++it, args...);
}
#define rep(i, begin, end) for (__typeof(end) i = (begin) - ((begin) > (end));\
i != (end) - ((begin) > (end)); i += 1 - 2 * ((begin) > (end)))
// Dont use #define int long long with this
//Orderded set
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/trie_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update> ordered_set;
// ordered_set st;
// st.order_of_key();
// st.find_by_order();
//Multiset
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int,null_type,less_equal<int>,rb_tree_tag,tree_order_statistics_node_update> ordered_set;
// For erase use:
st.erase(st.find_by_order(st.order_of_key(x)));
// And for check if exisit use :
int order = st.order_of_key(l);
int num = *st.find_by_order(order);
if(num == l){
 continue;
}
//Better hashmap
#include <ext/pb_ds/assoc_container.hpp>
#include <bits/extc++.h>
using namespace __gnu_pbds;
const int RANDOM = chrono::high_resolution_clock::now().time_since_epoch().count();
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
 const uint64_t C = lli(4e18 * acos(0)) | 71;
 lli operator()(lli x) const { return __builtin_bswap64(x*C); }
};s
gp_hash_table<lli,int,chash> h({},{},{},{},{1<<16});
// For CodeForces, or other places where hacking might be a problem:
struct chash { // To use most bits rather than just the lowest ones:
 const uint64_t C = ll(4e18 * acos(0)) | 71; // large odd number
 ll operator()(ll x) const { return __builtin_bswap64((x^RANDOM)*C); }
};
__gnu_pbds::gp_hash_table<ll, int, chash> h({},{},{},{}, {1 << 16});
int main(){__
 int n;
 cin>>n;
 vector<int> nums(n);
 for(auto &c:nums)cin>>c;
 bool flag = true;
 cout << "NO\0YES\0" + 3 * flag << endl;
  return 0;
}
```

1.4 template.cpp

template.cpp — 16 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define int long long
#define __ ios_base::sync_with_stdio(false);cin.tie(NULL);
signed main(){__
 int T= 1,n;
  while(T--){
    cin>>n;
    vector<int> nums(n);
    for(auto &c:nums)cin>>c;
    bool flag = true;
    cout << "NO\0YES\0" + 3 * flag << endl;
  }
  return 0;
}
```

1.5 template.py

template.py — 8 lines

```python
import sys
from collections import *
from itertools import *
input = sys.stdin.readline
for _ in range(int(input())):
  input()
  n, k = map(int, input().split())
  u = list(map(int, input().split()))
```

1.6 troubleshoot.txt

troubleshoot.txt 53 lines

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

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

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

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

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

mathematics (2)

2.1 Equations

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

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

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

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

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

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

2.2 Recurrences

If $a_n=c_1a_{n-1}+\cdots+c_ka_{n-k}$, 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: $\dfrac{a+b}{a-b}=\dfrac{\tan\frac{\alpha+\beta}{2}}{\tan\frac{\alpha-\beta}{2}}$

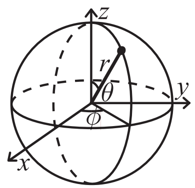
2.4.2 Quadrilaterals

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

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

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

2.4.3 Spherical coordinates



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

2.5 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x = \frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}\tan x = 1+\tan^2 x \qquad \frac{d}{dx}\arctan x = \frac{1}{1+x^2}$$

$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \qquad \int x\sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\operatorname{erf}(x) \qquad \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1)$$

Integration by parts:

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

2.6 Sums

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

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

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

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

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

2.7 Series

$$e^x = 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots,\ (-\infty < x < \infty)$$

$$\ln(1+x) = x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\ldots,\ (-1 < x \leq 1)$$

$$\sqrt{1+x} = 1+\frac{x}{2}-\frac{x^2}{8}+\frac{2x^3}{32}-\frac{5x^4}{128}+\ldots,\ (-1 \leq x \leq 1)$$

$$\sin x = x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots,\ (-\infty < x < \infty)$$

$$\cos x = 1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots,\ (-\infty < x < \infty)$$

2.8 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x-\mathbb{E}(X))^2 p_X(x)$ 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,

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

2.8.1 Discrete distributions

Binomial distribution

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

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

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

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

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each wich yields success with probability p is $\mathrm{Fs}(p)$, $0 \leq p \leq 1$.

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

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

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 $\mathrm{Po}(\lambda)$, $\lambda = t\kappa$.

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

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

2.8.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

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

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

2.9 Markov chains

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

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

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

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

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

strings (3)

3.1 AhoCorasick.cpp

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

AhoCorasick.cpp — 634404, 66 lines

```cpp
struct AhoCorasick {
  struct Node : map<char, int> {
    int link = 0;
    int cnt = 0;
    int tin = 0, tout = 0;
  };
  vector<Node> trie;
  vector<vi> graph;
  AhoCorasick() { newNode(); }
  int newNode() {
    trie.pb({});
    graph.pb({});
    return sz(trie) - 1;
  }
  int insert(string &s, int u = 0) {
    for (char c : s) {
      if (!trie[u][c])
        trie[u][c] = newNode();
      u = trie[u][c];
    }
    trie[u].cnt++;
    return u;
  }
  int go(int u, char c) {
    while (u && !trie[u].count(c))
      u = trie[u].link;
    return trie[u][c];
  }
  void pushLinks() {
    queue<int> Q;
    Q.push(0);
    while (!Q.empty()) {
      int u = Q.front();
      Q.pop();
      for (auto &[c, v] : trie[u]) {
        int l = (trie[v].link = u ? go(trie[u].link, c) : 0);
        trie[v].cnt += trie[l].cnt;
        Q.push(v);
      }
    }
  }
  void buildTree() {
    fore (u, 0, sz(trie))
      graph[trie[u].link].pb(u);
    int timer = 0;
    function<void(int)> dfs = [&](int u) {
      trie[u].tin = ++timer;
      for (int v : graph[u])
        if (!trie[v].tin)
          dfs(v);
      trie[u].tout = timer;
    };
    dfs(0);
  }
  int match(string &s, int u = 0) {
    lli ans = 0;
    for (char c : s) {
      u = go(u, c);
      ans += trie[u].cnt;
    }
    return ans;
  }
  Node& operator [](int u) {
    return trie[u];
  }
}AHO;
```

3.2 Hashing-codeforces.h

Hashing-codeforces.h — 46 lines

```cpp
typedef uint64_t ull;
static int C; // initialized below
// Arithmetic mod two primes and 2^32 simultaneously.
// "typedef uint64_t H;" instead if Thue-Morse does not apply.
template<int M, class B>
struct A {
 int x; B b; A(int x=0) : x(x), b(x) {}
 A(int x, B b) : x(x), b(b) {}
 A operator+(A o){int y = x+o.x; return{y - (y>=M)*M, b+o.b};}
 A operator-(A o){int y = x-o.x; return{y + (y< 0)*M, b-o.b};}
 A operator*(A o) { return {(int)(1LL*x*o.x % M), b*o.b}; }
 explicit operator ull() { return x ^(ull) b << 21; }
};
typedef A<1000000007, A<1000000009, unsigned>> H;
struct HashInterval {
 vector<H> ha, pw;
 HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
  pw[0] = 1;
  rep(i,0,sz(str))
   ha[i+1] = ha[i] * C + str[i],
   pw[i+1] = pw[i] * C;
 }
 H hashInterval(int a, int b) { // hash [a, b)
  return ha[b] - ha[a] * pw[b - a];
 }
};
vector<H> getHashes(string& str, int length) {
 if (sz(str) < length) return {};
 H h = 0, pw = 1;
 rep(i,0,length)
  h = h * C + str[i], pw = pw * C;
 vector<H> ret = {h};
```

```
  rep(i,length,sz(str)) {
    ret.push_back(h = h * C + str[i] - pw * str[i-length]);
  }
  return ret;
}
H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
#include <sys/time.h>
int main() {
  timeval tp;
  gettimeofday(&tp, 0);
  C = (int)tp.tv_usec; // (less than modulo)
  assert((ull)(H(1)*2+1-3) == 0);
  // ...
}
```

3.3 Hashing.h

Hashing.h 40 lines

```
// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is random,
// or work mod 10^9+7 if the Birthday paradox is not a problem.
struct H {
  typedef uint64_t ull;
  ull x; H(ull x=0) : x(x) {}
  #define OP(O,A,B) H operator O(H o) { ull r = x; asm (A "addq %%rdx,
      ↪ %0\n adcq $0,%0" : "+a"(r) : B); return r; }
  OP(+,,"d"(o.x)) OP(*,"mul %1\n", "r"(o.x) : "rdx")
  H operator-(H o) { return *this + ~o.x; }
  ull get() const { return x + !~x; }
  bool operator==(H o) const { return get() == o.get(); }
  bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (ll)1e11+3; // (order ~ 3e9; random also ok)
struct HashInterval {
  vector<H> ha, pw;
  HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
    pw[0] = 1;
    rep(i,0,sz(str))
      ha[i+1] = ha[i] * C + str[i],
      pw[i+1] = pw[i] * C;
  }
  H hashInterval(int a, int b) { // hash [a, b)
    return ha[b] - ha[a] * pw[b - a];
  }
};
vector<H> getHashes(string& str, int length) {
  if (sz(str) < length) return {};
  H h = 0, pw = 1;
  rep(i,0,length)
    h = h * C + str[i], pw = pw * C;
  vector<H> ret = {h};
  rep(i,length,sz(str)) {
    ret.push_back(h = h * C + str[i] - pw * str[i-length]);
  }
  return ret;
}
H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
```

3.4 KMP.cpp

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

$$0...x$$

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

KMP.cpp b6331e, 28 lines

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

3.5 Manacher.cpp

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

Manacher.cpp 22612e, 14 lines

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

3.6 MinRotation.cpp

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

MinRotation.cpp 975539, 9 lines

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

3.7 RollingHashes.cpp

RollingHashes.cpp 134 lines

```
#include <bits/stdc++.h>
using namespace std;
typedef unsigned long long ull;
// Generate random base in (before, after) open interval:
int gen_base(const int before, const int after) {
    auto seed = chrono::high_resolution_clock::now().time_since_epoch().
        ↪ count();
    seed ^= ull(new ull);
    mt19937 mt_rand(seed);
    int base = uniform_int_distribution<int>(before+1, after)(mt_rand);
    return base % 2 == 0 ? base-1 : base;
```

```cpp
}
struct PolyHash {
	static vector<int> pow1;
	static vector<ull> pow2;
	static int base;
	static inline int diff(int a, int b) {
		return (a -= b) < 0 ? a + 2147483647 : a;
	}
	static inline int mod(ull x) {
		x += 2147483647;
		x = (x >> 31) + (x & 2147483647);
		return int((x >> 31) + (x & 2147483647));
	}
	vector<int> pref1;
	vector<ull> pref2;
	inline int get_pow1(int p) const {
		static int __base[4] = {1, base, mod(ull(base) * base), mod(mod(ull(base) * base) * ull(base))};
		return mod(ull(__base[p % 4]) * pow1[p / 4]);
	}
	inline ull get_pow2(int p) const {
		static ull __base[4] = {ull(1), ull(base), ull(base) * base, ull(base) * base * base};
		return pow2[p / 4] * __base[p % 4];
	}
	PolyHash(const string& s)
		: pref1(s.size()+1u, 0)
		, pref2(s.size()+1u, 0)
	{
		const int n = s.size();
		pow1.reserve((n+3)/4);
		pow2.reserve((n+3)/4);
		int pow1_4 = mod(ull(base) * base);
		pow1_4 = mod(ull(pow1_4) * pow1_4);
		ull pow2_4 = ull(base) * base;
		pow2_4 *= pow2_4;
		while (4 * (int)pow1.size() <= n) {
			pow1.push_back(mod((ull)pow1.back() * pow1_4));
			pow2.push_back(pow2.back() * pow2_4);
		}
		int curr_pow1 = 1;
		ull curr_pow2 = 1;
		for (int i = 0; i < n; ++i) {
			assert(base > s[i]);
			pref1[i+1] = mod(pref1[i] + (ull)s[i] * curr_pow1);
			pref2[i+1] = pref2[i] + s[i] * curr_pow2;
			curr_pow1 = mod((ull)curr_pow1 * base);
			curr_pow2 *= base;
		}
	}
	inline pair<int, ull> operator()(const int pos, const int len, const int mxPow = 0) const {
		int hash1 = pref1[pos+len] - pref1[pos];
		ull hash2 = pref2[pos+len] - pref2[pos];
		if (hash1 < 0) hash1 += 2147483647;
		if (mxPow != 0) {
			hash1 = mod((ull)hash1 * get_pow1(mxPow-(pos+len-1)));
			hash2 *= get_pow2(mxPow-(pos+len-1));
		}
		return make_pair(hash1, hash2);
	}
};
int PolyHash::base((int)1e9+7);
vector<int> PolyHash::pow1{1};
vector<ull> PolyHash::pow2{1};
int main() {
	string a;
	{
		vector<char> buf(1+1000000);
		scanf("%1000000s", &buf[0]);
		a = string(&buf[0]);
	}
	PolyHash::base = gen_base(256, 2147483647);
	PolyHash hash_a(a);
	reverse(a.begin(), a.end());
	PolyHash hash_b(a);
	// Get length of strings (mxPow == n)
	const int n = (int)a.size();
	ull answ = 0;
	for (int i = 0, j = n-1; i < n; ++i, --j) {
		// Palindromes odd length:
		int low = 0, high = min(n-i, n-j)+1;
		while (high - low > 1) {
			int mid = (low + high) / 2;
			if (hash_a(i, mid, n) == hash_b(j, mid, n)) {
				low = mid;
			} else {
				high = mid;
			}
		}
		answ += low;
		// Palindromes even length:
		low = 0, high = min(n-i-1, n-j)+1;
		while (high - low > 1) {
			int mid = (low + high) / 2;
			if (hash_a(i+1, mid, n) == hash_b(j, mid, n)) {
				low = mid;
			} else {
				high = mid;
			}
		}
		answ += low;
	}
	cout << answ;
	return 0;
}
// int PolyHash::base((int)1e9+7);
// vector<int> PolyHash::pow1{1};
// vector<ull> PolyHash::pow2{1};
// int main(){
//   string a,b;
//   cin>>a;
//   b = a;
//   reverse(a.begin(),a.end());
//   const int mxPow = std::max((int)a.size(), (int)b.size());
//   PolyHash::base = gen_base(256, PolyHash::mod);
//   PolyHash hash_a(a), hash_b(b); n
//   for (int i = 0; i<a.size();i++){
//     auto need = hash_a(0,i+1,mxPow);
//     int size = a.size();
//     if
//     if (hash_a(i, b.size(), mxPow) == need) {
//       printf("%d ", i);
//     }
//   }
//   return 0;
// }
```

3.8 SuffixArray.cpp

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

SuffixArray.cpp 237f0d, 70 lines

```cpp
void radix_sort(vector<int> &P,vector<int> &c){
	int n = P.size();
	vector<int> cnt(n);
	for(auto d:c)
		cnt[d]++;
	vector<int> pos(n);
	vector<int> nP(n);
	pos[0]= 0;
	for(int i = 1;i<n;i++)
		pos[i] = pos[i-1]+cnt[i-1];
	for(auto d:P){
		int i = c[d];
```

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

3.9 SuffixTree.cpp

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

SuffixTree.cpp 797ab6, 59 lines

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

3.10 Trie.cpp

Trie.cpp 93 lines

```
#include <bits/stdc++.h>
using namespace std;
struct Trie{
    unordered_map<char, Trie*> child;
    int prefix;
    bool end;
};
struct Trie *getNode(){
    struct Trie *p = new Trie;
    //Must initialize values
    p->end = false;
    p->prefix = 0;
    return p;
}
void insert(struct Trie *root, string key){
    struct Trie *S = root;
```

```
    for(int i = 0 ; i < key.length(); i++){
        if(S->child.find(key[i]) == S->child.end())
            S->child[key[i]] = getNode();
        S = S->child[key[i]];
        S->prefix++;
    }
    S-> end =true;
}
bool search(struct Trie *root, string key){
    struct Trie *S = root;
    int n = key.size();
    for(int i = 0;i<n;i++){
        if(S->child.find(key[i]) == S->child.end())
            return false;
        S = S->child[key[i]];
    }
    if(S->end)return true;
    else
        return false;
}
int countprefixes(Trie* root,string s){
    int n= s.size();
    Trie* mov = root;
    for(int i=0;i<n;i++){
        if(mov->child.find(s[i]) == mov->child.end())
             return 0;
        mov=mov->child[s[i]];
    }
    return mov->prefix;
}
Trie* remove(Trie* root, string word, int depth = 0){
    if(!root)
        return NULL;
    if (depth == word.size()) {
        if (root->end)
            root->end = false;
        if (root->child.size()) {
            delete (root);
            root = NULL;
        }
        return root;
    }
    root->child[word[depth]] = remove(root->child[word[depth]], word,
         depth + 1);
    if (root->child.size()== 0 && root->end == false) {
        delete(root);
        root=NULL;
    }
    return root;
}
void print(Trie* root,char str[],int level){
    if(root->end){
        str[level] = '\0';
        cout<<str<<endl;
    }
    for(auto c:root->child){
        str[level] = c.first;
        print(c.second,str,level+1);
    }
}
int main(){
    int t,n,m;
    string s;
    cin >> t;
    Trie *root = getNode();
    while(t--){
        cin >> s;
        insert(root,s);
    }
    char str[100];
    print(root,str,0);
    cin>>m;
    for(int i = 0;i<m;i++){
        cin>>s;
        remove(root,s);
    }
    print(root,str,0);
}
```

3.11 TrieBinary.cpp

TrieBinary.cpp — 60 lines

```
#include <bits/stdc++.h>
using namespace std;
const int N = 20000043;
int t[N];
int nxt[N][2];
int szt = 1;
int new_vertex(){
  nxt[szt][0] = nxt[szt][1] = -1;
  t[szt] = 0;
  return szt++;
}
int go(int x, int y){
  if (nxt[x][y] == -1)
    nxt[x][y] = new_vertex();
  return nxt[x][y];
}
void add_number(int x, int val){
  int cur = 0;
  t[cur] += val;
  for(int i = 29; i >= 0; i--){
    int z = ((x >> i) & 1);
    cur = go(cur, z);
    t[cur] += val;
  }
}
int descend(int x){
  int ans = 0;
  int cur = 0;
  for(int i = 29; i >= 0; i--){
    int z = ((x >> i) & 1);
    int k;
    if(nxt[cur][z] != -1 && t[nxt[cur][z]] > 0)
      k = z;
    else
      k = z ^1;
    ans ^= (k << i);
    cur = go(cur, k);
  }
  return ans;
}
int n;
const int M = 300043;
int a[M];
int p[M];
int main() {
    int n;
    cin>>n;
    vector<int> a(n);
    vector<int> p(n);
    nxt[0][0] = nxt[0][1] = -1;
    for(auto &c:a)cin>>c;
    for(auto &c:p)cin>>c;
  for(auto c:p) add_number(c,1);
  for(int i = 0;i<n;i++){
    int z = descend(a[i]);
    cout<<a[i]^z<<" ";
    add_number(z, -1);
  }
  return 0;
}
```

3.12 Zfunc.cpp

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

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

Zfunc.cpp — acb5c6, 73 lines

```cpp
vector<int> zf (string s) {
 int n = s.size();
 vector<int> z (n);
 for (int i = 1, l = 0, r = 0; i < n; i++) {
  if (i <= r)
   z[i] = min (r - i + 1, z[i - l]);
  while (i + z[i] < n && s[z[i]] == s[i + z[i]])
   z[i]++;
  if (i + z[i] - 1 > r)
   l = i, r = i + z[i] - 1;
 }
 return z;
}
// Not sure if it's correct, works for leetcode problem
// Suppose you can change from pattern, p[i] to some set of characters c ,
// you can use this code to deal with that case
// ONLY CAN CHECK IF EXIST ONE OCURRENCE , IT WILL FAIL TO TRY TO GET ALL THE OCURRENCES
// LIKE IN THIS EXAMPLE
// Possible mappings:
// a -> {a,b}
// b -> {b,c}
// c -> {c,a}
// t = abc
// s = abcaaba
class MatchWithWildcards {
public:
  vector<int> z_function(const string& s,map<char,set<char>> &mp){
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
      if(i < r) {
        z[i] = min(r - i, z[i - l]);
      }
      while(i + z[i] < n && mp[s[z[i]]].count(s[i+z[i]])) {
        z[i]++;
      }
      if(i + z[i] > r) {
        l = i;
        r = i + z[i];
      }
    }
    return z;
  }
  bool match(const string& s, const string& pat, map<char,set<char>> & mp) {
    int n = pat.size();
    int m = s.size();
    if(m<n){
      cout<<endl;
      return false;
    }
    string match = pat+"#"+s;
    vector<int> z =z_function(match,mp);
    for(int i = n; i<match.size(); i++){
      if(z[i] == n){
        return true;
      }
    }
    return false;
  }
  // Mappings are of the form: [[a,b]] -> a can be replaced by b
  bool matchReplacement(string s, string sub, vector<vector<char>>& mappings) {
    map<char,set<char>> mp;
    for(auto c:mappings){
      mp[c[0]].insert(c[1]);
      mp[c[0]].insert(c[0]);
    }
    for(auto c:sub){
      mp[c].insert(c);
    }
    return match(s,sub,mp);
  }
};
```

3.13 palindromicTree.cpp

palindromicTree.cpp — 51 lines

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

3.14 positionInLexicograficOrder.cpp

Description: Get the position of a string in the lexicografic order with respect to all permutations of the string
Time: $\mathcal{O}(n)$

positionInLexicograficOrder.cpp — 685bb8, 25 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
int fact(int n){
  return (n<=1)?1:n*fact(n-1);
}
int findRank(string s) {
```

```
    int n = s.size();
    int mul = fact(n);
    int position = 1, i;
    vector<int> count(256);
    for (int i = 0; s[i]; ++i) count[s[i]]++;
    for (int i = 1; i < 256; ++i) count[i] += count[i - 1];
    for (i = 0; i < n; ++i) {
        mul /= n - i;
        position += count[s[i] - 1] * mul;
        for (int j = s[i]; j < 256; j++)count[j]--;
    }
    return position;
}
int main(){
    int n;
    string s;
    cin>>s;
    cout<<findRank(s);
}
```

3.15 suffixAutomaton.cpp

Description: Suffix automaton creates an automaton that recognizes all the suffixes of a string.
Time: $\mathcal{O}(N)$

suffixAutomaton.cpp 8ec082, 88 lines

```
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define __ ios_base::sync_with_stdio(false),cin.tie(NULL);
struct SuffixAutomaton {
    vector<map<int,int>> to;
    vector<int> link;
    vector<int> len;
    int last;
    SuffixAutomaton(string s) {
        to.push_back(map<int,int>());
        link.push_back(-1);
        len.push_back(0);
        last = 0;
        for(int i=0;i<s.size();i++) {
            int c = int(s[i]);
            to.push_back(map<int,int>());
            len.push_back(i+1);
            link.push_back(0);
            int cur = to.size() - 1;
            int p = last;
            while(p >= 0 && !to[p].count(c)) {
                to[p][c] = cur;
                p = link[p];
            }
            if(p != -1){
                int q = to[p][c];
                if(len[p] + 1 == len[q])
                    link[cur] = q;
                else {
                    to.push_back(to[q]);
                    len.push_back(len[p] + 1);
                    link.push_back(link[q]);
                    int clone = to.size()-1;
                    link[q] = link[cur] = clone;
                    while(p >= 0 && to[p].count(c) && to[p][c] == q) {
                        to[p][c] = clone;
                        p = link[p];
                    }
                }
            }
            last = cur;
        }
    }
};
//Aditonial implementations.
//Get de kth lexicografic string
vector<int> dp;
void getSize(SuffixAutomaton SA){
    int n = SA.to.size();
    vector<int> order(n);
    dp.resize(n,0);
    iota(order.begin(),order.end(),0);
    sort(order.begin(),order.end(),[&](int a,int b){
        return SA.len[a]>SA.len[b];
    });
    for(int i = 0;i<n;i++){
        dp[order[i]] = 1;
        for(auto c:SA.to[order[i]])
            dp[order[i]]+=dp[c.second];
    }
};
int main(){__
    int n = 1,k;
    string s;
    cin>>s;
    cin>>n;
    SuffixAutomaton SA(s);
    getSize(SA);
    for(int i = 0;i<n;i++){
        cin>>k;
        int u = 0;
        while(k){
            for(auto c:SA.to[u]){
                if(dp[c.second]>=k){
                    k--;
                    u = c.second;
                    cout<<char(c.first);
                    break;
                }
                else
                    k-=dp[c.second];
            }
        }
        cout<<endl;
    }
    return 0;
}
```

number-theory (4)

4.1 CRT.h

Description: Chinese Remainder Theorem. crt(a, m, b, n) computes x such that $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$. If $|a| < m$ and $|b| < n$, x will obey $0 \le x < \mathrm{lcm}(m, n)$. Assumes $mn < 2^{62}$. given a set of congruence equations -> $a \equiv a_1 (mod p_1)$ $a \equiv a_2 (mod p_2)$ ... $a \equiv a_k (mod p_k)$ Return a if p_i are pairwise coprimes
Time: $\log(n)$

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

```
int crt(int a, int m, int b, int n) {
  if (n > m) swap(a, b), swap(m, n);
  int x, y, g = euclid(m, n, x, y);
  assert((a - b) % g == 0); // else no solution
  x = (b - a) % n * x % n / g * m + a;
  return x < 0 ? x + m*n/g : x;
}
int lcm(int a,int b){
  return a*b/__gcd(a,b);
}
vector<int>nums;
vector<int>rem;
int CRT() {
    int prod = 1;
    for (int i = 0; i < nums.size(); i++)
        prod *= nums[i];
    int result = 0;
    for (int i = 0; i < nums.size(); i++) {
        int pp = prod / nums[i];
        result += rem[i] * inverse(pp, nums[i]) * pp;
    }
    return result % prod;
}
```

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

4.2 CarmichaelLambda.cpp

CarmichaelLambda.cpp 58 lines

```
typedef long long ll;
ll gcd(ll a, ll b)
{
 while (a) swap(a, b %= a);
 return b;
}
ll lcm(ll a, ll b)
{
 return a * (b / gcd(a, b));
}
ll carmichael_lambda(ll n)
{
 ll lambda = 1;
 if (n % 8 == 0)
  n /= 2;
 for (ll d = 2; d * d <= n; ++d)
  if (n % d == 0)
  {
   n /= d;
   ll y = d - 1;
   while (n % d == 0)
   {
    n /= d;
    y *= d;
   }
   lambda = lcm(lambda, y);
  }
 if (n > 1)
  lambda = lcm(lambda, n - 1);
 return lambda;
}
// lambda(n) for all n in [lo, hi)
vector<ll> carmichael_lambda(ll lo, ll hi)
{
 vector<ll> ps = primes(sqrt(hi) + 1);
 vector<ll> res(hi - lo), lambda(hi - lo, 1);
 iota(res.begin(), res.end(), lo);
 for (ll k = ((lo + 7) / 8) * 8; k < hi; k += 8)
  res[k - lo] /= 2;
 for (ll p : ps)
  for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p)
  {
   if (res[k - lo] < p)
    continue;
   ll t = p - 1;
   res[k - lo] /= p;
   while (res[k - lo] > 1 && res[k - lo] % p == 0)
   {
    t *= p;
    res[k - lo] /= p;
   }
   lambda[k - lo] = lcm(lambda[k - lo], t);
  }
 for (ll k = lo; k < hi; ++k)
  if (res[k - lo] > 1)
   lambda[k - lo] = lcm(lambda[k - lo], res[k - lo] - 1);
 return lambda; // lambda[k-lo] = lambda(k)
}
```

4.3 ContinuedFractions.h

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

ContinuedFractions.h dd6c5e, 21 lines

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

4.4 Euclidjav.h

Euclidjav.h 11 lines

```
static BigInteger[] euclid(BigInteger a, BigInteger b) {
 BigInteger x = BigInteger.ONE, yy = x;
 BigInteger y = BigInteger.ZERO, xx = y;
 while (b.signum() != 0) {
  BigInteger q = a.divide(b), t = b;
  b = a.mod(b); a = t;
  t = xx; xx = x.subtract(q.multiply(xx)); x = t;
  t = yy; yy = y.subtract(q.multiply(yy)); y = t;
 }
 return new BigInteger[]{x, y, a};
}
```

4.5 Factor.h

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).
Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.

Factor.h - "MillerRabin.h" a33cf6, 18 lines

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

```
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

4.6 FastCountDivisors.cpp

Description: Count the number of divisors of a large number

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

FastCountDivisors.cpp – "MillerRabin", "primes" 13e60e, 93 lines

```
/** Need primes[],lp[],N= 10^6 */
#define lli long long
bool isSquare(lli val){
  lli lo = 1, hi = val;
  while(lo <= hi){
    lli mid = lo + (hi - lo) / 2;
    lli tmp = (val / mid) / mid; // be careful with overflows!!
    if(tmp == 0)hi = mid - 1;
    else if(mid * mid == val)return true;
    else if(mid * mid < val)lo = mid + 1;
  }
  return false;
}
lli countDivisors(lli n) {
  lli ans = 1;
  for(int i = 0; i < primes.size(); i++){
    if(n == 1)break;
    int p = primes[i];
    if(n % p == 0){ // checks whether p is a divisor of n
      int num = 0;
      while(n % p == 0){
        n /= p;
        ++num;
      }
      // p^num divides initial n but p^(num+1) does not divide initial val
      // => p can be taken 0 to num times => num + 1 possibilities!!
      ans *= num + 1;
    }
  }
    if(n == 1)return ans; // first case
  else if(isPrime(n))return ans * 2; // second case
  else if(isSquare(n))return ans * 3; // third case but with p == q
  else return ans * 4; // third case with p != q
}
using uint32 = unsigned int;
using uint64 = unsigned long long;
using uint128 = __uint128_t;
// compute \sum_{i=1}^{n} \sigma(i) in O(n^{1/3}) time.
// it is also equal to \sum_{i=1}^{n} floor(n / i)
// takes ~100 ms for n = 1e18
uint128 sum_sigma0(uint64 n) {
  auto out = [n] (uint64 x, uint32 y) {
    return x * y > n;
  };
  auto cut = [n] (uint64 x, uint32 dx, uint32 dy) {
    return uint128(x) * x * dy >= uint128(n) * dx;
  };
  const uint64 sn = sqrtl(n);
  const uint64 cn = pow(n, 0.34); //cbrtl(n);
  uint64 x = n / sn;
  uint32 y = n / x + 1;
  uint128 ret = 0;
  stack<pair<uint32, uint32>> st;
  st.emplace(1, 0);
  st.emplace(1, 1);
  while (true) {
    uint32 lx, ly;
    tie(lx, ly) = st.top();
    st.pop();
    while (out(x + lx, y - ly)) {
    ret += x * ly + uint64(ly + 1) * (lx - 1) / 2;
    x += lx, y -= ly;
    }
    if (y <= cn) break;
    uint32 rx = lx, ry = ly;
    while (true) {
    tie(lx, ly) = st.top();
    if (out(x + lx, y - ly)) break;
    rx = lx, ry = ly;
    st.pop();
    }
    while (true) {
    uint32 mx = lx + rx, my = ly + ry;
    if (out(x + mx, y - my)) {
      st.emplace(lx = mx, ly = my);
    }
    else {
      if (cut(x + mx, lx, ly)) break;
      rx = mx, ry = my;
    }
    }
  }
  for (--y; y > 0; --y) ret += n / y;
  return ret * 2 - sn * sn;
}
auto ans = sum_sigma0(n);
string s = "";
while (ans > 0) {
  s += char('0' + ans % 10);
  ans /= 10;
}
reverse(s.begin(), s.end());
cout << s << '\n';
```

4.7 FracBinarySearch.h

Description: Given f and N, finds the smallest fraction $p/q \in [0,1]$ such that $f(p/q)$ is true, and $p,q \le N$. You may want to throw an exception from f if it finds an exact solution, in which case N can be removed.

Usage: `fracBS([](Frac f) { return f.p>=3*f.q; }, 10); // {1,3}`

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

FracBinarySearch.h 27ab3e, 24 lines

```
struct Frac { ll p, q; };
template<class F>
Frac fracBS(F f, ll N) {
  bool dir = 1, A = 1, B = 1;
  Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
  if (f(lo)) return lo;
  assert(f(hi));
  while (A || B) {
    ll adv = 0, step = 1; // move hi if dir, else lo
    for (int si = 0; step; (step *= 2) >>= si) {
      adv += step;
      Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
      if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
        adv -= step; si = 2;
      }
    }
    hi.p += lo.p * adv;
    hi.q += lo.q * adv;
    dir = !dir;
    swap(lo, hi);
    A = B; B = !!adv;
  }
  return dir ? hi : lo;
}
```

4.8 LinealDiophantine.cpp

LinealDiophantine.cpp 63 lines

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

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

4.9 LinearSieve.py

LinearSieve.py 11 lines

```
N = 10**6 +7
lp = [0]*(N+1)
primes = []
for i in range(2,N+1):
  if lp[i] == 0:
    lp[i] = i
    primes.append(i)
  for j in range(len(primes)):
    if i*primes[j]>N:break
    if primes[j]>lp[i]: break
    lp[primes[j]*i] = primes[j]
```

4.10 ModFloorDivision.h

Description: Sum of aritmetic floor division
f(a, b, c, n) $= \sum_{i=0}^{n} \lfloor \frac{(ai+b)}{c} \rfloor$.
Time: $\log(a)$,

ModFloorDivision.h f58bf7, 16 lines

```
int f(int a, int b, int c, int n){
  int m = (a*n + b)/c;
  if(n==0 || m==0) return b/c;
  if(n==1) return b/c + (a+b)/c;
  if(a<c && b<c) return m*n - f(c, c-b-1, a, m-1);
  else return (a/c)*n*(n+1)/2 + (b/c)*(n+1) + f(a%c, b%c, c, n);
}
// ∑_{k=1}^{n} ⌊n/k⌋)
int floor_sum(int n) {
  int sum = 0;
  for (int i = 1, last; i <= n; i = last + 1) {
    last = n / (n / i);
    sum += (n / i) * (last - i + 1);
  }
  return sum;
}
```

4.11 ModInverse.h

ModInverse.h 15 lines

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

4.12 ModLog.h

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

ModLog.h c040b8, 11 lines

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

4.13 ModMulLL.h

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for $0 \le a, b \le c \le 7.2 \cdot 10^{18}$.
Time: $\mathcal{O}(1)$ for modmul, $\mathcal{O}(\log b)$ for modpow

ModMulLL.h bbbd8f, 11 lines

```
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
  ull ans = 1;
```

```
  for (; e; b = modmul(b, b, mod), e /= 2)
    if (e & 1) ans = modmul(ans, b, mod);
  return ans;
}
```

4.14 ModPow.cpp

ModPow.cpp 10 lines

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

4.15 ModSqrt.h

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

ModSqrt.h - "ModPow.h" 19a793, 24 lines

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

4.16 ModSum.h

Description: Sums of mod'ed arithmetic progressions.
$f(a, b, c, n) = \sum_{i=0}^{to-1}(ki + c)\%m$.
Time: $\log(m)$, with a large constant.

ModSum.h 5c5bc5, 14 lines

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

4.17 ModularArithmetic.h

ModularArithmetic.h 19 lines

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

4.18 NumberTheory.cpp

NumberTheory.cpp 448 lines

```
#include <bits/stdc++.h>
using namespace std;
using Integer128 = __int128;
typedef __int128 Intsote;
typedef long long int lli;
/*
  Function to print __int128
*/
ostream& operator<<( ostream& dest, __int128_t value ){
  ostream::sentry s( dest );
  if ( s ) {
    __uint128_t tmp = value < 0 ? -value : value;
    char buffer[ 128 ];
    char* d = std::end( buffer );
    do {
      -- d;
      *d = "0123456789"[ tmp % 10 ];
      tmp /= 10;
    } while ( tmp != 0 );
    if ( value < 0 ) {
      -- d;
      *d = '-';
    }
    int len = end( buffer ) - d;
    if ( dest.rdbuf()->sputn( d, len ) != len ) {
      dest.setstate( ios_base::badbit );
    }
  }
  return dest;
}
//__________Exponenciacion y multiplicacion modular_________//
/*
 [Tested Timus 1141,1204, uva 1230,374,11029]
*/
//Ed1.- Do Not use mod_mult if it gives TLE
//Ed2.- Usually in small base and exponent
//Ed3.- Al chile ya no lo uses
lli mod_mult(lli a, lli b, lli mod){
  lli x = 0;
  while(b){
    if(b & 1) x = (x + a) % mod;
    a = (a << 1) % mod;
    b >>= 1;
  }
  return x;
}
lli mod_pow(lli a, lli n, lli mod){
  lli x = 1;
  while(n){
    if(n & 1) x = mod_mult(x, a, mod);
    a = mod_mult(a, a, mod);
```

```cpp
    n >>= 1;
  }
  return x;
}
int trail(lli a,lli b,int n){
   lli ntrail = mod_pow(10,n,10000000000);
   return mod_pow(a,b,ntrail);
}
int leading(lli a,lli b,lli n){
   n--;
   lli nleading = mod_pow(10,n,10000000000);
   return (int)(pow(10, fmod(b*log10(a), 1))*nleading);
}
//_________Euclides extendido___________//
/*
   Solve ax+by = (a,b)
  // Algoritmo de Euclides extendido entre a y b. Ademas de devolver el gcd(a, b), resuelve la ecuación diofantica con el par (x, y).
  //Si el parametro mod no es especificado, se resuelve con aritmetica normal; si mod se especifica, la identidad se resuelve modulo mod.
  [Tested Timus 1141,1204]
*/
int gcd(int a,int b,int &x,int &y){
  if(b==0) {x = 1; y = 0; return a;}
  int r = gcd(b, a%b, y, x);
  y -= a/b*x;
  return r;
}
//________Inverso modular_________//
int inverse(int a, int m){
  int x, y ;
   if isPrime(m)return mod_pow(a,m-2,m);
  if(gcd( a, m, x, y ) != 1) return 0;
  return (x%m + m) % m;
}
/*
   All inverse (1 to p-1)%p
*/
vector<lli> allinverse(lli p){
   vector<lli> ans(p);
   ans[1] = 1;
   for(lli i = 2;i<p;i++){
      ans[i] = p-(p/i)*ans[p%i]%p;
   }
   return ans;
}
//__________Linear Diophantine Equation__________//
/*
```

```cpp
   Use gcd -Extended euclides-
   Solve ax+by=c
   -Find any solution
   -Getting all solutions
   -Finding the number of solutions and the solutions in a given interval
   -Find the solution with minimum value of x+y
   [Tested Spoj - Crucial Equation,SGU 106,Codeforces - Ebony and Ivory, Codechef
   - Get AC in one go, LightOj - Solutions to an equation]
*/
bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
   g = gcd(abs(a), abs(b), x0, y0);
   if (c % g)
      return false;
   x0 *= c / g;
   y0 *= c / g;
   if (a < 0) x0 = -x0;
   if (b < 0) y0 = -y0;
   return true;
}
//___________PHI de euler_____________//
int phi(int n) {
   int result = n;
   for (int i = 2; i * i <= n; i++) {
      if(n % i == 0) {
         while(n % i == 0)
            n /= i;
         result -= result / i;
      }
   }
   if(n > 1)
      result -= result / n;
   return result;
}
//_________GCD and LCM__________//
/*
   [Tested ??]
*/
lli gcd(vector<lli> &nums){
   lli ans =0;
   for(lli &num:nums)ans =__gcd(ans,num);
   return ans;
}
lli lcm(lli a,lli b){
   return b*(a/__gcd(a,b));
}
lli lcm(vector<lli> & nums){
   lli ans = 1;
```

```cpp
   for(lli & num : nums) ans = lcm(ans, num);
   return ans;
}
//_________Teorema chino del residuo____________//
/*
   [Tested ???]
   if p1,p2,p3 are coprime
*/
vector<int>nums;
vector<int>rem;
int CRT() {
   int prod = 1;
   for (int i = 0; i < nums.size(); i++)
      prod *= nums[i];
   int result = 0;
   for (int i = 0; i < nums.size(); i++) {
      int pp = prod / nums[i];
      result += rem[i] * inverse(pp, nums[i]) * pp;
   }
   return result % prod;
}
/*
   general CRT if pi,p2,p3 no coprimes
   return 0 if no solution
*/
inline lli normalize(lli x, lli mod) { x %= mod; if (x < 0) x += mod; return x; }
vector<int> a;
vector<int> n;
lli LCM;
lli CRT(lli &ans){
   int t =a.size();
   ans = a[0];
   LCM = n[0];
   for(int i = 1; i < t; i++){
      int x1,d= gcd(LCM, n[i],x1,d);
      if((a[i] - ans) % d != 0) return 0;
      ans = normalize(ans + x1 * (a[i] - ans) / d % (n[i] / d) * LCM, LCM * n[i] / d);
      LCM = lcm(LCM, n[i]); // you can save time by replacing above LCM * n[i] /d by LCM = LCM * n[i] / d
   }
   return 1;
}
//__________Factorial modulo p___________//
/*
   O(P logp n)
*/
int factmod(int n, int p) {
```

```cpp
    int res = 1;
    while (n > 1) {
        res = (res * ((n/p) % 2 ? p-1 : 1)) % p;
        for (int i = 2; i <= n%p; ++i)
            res = (res * i) % p;
        n /= p;
    }
    return res % p;
}
//________Discrete Log________//
int dlog(int a, int b, int m) {
    int n = (int) sqrt (m + .0) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * a) % m;
    map<int, int> vals;
    for (int p = 1, cur = an; p <= n; ++p) {
        if (!vals.count(cur))
            vals[cur] = p;
        cur = (cur * an) % m;
    }
    for (int q = 0, cur = b; q <= n; ++q) {
        if (vals.count(cur)) {
            int ans = vals[cur] * n - q;
            return ans;
        }
        cur = (cur * a) % m;
    }
    return -1;
}
int powmod(int a, int b, int m) {
    int res = 1;
    while (b > 0) {
        if (b & 1) {
            res = (res * 1ll * a) % m;
        }
        a = (a * 1ll * a) % m;
        b >>= 1;
    }
    return res;
}
int dislog(int a, int b, int m) {
    int n = (int) sqrt (m + .0) + 1;
    map<int, int> vals;
    for (int p = n; p >= 1; --p)
        vals[powmod(a, p * n, m)] = p;
    for (int q = 0; q <= n; ++q) {
        int cur = (powmod(a, q, m) * 1ll * b) % m;
        if (vals.count(cur)) {
            int ans = vals[cur] * n - q;
            return ans;
        }
    }
    return -1;
}
//________Discrete root__________//
int generator(int p) {
    vector<int> fact;
    int phi = p-1, n = phi;
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            fact.push_back(i);
            while (n % i == 0)
                n /= i;
        }
    }
    if (n > 1)
        fact.push_back(n);
    for (int res = 2; res <= p; ++res) {
        bool ok = true;
        for (int factor : fact) {
            if (mod_pow(res, phi / factor, p) == 1) {
                ok = false;
                break;
            }
        }
        if (ok) return res;
    }
    return -1;
}
void discrete_root(int n, int k ,int a){
    int g = generator(n);
    // Baby-step giant-step discrete logarithm algorithm
    int sq = (int) sqrt (n + .0) + 1;
    vector<pair<int, int>> dec(sq);
    for (int i = 1; i <= sq; ++i)
        dec[i-1] = {mod_pow(g, i * sq * k % (n - 1), n), i};
    sort(dec.begin(), dec.end());
    int any_ans = -1;
    for (int i = 0; i < sq; ++i) {
        int my = mod_pow(g, i * k % (n - 1), n) * a % n;
        auto it = lower_bound(dec.begin(), dec.end(), make_pair(my, 0));
        if (it != dec.end() && it->first == my) {
            any_ans = it->second * sq - i;
            break;
        }
    }
    if (any_ans == -1) {
        puts("0");
        return 0;
    }
    // Print all possible answers
    int delta = (n-1) / gcd(k, n-1);
    vector<int> ans;
    for (int cur = any_ans % delta; cur < n-1; cur += delta)
        ans.push_back(mod_pow(g, cur, n));
    sort(ans.begin(), ans.end());
    printf("%d\n", ans.size());
    for (int answer : ans)
        printf("%d ", answer);
}
//_________________________Moudulo operator________________________//
//(A + B) mod C = (A mod C + B mod C) mod C
//(A * B) mod C = (A mod C * B mod C) mod C
// xy (mod a) ≡((x (mod a) * y) (mod a))
// A^B mod M = ( A^(M-1) * A^(M-1) *.......* A^(M-1) * A^(x) ) mod M
// a^(p-1) mod p = 1, When p is prime.
//Mod for negatives,also work in positives ,(a%mod+mod)%mod;
//_______________Trailing zeors in factorial in any base_______________//
/*
  [Tested Codeforces round 538-C]
*/
lli trail_fact(lli n,lli b){
  lli ans = 1000000000000000000LL;
 for (lli i=2; i<=b; i++) {
  if (1LL * i * i > b) i = b;
  int cnt = 0;
  while (b % i == 0) {b /= i; cnt++;}
  if (cnt == 0) continue;
  lli tmp = 0, mul = 1;
  while (mul <= n / i) {mul *= i; tmp += n / mul;}
  ans = min(ans, tmp / cnt);
 }
  return ans;
}
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
```

```cpp
}
bool find_any_solution(int a, int b, int c, int &x0, int &y0, int &g) {
   g = gcd(abs(a), abs(b), x0, y0);
   if (c % g) {
      return false;
   }
   x0 *= c / g;
   y0 *= c / g;
   if (a < 0) x0 = -x0;
   if (b < 0) y0 = -y0;
   return true;
}
void shift_solution(int & x, int & y, int a, int b, int cnt) {
   x += cnt * b;
   y -= cnt * a;
}
int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny
    , int maxy) {
   int x, y, g;
   if (!find_any_solution(a, b, c, x, y, g))
      return -1;
   a /= g;
   b /= g;
   int sign_a = a > 0 ? +1 : -1;
   int sign_b = b > 0 ? +1 : -1;
   shift_solution(x, y, a, b, (minx - x) / b);
   if (x < minx)
      shift_solution(x, y, a, b, sign_b);
   if (x > maxx)
      return -1;
   int lx1 = x;
   shift_solution(x, y, a, b, (maxx - x) / b);
   if (x > maxx)
      shift_solution(x, y, a, b, -sign_b);
   int rx1 = x;
   shift_solution(x, y, a, b, -(miny - y) / a);
   if (y < miny)
      shift_solution(x, y, a, b, -sign_a);
   if (y > maxy)
      return -1;
   int lx2 = x;
   shift_solution(x, y, a, b, -(maxy - y) / a);
   if (y > maxy)
      shift_solution(x, y, a, b, sign_a);
   int rx2 = x;
   if (lx2 > rx2)
      swap(lx2, rx2);
   int lx = max(lx1, lx2);
   int rx = min(rx1, rx2);
   if (lx > rx)
      return -1;
   return lx;
}
// This returns the pair {{min x, max x}, offset} or {{-1,-1},-1} if is
    not solution
pair<pair<int,int>,int> find_all_solutions(int a, int b, int c, int minx
    , int maxx, int miny, int maxy) {
   int x, y, g;
   if (!find_any_solution(a, b, c, x, y, g))
      return {{-1,-1},-1};
   a /= g;
   b /= g;
   int sign_a = a > 0 ? +1 : -1;
   int sign_b = b > 0 ? +1 : -1;
   shift_solution(x, y, a, b, (minx - x) / b);
   if (x < minx)
      shift_solution(x, y, a, b, sign_b);
   if (x > maxx)
      return {{-1,-1},-1};
   int lx1 = x;
   shift_solution(x, y, a, b, (maxx - x) / b);
   if (x > maxx)
      shift_solution(x, y, a, b, -sign_b);
   int rx1 = x;
   shift_solution(x, y, a, b, -(miny - y) / a);
   if (y < miny)
      shift_solution(x, y, a, b, -sign_a);
   if (y > maxy)
      return {{-1,-1},-1};
   int lx2 = x;
   shift_solution(x, y, a, b, -(maxy - y) / a);
   if (y > maxy)
      shift_solution(x, y, a, b, sign_a);
   int rx2 = x;
   if (lx2 > rx2)
      swap(lx2, rx2);
   int lx = max(lx1, lx2);
   int rx = min(rx1, rx2);
   if (lx > rx)
      return {{-1,-1},-1};
   return {{lx,rx},b};
}
int gauss(int n){
   return (n*(n+1))/2;
}
// sum of floor((p*i)/q), 1≤i≤n
int f(int p, int q, int n) {
   int ans = gauss(n) * (p/q);
   cout<<p<<" "<<q<<" "<<n<<" "<<ans<<endl;
   p %= q;
   if (p != 0) {
      int N = (p*n)/q;
      cout<<N<<endl;
      ans += n * N - f(q, p, N) + N/p;
   }
   return ans;
}
// sum of floor((a*i+b)/c), 1≤i≤n
lli f(lli a, lli b, lli c, lli n){
 lli m = (a*n + b)/c;
 if(n==0 || m==0) return b/c;
 if(n==1) return b/c + (a+b)/c;
 if(a<c && b<c) return m*n - f(c, c-b-1, a, m-1);
 else return (a/c)*n*(n+1)/2 + (b/c)*(n+1) + f(a%c, b%c, c, n);
}
```

4.19 PollarRho.cpp

PollarRho.cpp 135 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
#define endl "\n"
#define __ ios_base::sync_with_stdio(false);cin.tie(NULL);
typedef unsigned long long int ull;
typedef long long int lli;
ull mulMod( ull a,ull b,ull m){
   ull res = 0, tmp = a % m;
   while (b){
      if (b & 1){
         res = res + tmp;
         res = (res >= m ? res - m : res);
      }
      b >>= 1;
      tmp <<= 1;
      tmp = (tmp >= m ? tmp - m : tmp);
   }
   return res;
}
lli powMod(lli a,lli b,lli m){
   lli res = 1 % m, tmp = a % m;
   while (b){
      if (b & 1)
         res = mulMod(res, tmp, m);
      tmp = mulMod(tmp, tmp, m);
      b >>= 1;
   }
   return res;
```

```
}
bool millerRabin(lli n){
   int a[5] = { 2, 3, 7, 61, 24251 };
   if (n == 2) return true;
   if (n == 1 || (n & 1) == 0) return false;
   lli b = n - 1;
   for (int i = 0; i < 5; i++){
      if (a[i] >= n) break;
      while ((b & 1) == 0) b >>= 1;
      lli t = powMod(a[i], b, n);
      while (b != n - 1 && t != 1 && t != n - 1){
         t = mulMod(t, t, n);
         b <<= 1;
      }
      if (t == n - 1 || (b & 1)) continue;
      else return false;
   }
   return true;
}
lli gcd(lli a, lli b){
   while (b > 0){
      lli t = a % b;
      a = b;
      b = t;
   }
   return a;
}
lli pollard_rho(lli n){
   lli x = 2 % n, y = x, k = 2, i = 1;
   lli d = 1;
   while (true){
      i++;
      x = (mulMod(x, x, n) + 1) % n;
      d = gcd((y - x + n) % n, n);
      if (d > 1 && d < n)
         return d;
      if (y == x){
         lli d = 2;
         while (n % d) d++;
         return d;
      }
      if (i == k){
         y = x;
         k <<= 1;
      }
   }
}
lli factors[1000], fCount;
void _factorize(lli n){
   if (n <= 1) return;
   if (millerRabin(n)){
      factors[ fCount++ ] = n;
      return;
   }
   lli d = pollard_rho(n);
   _factorize(d);
   _factorize(n/d);
}
void factorize(lli n){
   fCount = 0;
   _factorize(n);
   sort(factors, factors + fCount);
}
vector<lli> getDivs(const vector<pair<lli,lli>> &factors)
{
   int n = SZ(factors);
   int factors_count = 1;
   for (int i = 0; i < n; ++i)
   {
      factors_count *= 1+factors[i].second;
   }
   vector<lli> divs(factors_count); divs[0] = 1;
   int count = 1;
   for (int stack_level = 0; stack_level < n; ++stack_level)
   {
      int count_so_far = count;
      int prime = factors[stack_level].first;
      int exponent = factors[stack_level].second;
      int multiplier = 1;
      for (int j = 0; j < exponent; ++j)
      {
         multiplier *= prime;
         for (int i = 0; i < count_so_far; ++i)
         {
            divs[count++] = divs[i] * multiplier;
         }
      }
   }
   return divs;
}
int main(){__
   int t;
   cin >> t;
   while (t--){
      lli n;
      cin >> n;
      factorize(n);
      for (int i = 0; i < fCount; i++){
         if(i==fCount-1)
            cout<<factors[i];
         else
            cout<<factors[i]<<" ";
      }
   cout<<endl;
   }
   return 0;
}
```

4.20 PollarRho2.cpp

PollarRho2.cpp 97 lines

```
#include <bits/stdc++.h>
#define endl '\n'
#define fi first
#define se second
#define MOD(n,k) ( ( ((n) % abs(k)) + abs(k) ) % abs(k))
#define forn(i,n) for (int i = 0; i < int(n); i++)
#define forr(i,a,b) for (int i = int(a); i <= int(b); i++)
#define all(v) v.begin(), v.end()
#define pb push_back
using namespace std;
typedef long long ll;
typedef long double ld;
typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<vi> vvi;
typedef vector<ii> vii;
using lli = ll;
tuple<lli, lli, lli> extendedGcd(lli a, lli b){
  if(b == 0){
    if(a > 0) return {a, 1, 0};
    else return {-a, -1, 0};
  }else{
    auto[d, x, y] = extendedGcd(b, a%b);
    return {d, y, x - y*(a/b)};
  }
}
lli modularInverse(lli a, lli m){
  auto[d, x, y] = extendedGcd(a, m);
  if(d != 1) return -1;
  if(x < 0) x += m;
  return x;
}
lli powerMod(lli b, lli e, lli m){
  lli ans = 1;
  b %= m;
  if(e < 0){
```

```
    b = modularInverse(b, m);
    e = -e;
  }
  while(e){
    if(e & 1) ans = ans * b % m;
    e >>= 1;
    b = b * b % m;
  }
  return ans;
}
bool isPrimeMillerRabin(lli n){
  if(n < 2) return false;
  if(!(n & 1)) return n == 2;
  lli d = n - 1, s = 0;
  for(; !(d & 1); d >>= 1, ++s);
  for(int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}){
    if(n == a) return true;
    lli m = powerMod(a, d, n);
    if(m == 1 || m == n - 1) continue;
    int k = 0;
    for(; k < s; ++k){
      m = m * m % n;
      if(m == n - 1) break;
    }
    if(k == s) return false;
  }
  return true;
}
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
lli aleatorio(lli a, lli b){
  std::uniform_int_distribution<lli> dist(a, b);
  return dist(rng);
}
lli getFactor(lli n){
  lli a = aleatorio(1, n - 1), b = aleatorio(1, n - 1);
  lli x = 2, y = 2, d = 1;
  while(d == 1){
    x = x * ((x + b) % n) % n + a;
    y = y * ((y + b) % n) % n + a;
    y = y * ((y + b) % n) % n + a;
    d = gcd(abs(x - y), n);
  }
  return d;
}
map<lli, int> fact;
void factorizePollardRho(lli n, bool clean = true){
  if(clean) fact.clear();
  while(n > 1 && !isPrimeMillerRabin(n)){
    lli f = n;
    for(; f == n; f = getFactor(n));
    n /= f;
    factorizePollardRho(f, false);
    for(auto&[p, a] : fact){
      while(n % p == 0){
        n /= p;
        ++a;
      }
    }
  }
  if(n > 1) ++fact[n];
}
```

4.21 PrimeBasis.cpp

PrimeBasis.cpp 116 lines

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define __ ios_base::sync_with_stdio(false),cin.tie(NULL);
#define endl '\n'
template<typename T>
struct PrimeBasis{
  void reduce_pair(T& x, T& y){
    bool to_swap = 0;
    if(x > y){
      to_swap ^= 1;
      swap(x, y);
    }
    while(x > 1 && y % x == 0){
      y /= x;
      if(x > y){
        to_swap ^= 1;
        swap(x, y);
      }
    }
    if(to_swap) swap(x, y);
  }
  vector<T> basis;
  void solve_inner(int pos, T &val){
    while(basis[pos] % val == 0) basis[pos] /= val;
    vector<T> curr_basis = {basis[pos], val};
    int c_ptr = 1;
    while(c_ptr < curr_basis.size()){
      for(int i=0;i<c_ptr;i++){
        reduce_pair(curr_basis[i], curr_basis[c_ptr]);
        if(curr_basis[c_ptr] == 1) break;
        if(curr_basis[i] == 1) continue;
        T g = gcd(curr_basis[c_ptr], curr_basis[i]);
        if(g > 1){
          curr_basis[c_ptr] /= g;
          curr_basis[i] /= g;
          curr_basis.push_back(g);
        }
      }
      ++c_ptr;
    }
    basis[pos] = curr_basis[0];
    val = curr_basis[1];
    for(int i=2;i<curr_basis.size();i++) if(curr_basis[i] > 1) basis.push_back(curr_basis[i]);
    if(basis[pos] == 1){
      swap(basis[pos], basis.back());
      basis.pop_back();
    }
  }
  void add_element(T val){
    for(int i=0;i<basis.size();i++){
      reduce_pair(val, basis[i]);
      if(basis[i] == 1){
        swap(basis[i], basis.back());
        basis.pop_back();
        continue;
      }
      if(val == 1) return;
      if(gcd(basis[i], val) > 1) solve_inner(i, val);
    }
    if(val > 1) basis.push_back(val);
  }
  void verify_basis(){
    for(int i=0;i<basis.size();i++){
      for(int j=i+1;j<basis.size();j++){
        assert(gcd(basis[i], basis[j]) == 1);
      }
    }
  }
  bool verify_element(T ele){
    for(auto &x : basis){
      while(ele % x == 0) ele /= x;
    }
    return (ele == 1);
  }
  auto factorization(T ele){
    vector<int> factors(basis.size());
    for(int i=0;i<basis.size();i++){
      while(ele % basis[i] == 0){
        factors[i]++;
        ele /= basis[i];
```

```
        }
      }
      return factors;
    }
    auto lcm(T a,T b){
      vector<int> lcm(basis.size());
      if(!verify_element(a))
        add_element(a);
      if(!verify_element(b))
        add_element(b);
      vector<int> fa = factorization(a);
      vector<int> fb = factorization(b);
      return lcm(fa,fb);
    }
    auto lcm(vector<int> fa,vector<int> fb){
      vector<int> lcm(basis.size());
      for(int i = 0;i<basis.size();i++){
        lcm[i] = max(fa[i],fb[i]);
      }
      return lcm;
    }
    auto gcd(T a,T b){
      vector<int> gcd(basis.size());
      if(!verify_element(a))
        add_element(a);
      if(!verify_element(b))
        add_element(b);
      vector<int> fa = factorization(a);
      vector<int> fb = factorization(b);
      for(int i = 0;i<basis.size();i++){
        gcd[i] = min(fa[i],fb[i]);
      }
      return gcd;
    }
};
```

4.22 Primes.cpp

Primes.cpp 436 lines

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int lli;
//_______Neds___________//
lli mod_mult(lli a, lli b, lli mod){
  lli x = 0;
  while(b){
    if(b & 1) x = (x + a) % mod;
    a = (a << 1) % mod;
    b >>= 1;
  }
  return x;
}
lli mod_pow(lli a, lli n, lli mod){
  lli x = 1;
  while(n){
    if(n & 1) x = mod_mult(x, a, mod);
    a = mod_mult(a, a, mod);
    n >>= 1;
  }
  return x;
}
//_________Criba de la funcion phi de euler__________//
bool is_composite[MAXN];
int phi[MAXN];
void sieve (int n) {
  std::fill (is_composite, is_composite + n, false);
  phi[1] = 1;
  for (int i = 2; i < n; ++i) {
    if (!is_composite[i]) {
      prime.push_back (i);
      phi[i] = i - 1;     //i is prime
    }
    for (int j = 0; j < prime.size () && i * prime[j] < n; ++j) {
      is_composite[i * prime[j]] = true;
      if (i % prime[j] == 0) {
        phi[i * prime[j]] = phi[i] * prime[j]; //prime[j] divides i
        break;
      } else {
        phi[i * prime[j]] = phi[i] * phi[prime[j]]; //prime[j] does not
        ↪ divide i
      }
    }
  }
}
/*Criba para hallar todos los valores de 1-n de una
función multiplicativa*/
int N = 10000007;
vector<int> F(N+1); //funcion multiplicativa evaluada en i
int g(int p, int a){
  //Ejemplo para la phi de euler
  return power(p, a - 1) * (p - 1);
}
vector<int> sieve(int N){
  vector<int> primes;
  vector<int> lp(N+1); //factor primo mas pequeño de i
  vector<int> cnt(N+1); //exponente del primo mas pequeño de i
  vector<int> pot(N+1); //pow(lp[i], cnt[i])
  F[1] = 1;
  for(int i = 2; i <= N; ++i){
    if(lp[i] == 0){
      primes.push_back(i);
      lp[i] = i;
      F[i] = g(i, 1);
      cnt[i] = 1;
      pot[i] = i;
    }
    for(int p : primes){
      int d = i * p;
      if(d > N) break;
      lp[d] = p;
      if(p == lp[i]){
        F[d] = F[i / pot[i]] * g(p, cnt[i]+1);
        cnt[d] = cnt[i] + 1;
        pot[d] = pot[i] * p;
        break;
      }else{
        F[d] = F[i] * F[p];
        cnt[d] = 1;
        pot[d] = p;
      }
    }
  }
  return f;
}
vector<int> Phi;
void phiSieve(int n){
    Phi.resize(n + 1);
    for(int i = 1; i <= n; ++i)
        Phi[i] = i;
    for(int i = 2; i <= n; ++i)
        if(Phi[i] == i)
            for(int j = i; j <= n; j += i)
                Phi[j] -= Phi[j] / i;
}
//____________***************************________//
vector<int> Mu;
void muSieve(int n){
    Mu.resize(n + 1, -1);
    Mu[0] = 0, Mu[1] = 1;
    for(int i = 2; i <= n; ++i)
        if(Mu[i])
            for(int j = 2*i; j <= n; j += i)
                Mu[j] -= Mu[i];
}
//_________bLOCK SIEVE___________//
int count_primes(int n) {
    const int S = 10000;
```

```
    vector<int> primes;
    int nsqrt = sqrt(n);
    vector<char> is_prime(nsqrt + 1, true);
    for (int i = 2; i <= nsqrt; i++) {
        if (is_prime[i]) {
            primes.push_back(i);
            for (int j = i * i; j <= nsqrt; j += i)
                is_prime[j] = false;
        }
    }
    int result = 0;
    vector<char> block(S);
    for (int k = 0; k * S <= n; k++) {
        fill(block.begin(), block.end(), true);
        int start = k * S;
        for (int p : primes) {
            int start_idx = (start + p - 1) / p;
            int j = max(start_idx, p) * p - start;
            for (; j < S; j += p)
                block[j] = false;
        }
        if (k == 0)
            block[0] = block[1] = false;
        for (int i = 0; i < S && start + i <= n; i++) {
            if (block[i])
                result++;
        }
    }
    return result;
}
//____________________PHI DE EULER_______________________//
//numero de numeros menores a n coprimos con n
int phi (int n){
  int result = n;
  for (int i=2; i*i<=n; ++i)
    if(n %i==0){
      while(n %i==0)
      n /= i;
      result -= result / i;
  }
  if (n > 1)
  result -= result / n;
  return result;
}
//____________Count divisors in n^1/3____________//
/*
    Need miller rabin,criba(),primes[],lp[],N= 10^6
    [Tested Codeforces GYM GCPC-15 F-Divisions]
*/
bool isSquare(lli val){
  lli lo = 1, hi = val;
  while(lo <= hi){
    lli mid = lo + (hi - lo) / 2;
    lli tmp = (val / mid) / mid; // be careful with overflows!!
    if(tmp == 0)hi = mid - 1;
    else if(mid * mid == val)return true;
    else if(mid * mid < val)lo = mid + 1;
  }
  return false;
}
lli countDivisors(lli n) {
    lli ans = 1;
  for(int i = 0; i < primes.size(); i++){
    if(n == 1)break;
    int p = primes[i];
    if(n % p == 0){ // checks whether p is a divisor of n
      int num = 0;
      while(n % p == 0){
        n /= p;
        ++num;
      }
      // p^num divides initial n but p^(num + 1) does not divide initial val
      // => p can be taken 0 to num times => num + 1 possibilities!!
      ans *= num + 1;
    }
  }
    if(n == 1)return ans; // first case
  else if(isPrime(n,20))return ans * 2; // second case
  else if(isSquare(n))return ans * 3; // third case but with p == q
  else return ans * 4; // third case with p != q
}
// arr is a primeFact of n with a pair <prime,exponent>
void generateDivisors(int curIndex, int curDivisor, vector<pair<int,int>>& arr){
    if (curIndex == arr.size()) {
        cout << curDivisor << ' ';
        return;
    }
    for (int i = 0; i <= arr[curIndex].y; ++i) {
        generateDivisors(curIndex + 1, curDivisor, arr);
        curDivisor *= arr[curIndex].x;
    }
}
//_________Prime Factors__________//
map<lli,lli> fact;
void trial_division4(lli n) {
    for (lli d : primes) {
        if (d * d > n)
            break;
        while (n % d == 0) {
            fact[d]++;
            n /= d;
        }
    }
}
void trial_division2(lli n) {
    while (n % 2 == 0) {
        fact[2]++;
        n /= 2;
    }
    for (long long d = 3; d * d <= n; d += 2) {
        while (n % d == 0) {
            fact[d]++;
            n /= d;
        }
    }
    if (n > 1)
        fact[n]++;
}
/*
    Pollard Method p-1
*/
lli pollard_p_1(lli n){
  int b = 13;
  int q[] = {2, 3, 5, 7, 11, 13};
  lli a = 5% n;
  for (int j = 0; j <10; j ++){
    while (__gcd(a, n)!= 1){
      mod_mult (a, a, n);
      a+= 3;
      a%= n;
    }
    for (int i = 0; i<6; i ++){
      int qq = q [i];
      int e = floor(log((double) b) / log((double) qq));
      lli aa = mod_pow(a, mod_pow (qq, e, n), n);
      if (aa == 0)
        continue;
        lli g = __gcd (aa-1, n);
      if (1 <g && g <n)
        return g;
    }
  }
  return 1;
}
/*
```

```
  Pollard rho
*/
lli pollard_rho (lli n, unsigned iterations_count = 100000){
 lli b0 = rand ()% n,b1 = b0,g;
 mod_mult (b1, b1, n);
 if (++b1 == n)
  b1 = 0;
 g = __gcd(abs(b1 - b0), n);
 for (unsigned count = 0; count <iterations_count && (g == 1 || g == n); count ++){
  mod_mult (b0, b0, n);
  if (++ b0 == n)
   b0 = 0;
  mod_mult (b1, b1, n);
  ++ b1;
  mod_mult (b1, b1, n);
  if (++ b1 == n)
   b1 = 0;
  g = __gcd(abs(b1 - b0), n);
 }
 return g;
}
lli pollard_bent (lli n, unsigned iterations_count = 19){
 lli b0 = rand ()% n,
  b1 = (b0 * b0 + 2)% n,
  a = b1;
 for (unsigned iteration = 0, series_len = 1; iteration < iterations_count; iteration ++, series_len *= 2){
  lli g = __gcd(b1-b0, n);
  for (unsigned len = 0; len <series_len && (g == 1 && g == n); len ++) {
   b1 = (b1 * b1 + 2)% n;
   g = __gcd(abs (b1-b0), n);
  }
  b0 = a;
  a = b1;
  if (g != 1 && g != n)
   return g;
 }
 return 1;
}
/*
  Pollard monte Carlo
*/
lli pollard_monte_carlo (lli n, unsigned m = 100){
 lli b = rand ()% (m-2) + 2;
 lli g = 1;
 for (int i = 0; i <100 && g == 1; i++){
  lli cur = primes[i];
  while (cur <= n)
   cur *= primes[i];
  cur /= primes[i];
  b = mod_pow (b, cur, n);
  g = __gcd(abs (b-1), n);
  if (g == n)
   g = 1;
 }
 return g;
}
lli prime_div_trivial (lli n){
 if (n == 2 || n == 3)
  return 1;
 if (n <2)
  return 0;
 if (!n&1)
  return 2;
 lli pi;
 for (auto p:primes){
  if (p*p >n)
   break;
  else
   if (n% p == 0)
    return p;
 }
 if (n <1000*10000)
  return 1;
 return 0;
}
lli ferma (lli n){
 lli x = floor(sqrt((double)n)),y = 0,r = x * x - y * y - n;
 for (;;)
  if (r == 0)
   return x != y? x*y: x + y;
  else
   if (r> 0){
    r-= y + y + 1;
    ++y;
   }
   else{
    r+= x + x + 1;
    ++x;
   }
}
lli mult(lli a, lli b, lli mod) {
  return (lli)a * b % mod;
}
lli f(lli x, lli c, lli mod) {
  return (mult(x, x, mod) + c) % mod;
}
lli brent(lli n, lli x0=2, lli c=1) {
  lli x = x0;
  lli g = 1;
  lli q = 1;
  lli xs, y;
  int m = 128;
  int l = 1;
  while (g == 1) {
    y = x;
    for (int i = 1; i < l; i++)
      x = f(x, c, n);
    int k = 0;
    while (k < l && g == 1) {
      xs = x;
      for (int i = 0; i < m && i < l - k; i++) {
        x = f(x, c, n);
        q = mult(q, abs(y - x), n);
      }
      g = __gcd(q, n);
      k += m;
    }
    l *= 2;
  }
  if (g == n) {
    do {
      xs = f(xs, c, n);
      g = __gcd(abs(xs - y), n);
    } while (g == 1);
  }
  return g;
}
void factorize (lli n){
  if (isPrime(n,20))
    fact[n]++;
  else{
    if (n <1000 * 1000){
      lli div = prime_div_trivial(n);
      fact[div]++;
      factorize(n / div);
    }
    else{
      lli div;
      // 'Pollards fast algorithms come first
      div = pollard_monte_carlo(n);
      if (div == 1)
        div = brent(n);
      if (div == 1)
        div = pollard_rho (n),cout<<"USE POLLAR_RHO\n";
```

```
        if (div == 1)
          div = pollard_p_1 (n),cout<<"USE POLLARD_P_1\n";
        if (div == 1)
          div = pollard_bent (n),cout<<"USE POLLARD_BENT\n";
        if (div == 1)
          div = ferma (n);
        // recursively process the found factors
        factorize (div);
        factorize (n / div);
      }
    }
}
// Get prime factors of a number in time O(log(n)) with precompute array of lowest factor of a number up to 10^7 complexity of precalc is O(nlognlogn)
// uses lowestprime
vector<int> facts;
void factorizeLog(lli n){
    while(n>1){
        facts.push_back(lowestPrime[n]);
        n/=lowestPrime[n];
    }
}
int main(){
  lli n,i,j;
    cin>>n;
  criba();
   // for(int i = 0;i<N;i++){
   //    cout<<primes[i]<<" ";
   // }
   // cin>>n;
   // if(!findPrimes(n))cout<<-1<<endl;
   // cout<<countDivisors(n);
   // factorize(n);
   lowestPrimeSieve(10000000);
   factorize(n);
   factorizeLog(n);
   for(auto c:facts)cout<<c<<" ";
   cout<<endl;
   // cout<<fact.size();
   for(auto c: fact)cout<<c.first<<"^"<<c.second<<" ";
}
```

4.23 SOSConvolutions.cpp

SOSConvolutions.cpp 110 lines

```
#include<bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9, mod = 998244353;
// s' $ s defines all subsets of s
namespace SOS {
const int B = 20; // Every input vector must need to be of size 1<<B
// $z(f(s))=\sum_{s' \subseteq s}{f(s')}$
// $O(B * 2^B)$
// zeta transform is actually SOS DP
vector<int> zeta_transform(vector<int> f) {
  for (int i = 0; i < B; i++) {
    for (int mask = 0; mask < (1 << B); mask++) {
      if ((mask & (1 << i)) != 0) {
        f[mask] += f[mask ^(1 << i)];// you can change the operator from + to min/gcd to find min/gcd of all f[submasks]
      }
    }
  }
  return f;
}
// $mu(f(s))=\sum_{s' $ s}{(-1)^|s\s'| * f(s')}$
// O(B * 2 ^B)
vector<int> mobius_transform(vector<int> f) {
  for (int i = 0; i < B; i++) {
    for (int mask = 0; mask < (1 << B); mask++) {
      if ((mask & (1 << i)) != 0) {
        f[mask] -= f[mask ^(1 << i)];
      }
    }
  }
  return f;
}
vector<int> inverse_zeta_transform(vector<int> f) {
  return mobius_transform(f);
}
vector<int> inverse_mobius_transform(vector<int> f) {
  return zeta_transform(f);
}
// z(f(s))=\sum_{s' is supermask of s}{f(s')}
// O(B * 2 ^B)
// zeta transform is actually SOS DP
vector<int> zeta_transform_for_supermasks(vector<int> f) {
  for (int i = 0; i < B; i++) {
    for (int mask = (1 << B) - 1 ; mask >= 0 ; mask--) {
      if ((mask & (1 << i)) == 0) f[mask] += f[mask ^(1 << i)] ;
    }
  }
  return f;
}
// f*g(s)=sum_{s' $ s} {f(s')*g(s\s')}
// O(B * B * 2 ^B)
vector<int> subset_sum_convolution(vector<int> f, vector<int> g) {
  vector< vector<int> > fhat(B + 1, vector<int> (1 << B, 0));
  vector< vector<int> > ghat(B + 1, vector<int> (1 << B, 0));
  // Make fhat[][] = {0} and ghat[][] = {0}
  for (int mask = 0; mask < (1 << B); mask++) {
    fhat[__builtin_popcount(mask)][mask] = f[mask];
    ghat[__builtin_popcount(mask)][mask] = g[mask];
  }
  // Apply zeta transform on fhat[][] and ghat[][]
  for (int i = 0; i <= B; i++) {
    for (int j = 0; j <= B; j++) {
      for (int mask = 0; mask < (1 << B); mask++) {
        if ((mask & (1 << j)) != 0) {
          fhat[i][mask] += fhat[i][mask ^(1 << j)];
          if (fhat[i][mask] >= mod) fhat[i][mask] -= mod;
          ghat[i][mask] += ghat[i][mask ^(1 << j)];
          if (ghat[i][mask] >= mod) ghat[i][mask] -= mod;
        }
      }
    }
  }
  vector< vector<int> > h(B + 1, vector<int> (1 << B, 0));
  // Do the convolution and store into h[][] = {0}
  for (int mask = 0; mask < (1 << B); mask++) {
    for (int i = 0; i <= B; i++) {
      for (int j = 0; j <= i; j++) {
        h[i][mask] += 1LL * fhat[j][mask] * ghat[i - j][mask] % mod;
        if (h[i][mask] >= mod) h[i][mask] -= mod;
      }
    }
  }
  // Apply inverse SOS dp on h[][]
  for (int i = 0; i <= B; i++) {
    for (int j = 0; j <= B; j++) {
      for (int mask = 0; mask < (1 << B); mask++) {
        if ((mask & (1 << j)) != 0) {
          h[i][mask] -= h[i][mask ^(1 << j)];
          if (h[i][mask] < 0) h[i][mask] += mod;
        }
      }
    }
  }
  vector<int> fog(1 << B, 0);
  for (int mask = 0; mask < (1 << B); mask++) fog[mask] = h[__builtin_popcount(mask)][mask];
  return fog;
}
};
int32_t main() {
  ios_base::sync_with_stdio(0);
```

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

4.24 Sieves.cpp

Sieves.cpp 127 lines

```cpp
// Also gets sieve of function $\mu$
const int N = 1000007;
vector<int> m(N+1);
void sieve(){
  vector<int> lp(N+1);
  vector<int> primes;
  m[1]= 1;
  for(int i = 2;i<=N;i++){
    if(lp[i]== 0){
      primes.push_back(i);
      lp[i]= i;
      m[i] = -1;
    }
    for(int j = 0;j<primes.size()&& primes[j]<=lp[i] && primes[j]*i<=N;j++){
      lp[primes[j]*i] = primes[j];
      if(lp[i]==primes[j])m[primes[j]*i]= 0;
      else m[primes[j]*i] = m[primes[j]]*m[i];
    }
  }
}
// Just get primes and lowest prime factor of each number
const int N = 10000000;
int lp[N+1];
vector<int> primes;
void sieve(){
    for (int i=2; i<=N; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            primes.push_back (i);
        }
        for (int j=0; j<(int)primes.size() && primes[j]<=lp[i] && i*primes[j]<=N; ++j){
            lp[i * primes[j]] = primes[j];
            if(i%primes[j]==0)break;
        }
    }
}
// Sieve with bitset is faster than the sieve with vector also gets the sum of all primes in range [1,n] and the number of primes in range [1,n]
const int NMAX = 100000000;
signed main() {
    bitset<NMAX / 2> bits;
    bits.set();
  auto sum = 2LL;
  int cnt = 1;
    for (int i = 3; i / 2 < bits.size(); i = 2 * bits._Find_next(i / 2) + 1) {
        sum += i;
        ++cnt;
        for (auto j = (int64_t) i * i / 2; j < bits.size(); j += i)
            bits[j] = 0;
    }
  cout << "sum = " << sum << endl;
  cout << "cnt = " << cnt << endl;
  return 0;
}
// Greatest prime sieve
vector<int> gp;
void greatestPrimeSieve(int n){
    gp.resize(n + 1, 1);
    gp[0] = gp[1] = 0;
    for(int i = 2; i <= n; ++i) gp[i] = i;
    for(int i = 2; i <= n; i++)
        if(gp[i] == i)
            for(int j = i; j <= n; j += i)
              gp[j] = i;
    }
vector<int> PrimesInRange;
void SegmentedSieve(int l ,int r) {
    auto sum = l<=2?2:0;
    if(l<=2)PrimesInRange.push_back(2);
    int cnt = 1;
    const int S = round(sqrt(r));
    vector<char> sieve(S + 1, true);
    vector<array<int, 2>> cp;
    for (int i = 3; i <= S; i += 2) {
        if (!sieve[i])
            continue;
        cp.push_back({i, (i * i - 1) / 2});
        for (int j = i * i; j <= S; j += 2 * i){
            sieve[j] = false;
        }
    }
    vector<char> block(S);
    int high = (r - 1) / 2;
    int x = l/S;
    int L = (x/2)*S;
    for(auto &i:cp){
        int p = i[0],idx = i[1];
        if(idx>L){
            i[1]-=L;
        }
        else{
            int X = (L-idx)/p;
            if((L-idx)%p)X++;
            if(X>=1 && idx<=L)
                i[1] = (idx+(p*X))-L;
        }
    }
    for (int low =(x/2)*S; low <= high; low += S) {
        fill(block.begin(), block.end(), true);
        for (auto &i : cp) {
            int p = i[0], idx = i[1];
            for (; idx < S; idx += p){
                block[idx] = false;
            }
            i[1] = idx - S;
        }
        if (low == 0)
            block[0] = false;
        for (int i = 0; i < S && low + i <= high; i++){
            if (block[i] && (((low+i)*2)+1)>=l){
                // push the primes here if needed
                ++cnt, sum += (low + i) * 2 + 1;
            }
        }
    };
   // cout << "sum = " << sum << endl;
   // cout << "cnt = " << cnt << endl;
}
vector<int> Phi;
void phiSieve(int n){
    Phi.resize(n + 1);
    for(int i = 1; i <= n; ++i)
        Phi[i] = i;
    for(int i = 2; i <= n; ++i)
        if(Phi[i] == i)
            for(int j = i; j <= n; j += i)
                Phi[j] -= Phi[j] / i;
}
```

4.25 divisorSigma.cpp

divisorSigma.cpp 33 lines

```cpp
typedef long long ll;
ll divisor_sigma(ll n)
{
  ll sigma = 0, d = 1;
  for (; d * d < n; ++d)
    if (n % d == 0)
      sigma += d + n / d;
  if (d * d == n)
    sigma += d;
  return sigma;
}
// sigma(n) for all n in [lo, hi)
vector<ll> divisor_sigma(ll lo, ll hi)
{
  vector<ll> ps = primes(sqrt(hi) + 1);
  vector<ll> res(hi - lo), sigma(hi - lo, 1);
  iota(res.begin(), res.end(), lo);
  for (ll p : ps)
    for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p)
    {
      ll b = 1;
      while (res[k - lo] > 1 && res[k - lo] % p == 0)
      {
        res[k - lo] /= p;
        b = 1 + b * p;
      }
      sigma[k - lo] *= b;
    }
  for (ll k = lo; k < hi; ++k)
    if (res[k - lo] > 1)
      sigma[k - lo] *= (1 + res[k - lo]);
  return sigma; // sigma[k-lo] = sigma(k)
}
```

4.26 euclid.h

euclid.h 5 lines

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

4.27 eulerPhi.cpp

eulerPhi.cpp 45 lines

```cpp
/*
  Euler Phi (Totient Function)
  Tested: SPOJ ETFS, AIZU NTL_1_D
*/
typedef long long ll;
ll euler_phi(ll n)
{
  if (n == 0)
    return 0;
  ll ans = n;
  for (ll x = 2; x * x <= n; ++x)
    if (n % x == 0)
    {
      ans -= ans / x;
      while (n % x == 0)
        n /= x;
    }
  if (n > 1)
    ans -= ans / n;
  return ans;
}
// phi(n) for all n in [lo, hi)
vector<ll> euler_phi(ll lo, ll hi)
{
  vector<ll> ps = primes(sqrt(hi) + 1);
  vector<ll> res(hi - lo), phi(hi - lo, 1);
  iota(res.begin(), res.end(), lo);
  for (ll p : ps)
    for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p)
    {
      if (res[k - lo] < p)
        continue;
      phi[k - lo] *= (p - 1);
      res[k - lo] /= p;
      while (res[k - lo] > 1 && res[k - lo] % p == 0)
      {
        phi[k - lo] *= p;
        res[k - lo] /= p;
      }
    }
  for (ll k = lo; k < hi; ++k)
    if (res[k - lo] > 1)
      phi[k - lo] *= (res[k - lo] - 1);
  return phi; // phi[k-lo] = phi(k)
}
```

4.28 fastPrimeCount.cpp

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

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

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

fastPrimeCount.cpp c65a97, 123 lines

```cpp
// If sum of primes is needed use int128
#define int __int128
#define MAXN 100
#define MAXM 100007
#define MAXP 10000007
int prime_cnt[MAXP];
int prime_sum[MAXP];
long long dp[MAXN][MAXM];
//Function to print __int128
std::ostream&
operator<<( std::ostream& dest, __int128_t value ){
    std::ostream::sentry s( dest );
    if ( s ) {
        __uint128_t tmp = value < 0 ? -value : value;
        char buffer[ 128 ];
        char* d = std::end( buffer );
        do
        {
            -- d;
            *d = "0123456789"[ tmp % 10 ];
            tmp /= 10;
        } while ( tmp != 0 );
        if ( value < 0 ) {
            -- d;
            *d = '-';
        }
        int len = std::end( buffer ) - d;
        if ( dest.rdbuf()->sputn( d, len ) != len ) {
            dest.setstate( std::ios_base::badbit );
        }
    }
    return dest;
}
vector<int> primes;
bitset<MAXP> is_prime;
// modify k to calc sum of primes p^k with p^k <= n
int k = 0;
int F(int n){
    return pow(n,k);
};
int pref(int n){
    if(k == 0)return n;
    if(k == 1)return (n*(n+1))/2;
    if(k == 2)return (n*(n+1)*(2*n+1))/6;
```

```
   return 1;
}
vector<int> lp(MAXP+1);
void gen(){
   lp.assign(MAXP,0);
   primes.clear();
   for(int i= 2;i<=MAXP;i++){
      if(lp[i]==0)lp[i] = i,primes.push_back(i);
      for(int j = 0;j<primes.size() && primes[j]<=lp[i] && primes[j]*i<=MAXP;j++)
         lp[primes[j]*i] = primes[j];
   }
   for (int i = 2; i < MAXP; i++){
      prime_cnt[i] = prime_cnt[i-1] + (lp[i]==i);
      prime_sum[i] = prime_sum[i-1] + (lp[i]==i?F(i):0);
   }
   for (int m = 0; m < MAXM; m++) dp[0][m] = pref(m);
   for (int n = 1; n < MAXN; n++){
      for (int m = 0; m < MAXM; m++){
         dp[n][m] = dp[n - 1][m] - (dp[n - 1][m/primes[n - 1]]*F(primes[n-1]));
      }
   }
}
int phi(int m, int n){
   if (n == 0) return pref(m);
   if (m < MAXM && n < MAXN) return dp[n][m];
   if (primes[n - 1] * primes[n - 1] >= m && m < MAXP) return prime_sum[m] - pref(n) + 1;
   return phi(m, n - 1) - (phi(m/primes[n - 1], n - 1)*F(primes[n-1]));
}
/*- for some reason this not work for sum of power primes or for k>=1 use count primes instead */
int lehmer(int m){
   if (m < MAXP) return prime_sum[m];
   int s = sqrt(0.5 + m), y = cbrt(0.5 + m);
   int a = prime_cnt[y];
   int res = phi(m, a);
   for (int i = a; primes[i] <= s; i++){
      int x = lehmer(m/primes[i]);
      int y = lehmer(primes[i-1]);
      res = res - ((lehmer(m / primes[i]) - lehmer(primes[i-1]))*F(primes[i])) - F(primes[i]);
   }
   a = prime_sum[s];
   return (res+a)-1;
}
/** Use this function if k>=1 */
int count_primes(int n) {
   vector<int> v;
   v.reserve((int)sqrt(n) * 2 + 20);
   int sq;{
      int k = 1;
      for (; k * k <= n; ++k) {
         v.push_back(k);
      }
      --k;
      sq = k;
      if (k * k == n) --k;
      for (; k >= 1; --k) {
         v.push_back(n / k);
      }
   }
   vector<int> s(v.size());
   for (int i = 0; i < s.size(); ++i)
      s[i] = pref(v[i]) - 1;
   auto geti = [&](int x) {
      if (x <= sq) return (int)x - 1;
      else return (int)(v.size() - (n / x));
   };
   for (int p = 2; p * p <= n; ++p) {
      if (s[p - 1] != s[p - 2]) {
         int sp = s[p - 2];
         int p2 = p * p;
         for (int i = (int)v.size() - 1; i >= 0; --i) {
            if (v[i] < p2) {
               break;
            }
            s[i] -= (s[geti(v[i] / p)] - sp) * F(p);
         }
      }
   }
   return s.back();
}
```

4.29 fastPrimeCount2.cpp

fastPrimeCount2.cpp 239 lines

```
#include <bits/stdtr1c++.h>
// O(n^2/3)
struct _count_primes_struct_t_ {
   vector<int> primes;
   vector<int> mnprimes;
   int ans;
   int y;
   vector<pair<pair<int, int>, char>> queries;
   int count_primes(int n) {
      // this y is actually n/y
      // also no logarithms, welcome to reality, this y is the best for n=10^12 or n=10^13
      y = pow(n, 0.64);
      if (n < 100) y = n;
      // linear sieve
      primes.clear();
      mnprimes.assign(y + 1, -1);
      ans = 0;
      for (int i = 2; i <= y; ++i) {
         if (mnprimes[i] == -1) {
            mnprimes[i] = primes.size();
            primes.push_back(i);
         }
         for (int k = 0; k < primes.size(); ++k) {
            int j = primes[k];
            if (i * j > y) break;
            mnprimes[i * j] = k;
            if (i % j == 0) break;
         }
      }
      if (n < 100) return primes.size();
      int s = n / y;
      for (int p : primes) {
         if (p > s) break;
         ans++;
      }
      // pi(n / y)
      int ssz = ans;
      // F with two pointers
      int ptr = primes.size() - 1;
      for (int i = ssz; i < primes.size(); ++i) {
         while (ptr >= i && (int)primes[i] * primes[ptr] > n)
            --ptr;
         if (ptr < i) break;
         ans -= ptr - i + 1;
      }
      // phi, store all queries
      phi(n, ssz - 1);
      sort(queries.begin(), queries.end());
      int ind = 2;
      int sz = primes.size();
      // the order in fenwick will be reversed, because prefix sum in a fenwick is just one query
      fenwick fw(sz);
      for (auto [na, sign] : queries) {
         auto [n, a] = na;
         while (ind <= n)
            fw.add(sz - 1 - mnprimes[ind++], 1);
         ans += (fw.ask(sz - a - 2) + 1) * sign;
```

```
        }
        queries.clear();
        return ans - 1;
    }
    void phi(int n, int a, int sign = 1) {
        if (n == 0) return;
        if (a == -1) {
            ans += n * sign;
            return;
        }
        if (n <= y) {
            queries.emplace_back(make_pair(n, a), sign);
            return;
        }
        phi(n, a - 1, sign);
        phi(n / primes[a], a - 1, -sign);
    }
    struct fenwick {
        vector<int> tree;
        int n;
        fenwick(int n = 0) : n(n) {
            tree.assign(n, 0);
        }
        void add(int i, int k) {
            for (; i < n; i = (i | (i + 1)))
                tree[i] += k;
        }
        int ask(int r) {
            int res = 0;
            for (; r >= 0; r = (r & (r + 1)) - 1)
                res += tree[r];
            return res;
        }
    };
} _count_primes_struct_;
// O(n^3/4)
int count_primes(int n) {
    auto f = [&](int n) {
        return n;
    };
    auto pref = [&](int n) {
        return (n*(n+1))/2;
    };
    vector<int> v;
    v.reserve((int)sqrt(n) * 2 + 20);
    int sq;{
        int k = 1;
        for (; k * k <= n; ++k) {
            v.push_back(k);
        }
        --k;
        sq = k;
        if (k * k == n) --k;
        for (; k >= 1; --k) {
            v.push_back(n / k);
        }
    }
    for(auto c:v)cout<<c<<" ";
    cout<<endl;
    vector<int> s(v.size());
    for (int i = 0; i < s.size(); ++i)
        s[i] = pref(v[i]) - 1;
    for(auto c:s)cout<<c<<" ";
    cout<<endl;
    auto geti = [&](int x) {
        if (x <= sq) return (int)x - 1;
        else      return (int)(v.size() - (n / x));
    };
    cout<<s[geti(37)]<<endl;
    for (int p = 2; p * p <= n; ++p) {
        cout<<p<<endl;
        if (s[p - 1] != s[p - 2]) {
            int sp = s[p - 2];
            int p2 = p * p;
            cout<<sp<<" "<<p2<<endl;
            for (int i = (int)v.size() - 1; i >= 0; --i) {
                if (v[i] < p2) {
                    break;
                }
                s[i] -= (s[geti(v[i] / p)] - sp) * f(p);
                cout<<"I: "<<i<<" "<<v[i]/p<<" "<<geti(v[i]/p)<<" "<<f(p)<<
                    ↪ endl;
            }
        }
    }
    for(auto c:s)cout<<c<<" ";
    cout<<endl;
    return s.back();
}
int count_primes2(int n) {
    vector<int> v;
    for (int k = 1; k * k <= n; ++k) {
        v.push_back(n / k);
        v.push_back(k);
    }
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
    // for(auto c:v)cout<<c<<" ";
    // cout<<endl;
    int sq = sqrt(n);
    auto geti = [&](int x) {
        if (x <= sq) return (int)x - 1;
        else      return (int)(v.size() - (n / x));
    };
    vector<int> dp(v.size());
    for (int i = 0; i < v.size(); ++i)
        dp[i] = v[i];
    int a = 0;
    for (int p = 2; p * p <= n; ++p) {
        if (dp[geti(p)] != dp[geti(p - 1)]) {
            ++a;
            for (int i = (int)v.size() - 1; i >= 0; --i) {
                if (v[i] < p * p) break;
                dp[i] -= dp[geti(v[i] / p)] - a;
            }
        }
    }
    return dp[geti(n)] - 1;
}
#define MAXN 100
#define MAXM 100010
#define MAXP 10000010
using namespace std;
int prime_cnt[MAXP];
long long dp[MAXN][MAXM];
vector<int> primes;
bitset<MAXP> is_prime;
// void sieve(){
//    is_prime[2] = true;
//    for (int i = 3; i < MAXP; i += 2) is_prime[i] = true;
//    for (int i = 3; i * i < MAXP; i += 2){
//        for (int j = i * i; is_prime[i] && j < MAXP; j += (i << 1)){
//            is_prime[j] = false;
//        }
//    }
//    for (int i = 1; i < MAXP; i++){
//        prime_cnt[i] = prime_cnt[i - 1] + is_prime[i];
//        if (is_prime[i]) primes.push_back(i);
//    }
// }
void sieve(){
    vector<int> lp(MAXP+1);
    for(int i= 2;i<=MAXP;i++){
        if(lp[i]==0)lp[i] = i,primes.push_back(i);
        for(int j = 0;j<primes.size() && primes[j]<=lp[i] && primes[j]*i<=
              ↪ MAXP;j++)
            lp[primes[j]*i] = primes[j];
```

```
    }
    for (int i = 1; i < MAXP; i++)
        prime_cnt[i] = prime_cnt[i - 1] + (lp[i]==i);
}
void gen(){
    sieve();
    for (int m = 0; m < MAXM; m++) dp[0][m] = m;
    for (int n = 1; n < MAXN; n++){
        for (int m = 0; m < MAXM; m++){
            dp[n][m] = dp[n - 1][m] - dp[n - 1][m / primes[n - 1]];
        }
    }
}
long long phi(long long m, int n){
    if (n == 0) return m;
    if (m < MAXM && n < MAXN) return dp[n][m];
    if ((long long)primes[n - 1] * primes[n - 1] >= m && m < MAXP) return
         prime_cnt[m] - n + 1;
    return phi(m, n - 1) - phi(m / primes[n - 1], n - 1);
}
long long lehmer(long long m){
    if (m < MAXP) return prime_cnt[m];
    int s = sqrt(0.5 + m), y = cbrt(0.5 + m);
    int a = prime_cnt[y];
    long long res = phi(m, a) + a - 1;
    for (int i = a; primes[i] <= s; i++){
        res = res - lehmer(m / primes[i]) + lehmer(primes[i]) - 1;
    }
    return res;
}
int main(){
    auto start = clock();
    gen();
    printf("Time taken to generate = %0.6f\n\n", (clock() - start) / (
         double)CLOCKS_PER_SEC);
    cout << lehmer(1e12) << endl;
    printf("\nTime taken = %0.6f\n", (clock() - start) / (double)
         CLOCKS_PER_SEC);
    return 0;
}
```

4.30 hash.cpp

hash.cpp 32 lines

```
#include <bits/stdc++.h>
using namespace std;
lli mod_mult(lli a, lli b){
  lli x = 0;
  while(b){
    if(b & 1) x = (x + a);
    a = (a << 1);
    b >>= 1;
  }
  return x;
}
unsigned int hashh(unsigned int x) {
    x = mod_mult(((x >> 16) ^x) ,0x45d9f3b);
    x = mod_mult(((x >> 16) ^x),0x45d9f3b);
    x = (x >> 16) ^x;
    return x;
}
unsigned int unhash(unsigned int x) {
    x = mod_mult((x >> 16) ^x, 0x119de1f3);
    x = mod_mult((x >> 16) ^x , 0x119de1f3);
    x = (x >> 16) ^x;
    return x;
}
int main(){
    int n;
    set<int> hashes;
    for(int i = 0;i<10000000;i++){
        hashes.insert(hashh(i));
    }
    cout<<hashes.size();
    return 0;
}
```

4.31 isPrime.cpp

isPrime.cpp 78 lines

```
#include <bits/stdc++.h>
using namespace std;
typedef unsigned long long int lli;
lli mod_mult(lli a, lli b, lli mod){
  lli x = 0;
  while(b){
    if(b & 1) x = (x + a) % mod;
    a = (a << 1) % mod;
    b >>= 1;
  }
  return x;
}
lli mod_pow(lli a, lli n, lli mod){
  lli x = 1;
  while(n){
    if(n & 1) x = mod_mult(x, a, mod);
    a = mod_mult(a, a, mod);
    n >>= 1;
  }
  return x;
}
lli random(lli a, lli b) {
    lli intervallLength = b - a + 1;
    int neededSteps = 0;
    lli base = RAND_MAX + 1LL;
    while(intervallLength > 0){
      intervallLength /= base;
      neededSteps++;
    }
    intervallLength = b - a + 1;
    lli result = 0;
    for(int stepsDone = 0; stepsDone < neededSteps; stepsDone++){
        result = (result * base + rand());
    }
    result %= intervallLength;
    if(result < 0) result += intervallLength;
    return result + a;
}
bool witness(lli a, lli n) {
  // check as in Miller Rabin Primality Test described
    lli u = n-1;
    int t = 0;
    while (u % 2 == 0) {
        t++;
        u /= 2;
    }
    lli next = mod_pow(a, u, n);
    if(next == 1)return false;
    lli last;
    for(int i = 0; i < t; i++) {
      last = next;
        next = mod_mult(last, last, n);//(last * last) % n;
        if (next == 1){
          return last != n - 1;
        }
    }
    return next != 1;
}
bool isPrime(lli n, int s) {
    if (n <= 1) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;
    for(int i = 0; i < s; i++) {
        lli a = random(1, n-1);
        if (witness(a, n)) return false;
    }
    return true;
```

```cpp
}
int main(){
  long long int n,t;
  cin>>t;
  while(t--){
    cin>>n;
    if(isPrime(n,20))cout<<"YES"<<endl;
    else cout<<"NO"<<endl;
  }
  return 0;
}
```

4.32 isPrimeFast.cpp

isPrimeFast.cpp 45 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef unsigned long long int ull;
typedef long long int lli;
#define i128 __int128
i128 powerMod(i128 a, i128 b, i128 mod) {
  i128 res = 1;
  while(b) {
    if(b & 1) res = res * a % mod;
    b >>= 1;
    a = a * a % mod;
  }
  return res;
}
bool isPrimeMillerRabin(lli n){
  if(n < 2) return false;
  if(n <= 3) return true;
  if( ~n & 1) return false;
  lli d = n-1, s = 0; //n-1 = 2^s*k
  for(;(~d&1); d>>=1, s++); //d = k
  for(lli a: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37})
  {
    if(n == a) return true;
    i128 residuo = powerMod(a, d, n);
    if(residuo == 1 or residuo == n-1) continue;
    lli x = s;
    while(--x)
    {
      residuo = residuo * residuo % n;
      if(residuo == n-1) break;
    }
    if(x==0) return false;
  }
  return true; //Probability = 1 - (1/4)^size_of(vector_a)
```

```cpp
}
int main(){
  lli n,t;
  cin>>t;
  while(t--){
    cin>>n;
    if(isPrime(n))cout<<"YES"<<endl;
    else cout<<"NO"<<endl;
  }
  return 0;
}
```

4.33 nthFibonacci.cpp

nthFibonacci.cpp 57 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long int lli;
const lli M = 1000000000000000000; // modulo
// Matrix exponentiation method
void multiply(lli F[2][2], lli M[2][2]) {
 lli x = F[0][0]*M[0][0] + F[0][1]*M[1][0];
 lli y = F[0][0]*M[0][1] + F[0][1]*M[1][1];
 lli z = F[1][0]*M[0][0] + F[1][1]*M[1][0];
 lli w = F[1][0]*M[0][1] + F[1][1]*M[1][1];
 F[0][0] = x;
 F[0][1] = y;
 F[1][0] = z;
 F[1][1] = w;
}
void power(lli F[2][2], lli n) {
 if( n == 0 || n == 1)
   return;
 lli M[2][2] = {{1,1},{1,0}};
 power(F, n/2);
 multiply(F, F);
 if (n%2 != 0)
   multiply(F, M);
}
lli fibonacciMatrix(lli n){
 lli F[2][2] = {{1,1},{1,0}};
 if (n == 0)
  return 0;
 power(F, n-1);
 return F[0][0];
}
// Aproximate formula
lli fibonacciApproximation(lli n) {
 double phi = (1 + sqrt(5)) / 2;
 return round(pow(phi, n) / sqrt(5));
}
// Recursive
map<lli, lli> F;
lli fibonacci(lli n) {
 if (F.count(n)) return F[n];
 lli k=n/2;
 if (n%2==0) { // n=2*k
  return F[n] = (fibonacci(k)*fibonacci(k) + fibonacci(k-1)*fibonacci(k-1)) % M;
 } else { // n=2*k+1
  return F[n] = (fibonacci(k)*fibonacci(k+1) + fibonacci(k-1)*fibonacci(k)) % M;
 }
}
// Exclude from here
int main(){
  lli n;
  F[0]=F[1]=1;
  cin>>n;
  cout<<fibonacciApproximation(n)<<endl;
  cout<<fibonacciMatrix(n)<<endl;
  cout<<fibonacci(n-1)<<endl;
  return 0;
}
```

4.34 numericMethods.cpp

numericMethods.cpp 142 lines

```cpp
//_______________________FFT_______________________________//
/*
 [Tested Codeforces 954I ]
*/
const double PI = acos(-1.0L);
using comp = complex<long double>;
using lli = long long int;
typedef vector<comp> vec;
#define print(A) for(auto c : A) cout << c << " ";
#define isZero(z) abs(z.real()) < 1e-3;
int lesspow2(int n){
 int ans = 1;
 while(ans<n)ans<<=1;
 return ans;
}
void fft(vec& a, int inv){
 int n = a.size();
 for(int i = 1,j = 0;i<n-1;i++){
  for(int k = n>>1;(j^= k) <k; k>>= 1);
   if(i<j) swap(a[i],a[j]);
```

```
  }
  for(int k = 1;k<n;k<<=1){
    comp wk = polar(1.0,PI/k*inv);
    for(int i = 0;i<n;i+= k<<1){
      comp w(1);
      for(int j = 0;j<k;j++,w = w*wk){
        comp t = a[i+j+k]*w;
        a[i+j+k] = a[i+j]-t;
        a[i+j] += t;
      }
    }
  }
  if(inv == -1)
    for(int i = 0;i<n;i++)
      a[i]/=n;
}
void fft(vector<cd> &a,int invert){
  int n=a.size();
  for(int i=1,j=0;i<n;i++){
    int z=(n>>1);
    for(;(j&z);z=(z>>1)){
      j=(j^z);
    }
    j=(j^z);
    if(i<j)
    swap(a[i],a[j]);
  }
  for(int len=2;len<=n;len=(len<<1)){
    double ang=(2*PI/len)*((invert?-1:1));
    cd wlen(cos(ang),sin(ang));
    for(int i=0;i<n;i+=len){
      cd w(1);
      for(int j=0;j<len/2;j++){
        cd u=a[i+j],v=a[i+j+len/2]*w;
        a[i+j]=u+v;
        a[i+j+len/2]=u-v;
        w*=wlen;
      }
    }
  }
  if(invert){
    for(int i=0;i<n;i++){
      a[i]/=n;
    }
  }
}
vec multiply(vec &a,vec &b){
  int n0 = a.size()+b.size()-1;
  int n = lesspow2(n0);
  a.resize(n);
  b.resize(n);
  fft(a,1);
  fft(b,1);
  for(int i = 0;i<n;i++)
    a[i]*= b[i];
  fft(a,-1);
  a.resize(n0);
  return a;
}
#include <bits/stdc++.h>
using namespace std;
const double PI = acos(-1.0L);
using lli = int64_t;
#define print(A)for(auto c:A)cout<<c<<" ";cout<<endl;
#define printc(A)for(auto c:A)cout<<c.real()<<" ";cout<<endl;
#define __ ios_base::sync_with_stdio(false);cin.tie(NULL);
#define endl '\n'
int N = 2500007;
int nearestPowerOfTwo(int n){
  int ans = 1;
  while(ans < n) ans <<= 1;
  return ans;
}
lli powerMod(lli b, lli e, lli m){
  lli ans = 1;
  e %= m-1;
  if(e < 0) e += m-1;
  while(e){
    if(e & 1) ans = ans * b % m;
    e >>= 1;
    b = b * b % m;
  }
  return ans;
}
template<int p, int g>
void ntt(vector<int> & X, int inv){
  int n = X.size();
  for(int i = 1, j = 0; i < n - 1; ++i){
    for(int k = n >> 1; (j ^= k) < k; k >>= 1);
    if(i < j) swap(X[i], X[j]);
  }
  vector<lli> wp(n>>1, 1);
  for(int k = 1; k < n; k <<= 1){
    lli wk = powerMod(g, inv * (p - 1) / (k<<1), p);
    for(int j = 1; j < k; ++j)
      wp[j] = wp[j - 1] * wk % p;
    for(int i = 0; i < n; i += k << 1){
      for(int j = 0; j < k; ++j){
        int u = X[i + j], v = X[i + j + k] * wp[j] % p;
        X[i + j] = u + v < p ? u + v : u + v - p;
        X[i + j + k] = u - v < 0 ? u - v + p : u - v;
      }
    }
  }
  if(inv == -1){
    lli nrev = powerMod(n, p - 2, p);
    for(int i = 0; i < n; ++i)
      X[i] = X[i] * nrev % p;
  }
}
template<int p, int g>
void mult(vector<int> &A, vector<int> &B){
  int sz = A.size() + B.size() - 1;
  int size = nearestPowerOfTwo(sz);
  A.resize(size), B.resize(size);
  ntt<p, g>(A, 1);
  for(int i = 0; i < size; i++)
    B[i] = (lli)A[i] * A[i] % p;
  ntt<p, g>(B, -1);
  B.resize(sz);
  // return A;
}
```

4.35 otros.cpp

otros.cpp 24 lines

```
//________________________String to int______________________//
int str_int(string st){
    int n;
    stringstream ss(st);
    ss>>n;
    return n;
}
//___________________________pow of large numbers_______________________________//
long long powerStrings(string sa, string sb){
    // Convert strings to number
    long long a = 0, b = 0;
    for (int i = 0; i < sa.length(); i++)
        a = (a * 10 + (sa[i] - '0')) % MOD;
    for (int i = 0; i < sb.length(); i++)
        b = (b * 10 + (sb[i] - '0')) % (MOD - 1);
    return Mod(a, b);
}
//_______________pow of string________________//
string b = "10000000000000000000000000";
long long remainderB = 0;
```

```cpp
long long MOD = 1000000007;
   // using Fermat Little
for (int i = 0; i < b.length(); i++)
 remainderB = (remainderB * 10 + b[i] - '0') % (MOD - 1);
```

4.36 pewds.cpp

pewds.cpp 446 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define M1 1000000007
#define M2 998244353
#define ll long long
#define ld long double
#define pll pair<ll,ll>
#define REP(i,a,b) for(ll i=a;i<b;i++)
#define REPI(i,a,b) for(ll i=b-1;i>=a;i--)
#define F first
#define S second
#define PB push_back
#define DB pop_back
#define MP make_pair
#define MT make_tuple
#define G(a,b) get<a>(b)
#define V(a) vector<a>
static mt19937 rng(chrono::steady_clock::now().time_since_epoch().count
     ↪ ());
template<typename T>
#define o_set(T) tree<T, null_type,less<T>, rb_tree_tag,
     ↪ tree_order_statistics_node_update>
//member functions :
//1. order_of_key(k) : number of elements strictly lesser than k
//2. find_by_order(k) : k-th element in the set
pll Egcd(ll,ll);
pll Egcd(ll x,ll y)
{
   if(x==0) return MP(0,1);
   pll t=Egcd(y%x,x);
   return MP(t.S-t.F*(y/x),t.F);
}
ll powM(ll x,ll y,ll m)
{
   ll ans=1,r=1;
   x%=m;
   while(r>0&&r<=y)
   {
      if(r&y)
      {
         ans*=x;
         ans%=m;
      }
      r<<=1;
      x*=x;
      x%=m;
   }
   return ans;
}
ll modI(ll a, ll m)
{
   ll m0=m,y=0,x=1;
   if(m==1) return 0;
   while(a>1)
   {
      ll q=a/m;
      ll t=m;
      m=a%m;
      a=t;
      t=y;
      y=x-q*y;
      x=t;
   }
   if(x<0) x+=m0;
   return x;
}
void Miden(ll **p1,ll n)
{
   ll (*x)[n]=(ll(*)[n]) p1;
   REP(i,0,n)
   {
      REP(j,0,n)
      {
         x[i][j]=0;
      }
      x[i][i]=1;
   }
   return;
}
void Mmult(ll **p1,ll **p2,ll **ans,ll x,ll y,ll z,ll m)
{
   ll (*a)[y]=(ll (*)[y])p1;
   ll (*b)[z]=(ll (*)[z])p2;
   ll (*c)[z]=(ll (*)[z])ans;
   REP(i,0,x)
   {
      REP(j,0,z)
      {
         c[i][j]=0;
         REP(k,0,y)
         {
            c[i][j]+=a[i][k]*b[k][j];
            c[i][j]%=m;
         }
      }
   }
   return;
}
void Mpow(ll **p1,ll **ans,ll n,ll y,ll m)
{
   if(y==0)
   {
      Miden(ans,n);
      return;
   }
   ll t[n][n];
   Mpow(p1,(ll **)t,n,y/2,m);
   ll z[n][n];
   Mmult((ll **)t,(ll **)t,(ll **)z,n,n,n,m);
   if(y%2)
   {
      Mmult((ll **)z,p1,ans,n,n,n,m);
   }
   else
   {
      Miden((ll **)t,n);
      Mmult((ll **)z,(ll **)t,ans,n,n,n,m);
   }
   return;
}
bool isprime(ll n)
{
   if(n<2)
      return false;
   for(ll x:{2,3,5,7,11,13,17,19,23,29,31,37})
   {
      if(n==x)
         return true;
      bool flag=true;
      ll r=1;
      ll t=1;
      while(r<=((n-1)>>__builtin_ctzll(n-1)))
      {
         if(r&((n-1)>>__builtin_ctzll(n-1)))
            t=((__int128)t*x)%n;
         x=((__int128)x*x)%n;
```

```
            r<<=1;
        }
        if(t==1||t==n-1)
            flag=false;
        for(r=0;r<__builtin_ctzll(n-1);r++)
        {
            t=((__int128)t*t)%n;
            if(t==n-1)
                flag=false;
        }
        if(flag)
            return false;
    }
    return true;
}
ll PrimRoot(ll p,ll x)
{
    //finds primitive root of prime p greater than x(If it doesnt exist, returns 0)
    V(ll) v;
    ll t=p-1;
    REP(i,2,t+1)
    {
        if(i*i>t) break;
        if(t%i==0)
        {
            v.PB((p-1)/i);
            while(t%i==0)
            {
                t/=i;
            }
        }
    }
    if(t>1) v.PB((p-1)/t);
    REP(i,x+1,p)
    {
        ll flag=0;
        REP(j,0,((ll)v.size()))
        {
            if(powM(i,v[j],p)==1)
            {
                flag=1;
                break;
            }
        }
        if(flag==0)
        {
            return i;
        }
    }
    return 0;
}
void fft(V(ll) &a,ll n,bool invert,ll m,ll x)
{
    REP(i,0,n)
    {
        ll y=0;
        REP(j,0,__builtin_ctzll(n))
        {
            if((1LL<<j)&i)
            {
                y|=(1LL<<(__builtin_ctzll(n)-j-1));
            }
        }
        if(y>i)
        {
            swap(a[i],a[y]);
        }
    }
    if(invert) x=modI(x,m);
    REP(s,1,__builtin_ctzll(n)+1)
    {
        ll y=powM(x,(n/(1LL<<s)),m);
        REP(j,0,(n/(1LL<<s)))
        {
            ll r=1;
            REP(i,0,(1LL<<(s-1)))
            {
                ll u=a[i+j*(1LL<<s)];
                ll v=(r*a[i+j*(1LL<<s)+(1LL<<(s-1))])%m;
                a[i+j*(1LL<<s)]=u+v;
                if(a[i+j*(1LL<<s)]>m) a[i+j*(1LL<<s)]-=m;
                a[i+j*(1LL<<s)+(1LL<<(s-1))]=u-v;
                if(a[i+j*(1LL<<s)+(1LL<<(s-1))]<0) a[i+j*(1LL<<s)+(1LL<<(s-1))]+=m;
                r*=y;
                r%=m;
            }
        }
    }
    if(invert)
    {
        ll invn=modI(n,m);
        REP(i,0,n)
        {
            a[i]=(a[i]*invn)%m;
        }
    }
    return;
}
void PolyMult(V(ll) &a,V(ll) &b,V(ll) &v,ll m,ll x)
{
    ll n=1;
    while(n<((ll)a.size())+((ll)b.size()))
    {
        n<<=1;
    }
    V(ll) fa(a.begin(),a.end());
    fa.resize(n,0);
    V(ll) fb(b.begin(),b.end());
    fb.resize(n,0);
    ll y=powM(x,(m-1)/n,m);
    fft(fa,n,false,m,y);
    fft(fb,n,false,m,y);
    v.resize(n,0);
    REP(i,0,n)
    {
        v[i]=((fa[i]*fb[i])%m);
    }
    fft(v,n,true,m,y);
    v.resize(((ll)a.size())+((ll)b.size())-1,0LL);
    return;
}
void PolyInverse(V(ll) &a,V(ll) &v,ll n,ll m,ll x)
{
    v.clear();
    v.PB(modI(a[0],m));
    while(((ll)v.size())<n)
    {
        ll tmpsz=(((ll)v.size())<<1);
        V(ll) tmpa(tmpsz,0LL);
        REP(i,0,min(((ll)a.size()),tmpsz))
        {
            tmpa[i]=a[i];
        }
        V(ll) tmppr;
        PolyMult(tmpa,v,tmppr,m,x);
        tmppr.resize(tmpsz,0LL);
        REP(i,0,tmpsz)
        {
            tmppr[i]=((M2-tmppr[i])%M2);
        }
        tmppr[0]=((tmppr[0]+2)%M2);
        V(ll) tmpv(v.begin(),v.end());
        PolyMult(tmppr,tmpv,v,m,x);
        v.resize(tmpsz,0LL);
    }
```

```
    v.resize(n,0LL);
    return;
}
void PolyDiv(V(ll) &a,V(ll) &b,V(ll) &q,V(ll) &r,ll m,ll x)
{
    if(((ll)a.size())<((ll)b.size()))
    {
        r=a;
        r.resize(((ll)b.size())-1,0LL);
        q.clear();
        q.PB(0LL);
        return;
    }
    V(ll) ra(((ll)a.size())-((ll)b.size())+1,0LL);
    REP(i,0,((ll)a.size())-((ll)b.size())+1)
    {
        ra[i]=a[((ll)a.size())-1-i];
    }
    V(ll) rb(((ll)b.size()),0LL);
    REP(i,0,((ll)b.size()))
    {
        rb[i]=b[((ll)b.size())-1-i];
    }
    V(ll) irb;
    PolyInverse(rb,irb,((ll)a.size())-((ll)b.size())+1,m,x);
    V(ll) rq;
    PolyMult(ra,irb,rq,m,x);
    rq.resize(((ll)a.size())-((ll)b.size())+1,0LL);
    q.resize(((ll)a.size())-((ll)b.size())+1,0LL);
    REP(i,0,((ll)rq.size()))
    {
        q[i]=rq[((ll)rq.size())-1-i];
    }
    V(ll) tmppr;
    PolyMult(b,q,tmppr,m,x);
    r.resize(((ll)b.size())-1,0LL);
    REP(i,0,((ll)r.size()))
    {
        r[i]=((a[i]+M2-tmppr[i])%M2);
    }
    return;
}
ll fn(ll x,ll rn[])
{
    if(x==rn[x])
        return x;
    else
        return rn[x]=fn(rn[x],rn);
}
bool un(ll x,ll y,ll rn[],ll sz[])
{
    x=fn(x,rn);
    y=fn(y,rn);
    if(x==y)
        return false;
    if(sz[x]<sz[y])
        swap(x,y);
    sz[x]+=sz[y];
    rn[y]=x;
    return true;
}
void build(ll v,ll tl,ll tr,ll st[],ll lz[],bool f[],ll a[])
{
    if(tl==tr)
    {
        st[v]=a[tl];
        lz[v]=0LL;
        f[v]=false;
        return;
    }
    build((v<<1),tl,((tl+tr)>>1),st,lz,f,a);
    build((v<<1)|1,((tl+tr)>>1)+1,tr,st,lz,f,a);
    //operation
    st[v]=st[(v<<1)]+st[(v<<1)|1];
    lz[v]=0LL;
    f[v]=false;
    return;
}
void push(ll v,ll tl,ll tr,ll st[],ll lz[],bool f[])
{
    if(f[v])
    {
        //operation
        st[(v<<1)]=lz[(v<<1)]=st[(v<<1)|1]=lz[(v<<1)|1]=0LL;
        f[(v<<1)]=f[(v<<1)|1]=true;
        f[v]=false;
    }
    //operation
    st[(v<<1)]+=lz[v]*(((tl+tr)>>1)-tl+1);
    //operation
    lz[(v<<1)]+=lz[v];
    //operation
    st[(v<<1)|1]+=lz[v]*(tr-((tl+tr)>>1));
    //operation
    lz[(v<<1)|1]+=lz[v];
    lz[v]=0LL;
    return;
}
void update(ll v,ll tl,ll tr,ll l,ll r,ll val,bool set,ll st[],ll lz[],
    ↪ bool f[])
{
    if(l>r)
    {
        return;
    }
    if(l==tl&&tr==r)
    {
        if(set)
        {
            //operation
            st[v]=lz[v]=0LL;
            f[v]=true;
        }
        //operation
        st[v]+=val*(tr-tl+1);
        //operation
        lz[v]+=val;
        return;
    }
    push(v,tl,tr,st,lz,f);
    update((v<<1),tl,((tl+tr)>>1),l,min(r,((tl+tr)>>1)),val,set,st,lz,f);
    update((v<<1)|1,((tl+tr)>>1)+1,tr,max(l,((tl+tr)>>1)+1),r,val,set,st,
        ↪ lz,f);
    //operation
    st[v]=st[(v<<1)]+st[(v<<1)|1];
    return;
}
ll query(ll v,ll tl,ll tr,ll l,ll r,ll st[],ll lz[],bool f[])
{
    if(l>r)
    {
        return 0LL;
    }
    if(l==tl&&tr==r)
    {
        return st[v];
    }
    push(v,tl,tr,st,lz,f);
    //operation
    return query((v<<1),tl,((tl+tr)>>1),l,min(r,((tl+tr)>>1)),st,lz,f)+
        ↪ query((v<<1)|1,((tl+tr)>>1)+1,tr,max(l,((tl+tr)>>1)+1),r,st,
        ↪ lz,f);
}
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
```

```cpp
    cout.tie(0);
    //freopen("input.txt", "r", stdin);
    //freopen("output.txt", "w", stdout);
    ll ntc=1;
    //cin>>ntc;
    REP(tc,1,ntc+1)
    {
        //cout<<"Case #"<<tc<<": ";
        ll n;
        cin>>n;
        ll a[n];
        REP(i,0,n)
        {
            ll t;
            cin>>t;
            a[i]=t;
        }
        cout<<'\n';
    }
    return 0;
}
```

4.37 segmentedSieve.cpp

segmentedSieve.cpp 72 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define __ ios_base::sync_with_stdio(false),cin.tie(NULL);
#define int long long
vector<int> PrimesInRange;
void calcPrimes(int l ,int r) {
    auto sum = l<=2?2:0;
    if(l<=2)PrimesInRange.push_back(2);
    int cnt = 1;
    const int S = round(sqrt(r));
    vector<char> sieve(S + 1, true);
    vector<array<int, 2>> cp;
    for (int i = 3; i <= S; i += 2) {
        if (!sieve[i])
            continue;
        cp.push_back({i, (i * i - 1) / 2});
        for (int j = i * i; j <= S; j += 2 * i){
            sieve[j] = false;
        }
    }
    vector<char> block(S);
    int high = (r - 1) / 2;
    int x = l/S;
    int L = (x/2)*S;
    for(auto &i:cp){
        int p = i[0],idx = i[1];
        if(idx>L){
            i[1]-=L;
        }
        else{
            int X = (L-idx)/p;
            if((L-idx)%p)X++;
            if(X>=1 && idx<=L)
                i[1] = (idx+(p*X))-L;
        }
    }
    for (int low =(x/2)*S; low <= high; low += S) {
        fill(block.begin(), block.end(), true);
        for (auto &i : cp) {
            int p = i[0], idx = i[1];
            for (; idx < S; idx += p){
                block[idx] = false;
            }
            i[1] = idx - S;
        }
        if (low == 0)
            block[0] = false;
        for (int i = 0; i < S && low + i <= high; i++){
            if (block[i] && (((low+i)*2)+1)>=l){
                // push the primes here if needed
                ++cnt, sum += (low + i) * 2 + 1;
            }
        }
    };
  // cout << "sum = " << sum << endl;
  // cout << "cnt = " << cnt << endl;
}
signed main(){__
    int l,r,t,id = 1;
    cin>>t;
    while(t--){
        if(id>1)cout<<endl;
        PrimesInRange.clear();
        cin>>l>>r;
        calcPrimes(l,r);
        // for(auto c:primes)
        //    cout<<c<<endl;
        id++;
    }
  return 0;
}
```

graph (5)

5.1 2sat.cpp

2sat.cpp 70 lines

```cpp
int N,timeD ;
vector<vector<int> > gr(1007);
vector<int> val, comp, z;
vector<int> values; // 0 = false, 1 = true
void addCondition(int u, int v,int Nu,int Nv) {
  (u*=2)^=Nu;
  (v*=2)^=Nv;
  gr[u^1].push_back(v);
  gr[v^1].push_back(u);
}
// 0 -> 00
// 1 -> 01
// 2 -> 10
// 3 -> 11
// For must be same mask = 9 -> 1001
// for must be diferent mask = 6 -> 0110 ./
void canBe(int u,int v,int mask){
  if(!(mask&1)){
    addCondition(u,v,1,1);
  }
  if(!((mask>>1)&1)){
    addCondition(u,v,1,0);
  }
  if(!((mask>>2)&1)){
    addCondition(u,v,0,1);
  }
  if(!((mask>>3)&1)){
    addCondition(u,v,0,0);
  }
}
void cannotBe(int u,int v,int mask){
  if((mask&1)){
    addCondition(u,v,0,0);
  }
  if(((mask>>1)&1)){
    addCondition(u,v,0,1);
  }
  if(((mask>>2)&1)){
    addCondition(u,v,1,0);
  }
  if(((mask>>3)&1)){
    addCondition(u,v,1,1);
  }
}
```

```cpp
int dfs(int i) {
  int low = val[i] = ++timeD, x; z.push_back(i);
  for(int j = 0;j<gr[i].size();j++){
    int e = gr[i][j];
    if (!comp[e])
      low = min(low, val[e] ?: dfs(e));
  }
  ++timeD;
  if (low == val[i]) do {
    x = z.back(); z.pop_back();
    comp[x] = timeD;
    if (values[x>>1] == -1)
      values[x>>1] = !(x&1);
  } while (x != i);
  return val[i] = low;
}
bool solve() {
  values.assign(N, -1);
  val.assign(2*N, 0); comp = val;
  timeD = 0;
  for(int i = 0;i<2*N;i++)
        if(!comp[i])
            dfs(i);
  for(int i = 0;i<N;i++) if (comp[2*i] == comp[2*i+1]) return 0;
  return 1;
}
```

5.2 Arborescence2.cpp

Arborescence2.cpp 72 lines

```cpp
/*
  Minimum Arborescence (Chu-Liu/Edmonds)
  Tested: UVA 11183
  Complexity: O(mn)
*/
template<typename T>
struct minimum_aborescense
{
  struct edge
  {
    int src, dst;
    T weight;
  };
  vector<edge> edges;
  void add_edge(int u, int v, T w)
  {
    edges.push_back({ u, v, w });
  }
  T solve(int r)
  {
    int n = 0;
    for (auto e : edges)
      n = max(n, max(e.src, e.dst) + 1);
    int N = n;
    for (T res = 0;;)
    {
      vector<edge> in(N, { -1, -1, numeric_limits<T>::max() });
      vector<int> C(N, -1);
      for (auto e : edges) // cheapest comming edges
        if (in[e.dst].weight > e.weight)
          in[e.dst] = e;
      in[r] = {r, r, 0};
      for (int u = 0; u < N; ++u)
      { // no comming edge ==> no aborescense
        if (in[u].src < 0)
          return numeric_limits<T>::max();
        res += in[u].weight;
      }
      vector<int> mark(N, -1); // contract cycles
      int index = 0;
      for (int i = 0; i < N; ++i)
      {
        if (mark[i] != -1)
          continue;
        int u = i;
        while (mark[u] == -1)
        {
          mark[u] = i;
          u = in[u].src;
        }
        if (mark[u] != i || u == r)
          continue;
        for (int v = in[u].src; u != v; v = in[v].src)
          C[v] = index;
        C[u] = index++;
      }
      if (index == 0)
        return res; // found arborescence
      for (int i = 0; i < N; ++i) // contract
        if (C[i] == -1)
          C[i] = index++;
      vector<edge> next;
      for (auto &e : edges)
        if (C[e.src] != C[e.dst] && C[e.dst] != C[r])
          next.push_back({ C[e.src], C[e.dst],
            e.weight - in[e.dst].weight });
      edges.swap(next);
      N = index;
      r = C[r];
    }
  }
};
```

5.3 BellmanFord.h

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

BellmanFord.h 830a8f, 21 lines

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

5.4 CentroidDecomposition.cpp

Description: Decomposes a tree into centroids and create a new tree with the centroids
Time: $\mathcal{O}(n \log n)$

CentroidDecomposition.cpp b5bb88, 65 lines

```cpp
const int maxn = 100005;
vector<int> graph[maxn];
vector<int> parent(maxn,-1);
vector<int> depth(maxn,-1);
vector<int> best(maxn,1e16);
int P[maxn][25];
bitset<maxn> cent;
int sz[maxn];
void dfs (int u, int p = -1,int d = 0) {
  sz[u] = 1;
```

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

5.5 DFSMatching.h

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

DFSMatching.h — 8b9b78, 63 lines

```
bool find(int j, vector<vector<int>>& g, vector<int>& btoa, vector<int>& vis) {
 if (btoa[j] == -1) return 1;
 vis[j] = 1; int di = btoa[j];
 for (int e : g[di])
  if (!vis[e] && find(e, g, btoa, vis)) {
   btoa[e] = di;
   return 1;
  }
 return 0;
}
int dfsMatching(vector<vector<int>>& g, vector<int>& btoa) {
 vector<int> vis;
 for(int i = 0;i<g.size();i++) {
  vis.assign(btoa.size(), 0);
  for (int j : g[i])
   if (find(j, g, btoa, vis)) {
    btoa[j] = i;
    break;
   }
 }
 return btoa.size() - (int)count(btoa.begin(),btoa.end(), -1);
}
const int maxn = 100007;
vector<int> graph[maxn];
// If you graph is not dividen in proper way (not divided in two sets L and R) call this function
vector<int> ConverLR(int n){
  vector<bool> vis(n);
  vector<int> color(n);
  auto bfsColor = [&](int s){
    vector<int> q;
    q.push_back(s);
    while(q.size()){
      int u = q.back();
      vis[u] = true;
      q.pop_back();
      for(auto v:graph[u]){
        if(!vis[v]){
          q.push_back(v);
          color[v] = color[u]^1;
        }
      }
    }
  };
  for(int i = 0;i<n;i++)
    if(!vis[i])
      bfsColor(i);
  map<int,int> mpR;
  int m= 0,key = 0;
  vector<vector<int>> g;
  for(int i = 0;i<n;i++){
    if(color[i]){
      g.push_back(vector<int>());
      for(auto d:graph[i]){
        if(!mpR.count(d))
          mpR[d] = key++;
        g.back().push_back(mpR[d]);
      }
    }
    else m++;
  }
 vector<int> match(m, -1);
  return dfsMatching(g,match);
}
```

5.6 DSURollback.cpp

DSURollback.cpp — 78 lines

```
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define int long long
#define __ ios_base::sync_with_stdio(false);cin.tie(NULL);
const int N = 3e5 + 7;
int p[N], sz[N], ans;
stack<int> st;
int n, k, u[N << 1], v[N << 1], o[N << 1];
char op[N << 1];
map<pair<int,int>, int> mp;
int find(int u) {
  while(p[u] != u) u = p[u]; // Notice: No path compression. Path Compression will make the algorithm O(n^2)
  return u;
```

```
}
void Union(int u, int v) {
    u = find(u); v = find(v);
    if(u == v) return;
    if(sz[u] > sz[v]) swap(u, v);
    p[u] = v;
    sz[v] += sz[u];
    ans--;
    st.push(u);
}
void rollbax(int t) {
    while(st.size() > t) {
        int u = st.top(); st.pop();
        sz[p[u]] -= sz[u];
        p[u] = u; ans++;
    }
}
void solve(int l, int r) {
    if(l == r) {
        if(op[l] == '?') cout<<ans<<endl;
        return;
    }
    int m = l + r >> 1, now = st.size();
    for(int i = m + 1; i <= r; i++)
        if(o[i] < l) Union(u[i], v[i]);
    solve(l, m);
    rollbax(now);
    for(int i = l; i <= m; i++)
        if(o[i] > r) Union(u[i], v[i]);
    solve(m + 1, r);
    rollbax(now);
}
signed main() {
    freopen("connect.in", "r", stdin);
    freopen("connect.out", "w", stdout);
    cin>>n>>k;
    for(int i = 1; i <= n; i++)
        p[i] = i, sz[i] = 1;
    if(!k) return 0;
    for(int i = 1; i <= k; i++) {
        cin>>op[i];
        if(op[i] == '?') continue;
        cin>>u[i]>>v[i];
        if(u[i] > v[i])
            swap(u[i], v[i]);
        pair<int,int> x(u[i], v[i]);
        if(mp.count(x)) {
            o[i] = mp[x];
            o[o[i]] = i;
            mp.erase(x);
        } else {
            mp[x] = i;
        }
    }
    int idx = k;
    for(auto it : mp) {
        o[it.second] = ++idx;
        o[idx] = it.second;
        op[idx] = '-';
        tie(u[idx], v[idx]) = it.first;
    }
    ans = n;
    solve(1, idx);
}
```

5.7 DSUTree.cpp

DSUTree.cpp 37 lines

```
void dfs_size(int v, int p) {
 sz[v] = 1;
 for (auto u : adj[v]) {
  if (u != p) {
   dfs_size(u, v);
   sz[v] += sz[u];
  }
 }
}
vector<int> *vec[maxn];
int cnt[maxn];
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
          mx = sz[u], bigChild = u;
    for(auto u : g[v])
       if(u != p && u != bigChild)
          dfs(u, v, 0);
    if(bigChild != -1)
        dfs(bigChild, v, 1), vec[v] = vec[bigChild];
    else
        vec[v] = new vector<int> ();
    vec[v]->push_back(v);
    cnt[ col[v] ]++;
    for(auto u : g[v])
       if(u != p && u != bigChild)
           for(auto x : *vec[u]){
               cnt[ col[x] ]++;
               vec[v] -> push_back(x);
           }
    //now cnt[c] is the number of vertices in subtree of vertex v that has color c.
    // note that in this step *vec[v] contains all of the subtree of vertex v.
    if(keep == 0)
        for(auto u : *vec[v])
            cnt[ col[u] ]--;
}
```

5.8 Dijkstra.cpp

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

Dijkstra.cpp 07389d, 62 lines

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

```
        s.insert(make_pair(D[u], u));
      }
    }
  }
}
// A bit faster
int dijkstra(int ini, int fin, int n){
  vector<int> dist(n,INF);
  dist[ini] = 0;
  priority_queue<pair<int, int>, vector<pair<int,int>>, greater<pair<int, int>> > pq;
  pq.push({0, ini});
  while (pq.size() != 0) {
    int minVal = pq.top().first;
    int idx = pq.top().second;
    pq.pop();
    if(dist[idx] < minVal)continue;
    for(auto arista: graph[idx]){
      int newDist = dist[idx] + arista.second ;
      if(newDist < dist[arista.first]){
        dist[arista.first] = newDist;
        pq.push({newDist, arista.first});
      }
    }
    // uncomment this if you only need the distance to one node
    //if(idx == fin)return dist[fin]+inc;
  }
  return -1;
}
```

5.9 Dinic.h

Dinic.h 56 lines

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

5.10 DirectedMST.h

Description: Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.
Time: $\mathcal{O}(E \log V)$

DirectedMST.h 833b20, 59 lines

```
#include "../data-structures/UnionFindRollback.h"
struct Edge { int a, b; ll w; };
struct Node {
  Edge key;
  Node *l, *r;
  ll delta;
  void prop() {
    key.w += delta;
    if (l) l->delta += delta;
    if (r) r->delta += delta;
    delta = 0;
  }
  Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
  if (!a || !b) return a ?: b;
  a->prop(), b->prop();
  if (a->key.w > b->key.w) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }
pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
  RollbackUF uf(n);
  vector<Node*> heap(n);
  for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
  ll res = 0;
  vi seen(n, -1), path(n), par(n);
  seen[r] = r;
  vector<Edge> Q(n), in(n, {-1,-1}), comp;
  deque<tuple<int, int, vector<Edge>>> cycs;
  rep(s,0,n) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
      if (!heap[u]) return {-1,{}};
      Edge e = heap[u]->top();
      heap[u]->delta -= e.w, pop(heap[u]);
      Q[qi] = e, path[qi++] = u, seen[u] = s;
      res += e.w, u = uf.find(e.a);
      if (seen[u] == s) {
        Node* cyc = 0;
        int end = qi, time = uf.time();
        do cyc = merge(cyc, heap[w = path[--qi]]);
        while (uf.join(u, w));
        u = uf.find(u), heap[u] = cyc, seen[u] = -1;
        cycs.push_front({u, time, {&Q[qi], &Q[end]}});
      }
    }
    rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
  }
  for (auto& [u,t,comp] : cycs) { // restore sol (optional)
```

```
  uf.rollback(t);
  Edge inEdge = in[u];
  for (auto& e : comp) in[uf.find(e.b)] = e;
  in[uf.find(inEdge.b)] = inEdge;
 }
 rep(i,0,n) par[i] = in[i].a;
 return {res, par};
}
```

5.11 DominatorTree.cpp

DominatorTree.cpp — 86 lines

```
#include<bits/stdc++.h>
using namespace std;
const int N = 2e5 + 9;
vector<int> g[N];
vector<int> t[N], rg[N], bucket[N]; //t = dominator tree of the nodes reachable from root
int sdom[N], par[N], idom[N], dsu[N], label[N];
int id[N], rev[N], T;
int find_(int u, int x = 0) {
 if(u == dsu[u]) return x ? -1 : u;
 int v = find_(dsu[u], x+1);
 if(v < 0)return u;
 if(sdom[label[dsu[u]]] < sdom[label[u]]) label[u] = label[dsu[u]];
 dsu[u] = v;
 return x ? v : label[u];
}
void dfs(int u) {
 T++; id[u] = T;
 rev[T] = u; label[T] = T;
 sdom[T] = T; dsu[T] = T;
 for(int i = 0; i < g[u].size(); i++) {
  int w = g[u][i];
  if(!id[w]) dfs(w), par[id[w]] = id[u];
  rg[id[w]].push_back(id[u]);
 }
}
void build(int r, int n) {
 dfs(r);
 n = T;
 for(int i = n; i >= 1; i--) {
  for(int j = 0; j < rg[i].size(); j++) sdom[i] = min(sdom[i], sdom[find_(rg[i][j])]);
  if(i > 1) bucket[sdom[i]].push_back(i);
  for(int j = 0; j < bucket[i].size(); j++) {
   int w = bucket[i][j];
   int v = find_(w);
   if(sdom[v] == sdom[w]) idom[w] = sdom[w];
   else idom[w] = v;
  }
  if(i > 1) dsu[i] = par[i];
 }
 for(int i = 2; i <= n; i++) {
  if(idom[i] != sdom[i]) idom[i]=idom[idom[i]];
  t[rev[i]].push_back(rev[idom[i]]);
  t[rev[idom[i]]].push_back(rev[i]);
 }
}
int st[N], en[N];
void yo(int u, int pre = 0) {
 st[u] = ++T;
 for(auto v: t[u]) {
  if(v == pre) continue;
  yo(v, u);
 }
 en[u] = T;
}
int main() {
 ios_base::sync_with_stdio(0);
 cin.tie(0);
 int n, m;
 while(cin >> n >> m) {
  vector<pair<int, int>> ed;
  for(int i = 0; i < m; i++) {
   int u, v;
   cin >> u >> v;
   g[u].push_back(v);
   ed.push_back({u, v});
  }
  build(1, n);
  T = 0;
  yo(1);
  vector<int> ans;
  for(int i = 0; i < m; i++) {
   int u = ed[i].first, v = ed[i].second;
   if(st[u] && !(st[v] <= st[u] && en[u] <= en[v])) ans.push_back(i);
  }
  yo(1);
  cout << ans.size() << '\n';
  for(auto x: ans) cout << x + 1 << ' ';
  cout << '\n';
  T = 0;
  for(int i = 0; i <= n; i++) {
   t[i].clear(), g[i].clear(), rg[i].clear(), bucket[i].clear();
   sdom[i] = par[i] = idom[i] = dsu[i] = label[i] = id[i] = rev[i] = st[i] = en[i] = 0;
  }
 }
 return 0;
}
```

5.12 EdgeColoring.h

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

EdgeColoring.h — ec3455, 31 lines

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

5.13 EdmondsKarp.h

EdmondsKarp.h — 71 lines

```
vector<vector<int>> capacity;
vector<vector<int>> graph;
```

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

5.14 EulerWalk.h

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

EulerWalk.h — 780b64, 15 lines

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
  int n = sz(gr);
  vi D(n), its(n), eu(nedges), ret, s = {src};
  D[src]++; // to allow Euler paths, not just cycles
  while (!s.empty()) {
    int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
    if (it == end){ ret.push_back(x); s.pop_back(); continue; }
    tie(y, e) = gr[x][it++];
    if (!eu[e]) {
      D[x]--, D[y]++;
      eu[e] = 1; s.push_back(y);
    }}
  for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
  return {ret.rbegin(), ret.rend()};
}
```

5.15 FloydWarshall.h

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

FloydWarshall.h — ccbb1e, 17 lines

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

5.16 GeneralMatching.h

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

GeneralMatching.h — e91247, 57 lines

```
const int maxn = 507;
vector<int> graph[maxn];
vector<int> Blossom(int n) {
    int timer = -1;
    vector<int> mate(n, -1), label(n), parent(n), orig(n), aux(n, -1), q;
    auto lca = [&](int x, int y) {
        for (timer++; ; swap(x, y)) {
            if (x == -1) continue;
            if (aux[x] == timer) return x;
            aux[x] = timer;
            x = (mate[x] == -1 ? -1 : orig[parent[mate[x]]]);
        }
    };
    auto blossom = [&](int v, int w, int a) {
        while (orig[v] != a) {
            parent[v] = w; w = mate[v];
            if (label[w] == 1) label[w] = 0, q.push_back(w);
            orig[v] = orig[w] = a; v = parent[w];
        }
    };
```

```cpp
  auto augment = [&](int v) {
    while (v != -1) {
      int pv = parent[v], nv = mate[pv];
      mate[v] = pv;
      mate[pv] = v;
      v = nv;
    }
  };
  auto bfs = [&](int root) {
    fill(label.begin(), label.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    q.clear();
    label[root] = 0; q.push_back(root);
    for (int i = 0; i < (int)q.size(); ++i) {
      int v = q[i];
      for (auto x : graph[v]) {
        if (label[x] == -1) {
          label[x] = 1;
          parent[x] = v;
          if (mate[x] == -1)
            return augment(x), 1;
          label[mate[x]] = 0; q.push_back(mate[x]);
        }
        else if (label[x] == 0 && orig[v] != orig[x]) {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a);
          blossom(v, x, a);
        }
      }
    }
    return 0;
  };
  for (int i = 0; i < n; i++)
    if (mate[i] == -1)
      bfs(i);
  return mate;
}
```

5.17 GlobalMinCut.h

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

GlobalMinCut.h — f206fe, 21 lines

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

5.18 GomoryHu.h

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

GomoryHu.h - "PushRelabel.h" — 0418b3, 13 lines

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

5.19 HLD.cpp

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most log(n) light edges.
Time: $\mathcal{O}\left((\log N)^2\right)$ per query in a path

HLD.cpp - "SegmentTree.cpp" — d18d4e, 61 lines

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

```cpp
    update(1,0,n-1,stpos[head[a]],stpos[a],val);
    a = p[head[a]];
  }
  if(lvl[a]> lvl[b])
    swap(a,b);
  update(1,0,n-1,stpos[a],stpos[b],val);
}
```

5.20 Hungarian.cpp

Hungarian.cpp 46 lines

```cpp
/*
 Tested: TIMUS 1833
 Complexity: O(n^3)
*/
// max weight matching
template<typename T>
T hungarian(const vector<vector<T>> &a)
{
 int n = a.size(), p, q;
 vector<T> fx(n, numeric_limits<T>::min()), fy(n, 0);
 vector<int> x(n, -1), y(n, -1);
 for (int i = 0; i < n; ++i)
  for (int j = 0; j < n; ++j)
   fx[i] = max(fx[i], a[i][j]);
 for (int i = 0; i < n;)
 {
  vector<int> t(n, -1), s(n + 1, i);
  for (p = q = 0; p <= q && x[i] < 0; ++p)
   for (int k = s[p], j = 0; j < n && x[i] < 0; ++j)
    if (fx[k] + fy[j] == a[k][j] && t[j] < 0)
    {
     s[++q] = y[j], t[j] = k;
     if (s[q] < 0)
      for (p = j; p >= 0; j = p)
       y[j] = k = t[j], p = x[k], x[k] = j;
    }
  if (x[i] < 0)
  {
   T d = numeric_limits<T>::max();
   for (int k = 0; k <= q; ++k)
    for (int j = 0; j < n; ++j)
     if (t[j] < 0)
      d = min(d, fx[s[k]] + fy[j] - a[s[k]][j]);
   for (int j = 0; j < n; ++j)
    fy[j] += (t[j] < 0 ? 0 : d);
   for (int k = 0; k <= q; ++k)
    fx[s[k]] -= d;
  }
  else
   ++i;
 }
 T ret = 0;
 for (int i = 0; i < n; ++i)
  ret += a[i][x[i]];
 return ret;
}
```

5.21 LCA.cpp

Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.
Time: $\mathcal{O}\left(N \log N + Qlog(n)\right)$

LCA.cpp 8ce7ee, 182 lines

```cpp
const int maxn = 100007;
const int mxlog = 25;
vector<int> graph[maxn];
int parent[mxlog][maxn];
vector<int> deep(maxn);
int N;
void add_edge(int u,int v){
   graph[u].push_back(v);
   graph[v].push_back(u);
}
void dfs(int u,int p = -1,int d = 0){
   deep[u] = d;
   parent[0][u] = p;
   for(auto v:graph[u]){
      if(v== p)continue;
      dfs(v,u,d+1);
   }
}
void build(){
   for(int i = 0;i<N;i++)for(int j = 0;j<mxlog;j++)parent[j][i] = -1;
   for(int i = 0;i<N;i++)deep[i]= -1;
   dfs(0);
   for(int i = 0;i<N;i++)
      if(deep[i]== -1)dfs(i);
   for(int i = 1;i<mxlog;i++){
      for(int u = 0;u<N;u++){
         if(parent[i-1 ][u]!= -1)
         parent[i][u] = parent[i-1][parent[i-1][u]];
      }
   }
}
// extra: minimum node in the path
const int mxlog = 20;
int P[mxlog][300000];
int MN[mxlog][300000];
vector<int> deep(300000);
vector<int> dist(300000);
int N;
void dfsLCA(int u,int p = -1,int d = 0){
   deep[u] = d;
   P[0][u] = p;
   MN[0][u] = dist[u];
   if(p!=-1)MN[0][u] =min(MN[0][u],dist[p]);
   for(auto v:graph[u]){
      if(v== p)continue;
      dfsLCA(v,u,d+1);
   }
}
int lca(int u ,int v){
   if(deep[u]>deep[v])swap(u,v);
   int diff = deep[v]-deep[u];
   for(int i = mxlog-1;i>=0;i--){
      if(diff & (1<<i))
         v = parent[i][v];
   }
   if(u == v)return u;
   for(int i = mxlog-1;i>=0;i--){
      if(parent[i][u]!= parent[i][v]){
         u = parent[i][u];
         v = parent[i][v];
      }
   }
   return parent[0][u];
}
void build(){
   for(int i = 0;i<N;i++)for(int j = 0;j<mxlog;j++)P[j][i] = -1;
   for(int i = 0;i<N;i++)deep[i]= -1;
   dfsLCA(0);
   for(int i = 0;i<N;i++)
      if(deep[i]== -1)dfsLCA(i);
   for(int i = 1;i<mxlog;i++){
      for(int u = 0;u<N;u++){
         if(P[i-1 ][u]!= -1){
            P[i][u] = P[i-1][P[i-1][u]];
            MN[i][u] = min(MN[i-1][u],MN[i-1][P[i-1][u]]);
         }
      }
   }
}
int lca(int u ,int v){
   if(deep[u]>deep[v])swap(u,v);
   int diff = deep[v]-deep[u];
   int mn = min(dist[u],dist[v]);
```

```cpp
    // cout<<"LCA: "<<u<<" "<<v<<" "<<mn<<endl;
    for(int i = mxlog-1;i≥0;i--){
        if(diff & (1<<i)){
            mn = min(mn,MN[i][v]);
            // cout<<mn<<endl;
            v = P[i][v];
        }
    }
    if(u == v)return mn;
    for(int i = mxlog-1;i≥0;i--){
        if(P[i][u]≠ P[i][v]){
            mn = min(mn,min(MN[i][u],MN[i][v]));
            // cout<<mn<<endl;
            u = P[i][u];
            v = P[i][v];
        }
    }
    mn = min(mn,min(MN[0][u],MN[0][v]));
    return mn;
}
// extra: OR with node values in the path u->v
const int LOG = 20;
const int maxn = 200007;
vector<int> graph[maxn];
vector<vector<int>> parent(maxn, vector<int>());
vector<vector<int>> orBin(maxn, vector<int>());
vector<int> depth(maxn);
vector<int> a(maxn);
void dfs(int u, int p, int d) {
    parent[u][0] = p;
    orBin[u][0] = a[u];
    if (p ≠ -1) {
        orBin[u][0] |= a[p];
    }
    depth[u] = d;
    for (int v : graph[u]) {
        if (v == p) continue;
        dfs(v, u, d + 1);
    }
}
pair<int, int> lca(int u, int v) {
    if (depth[u] < depth[v]) {
        swap(u, v);
    }
    int diff = depth[u] - depth[v];
    int OR = a[u] | a[v];
    for (int x = LOG - 1; x ≥ 0; --x) {
        if ((diff >> x) & 1) {
            OR |= orBin[u][x];
            u = parent[u][x];
        }
    }
    if (u == v) {
        return {OR, u};
    }
    for (int x = LOG - 1; x ≥ 0; --x) {
        if (parent[u][x] ≠ parent[v][x]) {
            OR |= orBin[u][x];
            OR |= orBin[v][x];
            u = parent[u][x];
            v = parent[v][x];
        }
    }
    OR |= orBin[u][0] | orBin[v][0];
    return {OR, parent[u][0]};
}
int dist(int u, int v, int LCA) {
    return depth[u] + depth[v] - 2 * depth[LCA];
}
int getParent(int u, int x) {
    for (int i = LOG - 1; i ≥ 0; --i) {
        if ((x >> i) & 1) {
            u = parent[u][i];
        }
    }
    return u;
}
int getNode(int u, int v, int LCA, int x) {
    if (dist(u, LCA, LCA) ≥ x) {
        return getParent(u, x);
    } else {
        return getParent(v, dist(v, LCA, LCA) - (x - dist(u, LCA, LCA)));
    }
}
// Usage:
// Cleaning
for (int i = 0; i < n; ++i) {
    graph[i].clear();
    parent[i] = vector<int>(LOG, -1);
    orBin[i] = vector<int>(LOG, 0);
}
dfs(0, -1, 0);
for (int u = 1; u < LOG; ++u) {
    for (int i = 0; i < n; ++i) {
        if (parent[i][u - 1] == -1)continue;
        parent[i][u] = parent[parent[i][u - 1]][u - 1];
        orBin[i][u] = orBin[i][u - 1] | orBin[parent[i][u - 1]][u - 1];
    }
}
```

5.22 LCA.py

Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected this version also calc the the min/max of the path from u to v.
Time: $\mathcal{O}(N \log N + Q)$

LCA.py 98 lines

```python
LOG = 10
parent = [[-1]*LOG for _ in range(n)]
mxBin = [[0]*LOG for _ in range(n)]
depth = [0]*n
def dfs(u,p = -1,mx = 0,d = 0):
    parent[u][0] = p
    mxBin[u][0] = mx
    depth[u] = d
    for v,w,id in graph[u]:
        if v == p: continue
        dfs(v,u,w,d+1)
dfs(0)
for u in range(1,LOG):
    for i in range(n):
        if parent[i][u-1]==-1: continue
        parent[i][u] = parent[parent[i][u-1]][u-1]
        mxBin[i][u] = max(mxBin[i][u-1],mxBin[parent[i][u-1]][u-1])
def lca(u,v):
    if depth[u]<depth[v]: u,v = (v,u)
    diff = depth[u]-depth[v]
    mx = 0
    for x in range(LOG-1,-1,-1):
        if (diff>>x) &1:
            mx = max(mx,mxBin[u][x])
            u = parent[u][x]
    if u == v:
        return (mx,u)
    for x in range(LOG-1,-1,-1):
        if parent[u][x] ≠ parent[v][x]:
            mx = max(mx,mxBin[u][x])
            mx = max(mx,mxBin[v][x])
            u = parent[u][x]
            v = parent[v][x]
    mx = max(mx,mxBin[u][0])
    mx = max(mx,mxBin[v][0])
    return (mx,parent[u][0])
# OR version and some util functions
graph = [[] for i in range(n)]
for i in range(n-1):
u,v = map(int,input().split())
u-=1
```

```
v-=1
graph[u].append(v)
graph[v].append(u)
LOG = 20
parent = [[-1]*LOG for _ in range(n)]
orBin = [[0]*LOG for _ in range(n)]
depth = [0]*n
def dfs(s):
   stack = [(s,-1,0)]
   while stack:
   u,p,d = stack.pop()
   parent[u][0] = p
   orBin[u][0] = a[u]
   if p != -1:
      orBin[u][0] |= a[p]
   depth[u] = d
   for v in graph[u]:
      if v == p: continue
      stack.append((v,u,d+1))
dfs(0)
for u in range(1,LOG):
   for i in range(n):
      if parent[i][u-1]==-1: continue
      parent[i][u] = parent[parent[i][u-1]][u-1]
      orBin[i][u] = orBin[i][u-1] | orBin[parent[i][u-1]][u-1]
def lca(u,v):
   if depth[u]<depth[v]: u,v = (v,u)
   diff = depth[u]-depth[v]
   OR = a[u]|a[v]
   for x in range(LOG-1,-1,-1):
      if (diff>>x) &1:
         OR |= orBin[u][x]
         u = parent[u][x]
   if u == v:
      return (OR,u)
   for x in range(LOG-1,-1,-1):
      if parent[u][x] != parent[v][x]:
         OR |= mxBin[u][x]
         OR |= mxBin[v][x]
         u = parent[u][x]
         v = parent[v][x]
   OR |= orBin[u][0] | orBin[v][0]
   return (OR,parent[u][0])
def dist(u,v,LCA):
return depth[u] + depth[v] - 2*depth[LCA]
# get parent at x distance from u
def getParent(u,x):
for i in range(LOG-1,-1,-1):
   if (x>>i)&1:
      u = parent[u][i]
return u
# get node at x distance from u in the path from u to v
def getNode(u,v,LCA,x):
if dist(u,LCA,LCA) >= x:
   return getParent(u,x)
else:
   return getParent(v,dist(v,LCA,LCA)-(x-dist(u,LCA,LCA)))
```

5.23 LCASparseTable.cpp

Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.
Time: $\mathcal{O}(N \log N + Q), \mathcal{O}(1)\, perquery$

LCASparseTable.cpp a376d8, 52 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define __ ios_base::sync_with_stdio(false),cin.tie(NULL);
#define endl '\n'
const int maxn = 200105;
const int mxlog = 25;
vector<int> graph[maxn];
vector<int> level[maxn];
vector<int> euler, first,height;
pair<int,int> ST[maxn][25];
int lg[maxn];
void dfs(int u, int h = 0,int p = -1) {
   first[u] = euler.size();
   height[u] = h;
   level[h].push_back(u);
   euler.push_back(u);
   for (auto v : graph[u]) {
      if (v == p) continue;
      dfs(v, h + 1,u);
      euler.push_back(u);
   }
}
void build(vector<int> &A,int n){
   lg[1] = 0;
   for(int i = 2;i<maxn;i++)
      lg[i] = lg[i/2]+1;
   for(int i = 0;i<n;i++)
      ST[i][0] = {height[A[i]],A[i]};
   for(int j = 1;j<25;j++)
      for(int i=0; i+(1<<j)<=n; i++)
         ST[i][j] = min(ST[i][j-1],ST[i+(1<<(j-1))][j-1]);
}
void LCA(int n, int root = 0) {
   first.resize(n);
   height.resize(n);
   euler.reserve(n * 2);
   dfs(root);
   int m = euler.size();
   build(euler,m);
}
int query(int l, int r){
   int j = lg[r-l+1];
   return min(ST[l][j],ST[r-(1<<j)+1][j]).second;
}
int lca(int u, int v){
   int left = first[u], right = first[v];
   // cout<<left<<" "<<right<<endl;
   if (left > right)
      swap(left, right);
   return query(left, right);
}
```

5.24 LinkCutTree.h

Description: Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.
Time: All operations take amortized $\mathcal{O}(\log N)$.

LinkCutTree.h 3de263, 89 lines

```cpp
struct SplayTree {
  struct Node {
    int ch[2] = {0, 0}, p = 0;
    long long self = 0, path = 0; // Path aggregates
    long long sub = 0, vir = 0;  // Subtree aggregates
    bool flip = 0;               // Lazy tags
  };
  vector<Node> T;
  SplayTree(int n) : T(n + 1) {}
  void push(int x) {
    if (!x || !T[x].flip) return;
    int l = T[x].ch[0], r = T[x].ch[1];
    T[l].flip ^= 1, T[r].flip ^= 1;
    swap(T[x].ch[0], T[x].ch[1]);
    T[x].flip = 0;
  }
  void pull(int x) {
    int l = T[x].ch[0], r = T[x].ch[1]; push(l); push(r);
    T[x].path = T[l].path + T[x].self + T[r].path;
    T[x].sub = T[x].vir + T[l].sub + T[r].sub + T[x].self;
  }
  void set(int x, int d, int y) {
    T[x].ch[d] = y; T[y].p = x; pull(x);
  }
```

```
 void splay(int x) {
  auto dir = [&](int x) {
   int p = T[x].p; if (!p) return -1;
   return T[p].ch[0] == x ? 0 : T[p].ch[1] == x ? 1 : -1;
  };
  auto rotate = [&](int x) {
   int y = T[x].p, z = T[y].p, dx = dir(x), dy = dir(y);
   set(y, dx, T[x].ch[!dx]);
   set(x, !dx, y);
   if (~dy) set(z, dy, x);
   T[x].p = z;
  };
  for (push(x); ~dir(x); ) {
   int y = T[x].p, z = T[y].p;
   push(z); push(y); push(x);
   int dx = dir(x), dy = dir(y);
   if (~dy) rotate(dx != dy ? x : y);
   rotate(x);
  }
 }
};
struct LinkCut : SplayTree {
 LinkCut(int n) : SplayTree(n) {}
 int access(int x) {
  int u = x, v = 0;
  for (; u; v = u, u = T[u].p) {
   splay(u);
   int& ov = T[u].ch[1];
   T[u].vir += T[ov].sub;
   T[u].vir -= T[v].sub;
   ov = v; pull(u);
  }
  return splay(x), v;
 }
 void reroot(int x) {
  access(x); T[x].flip ^= 1; push(x);
 }
 void Link(int u, int v) {
  reroot(u); access(v);
  T[v].vir += T[u].sub;
  T[u].p = v; pull(v);
 }
 void Cut(int u, int v) {
  reroot(u); access(v);
  T[v].ch[0] = T[u].p = 0; pull(v);
 }
 // Rooted tree LCA. Returns 0 if u and v arent connected.
 int LCA(int u, int v) {
  if (u == v) return u;
  access(u); int ret = access(v);
  return T[u].p ? ret : 0;
 }
 // Query subtree of u where v is outside the subtree.
 long long Subtree(int u, int v) {
  reroot(v); access(u); return T[u].vir + T[u].self;
 }
 // Query path [u..v]
 long long Path(int u, int v) {
  reroot(u); access(v); return T[v].path;
 }
 // Update vertex u with value v
 void Update(int u, long long v) {
  access(u); T[u].self = v; pull(u);
 }
};
```

5.25 MaximalCliques.h

Description: A clique is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent.Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.
Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs

MaximalCliques.h b0d5b1, 12 lines

```
typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R={}) {
 if (!P.any()) { if (!X.any()) f(R); return; }
 auto q = (P | X)._Find_first();
 auto cands = P & ~eds[q];
 rep(i,0,sz(eds)) if (cands[i]) {
  R[i] = 1;
  cliques(eds, f, P & eds[i], X & eds[i], R);
  R[i] = P[i] = 0; X[i] = 1;
 }
}
```

5.26 MaximumClique.h

Description: a cliqeu is a subsets of vertices, all adjacent to each other, also called complete subgraphs, Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.
Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

MaximumClique.h f7c0bc, 49 lines

```
typedef vector<bitset<200>> vb;
struct Maxclique {
 double limit=0.025, pk=0;
 struct Vertex { int i, d=0; };
 typedef vector<Vertex> vv;
 vb e;
 vv V;
 vector<vi> C;
 vi qmax, q, S, old;
 void init(vv& r) {
  for (auto& v : r) v.d = 0;
  for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
  sort(all(r), [](auto a, auto b) { return a.d > b.d; });
  int mxD = r[0].d;
  rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
 }
 void expand(vv& R, int lev = 1) {
  S[lev] += S[lev - 1] - old[lev];
  old[lev] = S[lev - 1];
  while (sz(R)) {
   if (sz(q) + R.back().d <= sz(qmax)) return;
   q.push_back(R.back().i);
   vv T;
   for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
   if (sz(T)) {
    if (S[lev]++ / ++pk < limit) init(T);
    int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
    C[1].clear(), C[2].clear();
    for (auto v : T) {
     int k = 1;
     auto f = [&](int i) { return e[v.i][i]; };
     while (any_of(all(C[k]), f)) k++;
     if (k > mxk) mxk = k, C[mxk + 1].clear();
     if (k < mnk) T[j++].i = v.i;
     C[k].push_back(v.i);
    }
    if (j > 0) T[j - 1].d = 0;
    rep(k,mnk,mxk + 1) for (int i : C[k])
     T[j].i = i, T[j++].d = k;
    expand(T, lev + 1);
   } else if (sz(q) > sz(qmax)) qmax = q;
   q.pop_back(), R.pop_back();
  }
 }
 vi maxClique() { init(V), expand(V); return qmax; }
 Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
  rep(i,0,sz(e)) V.push_back({i});
 }
};
```

5.27 MaximumIndependentSet.h

MaximumIndependentSet.h 1 lines

5.28 MinCostMaxFlow.h

Description: Min-cost max-flow. cap[i][j] != cap[j][i] is allowed; double edges are not. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.
Time: Approximately $\mathcal{O}\left(E^2\right)$

MinCostMaxFlow.h 6890f1, 69 lines

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

5.29 MinCut.h

MinCut.h 1 lines

5.30 MinimumVertexCover.h

MinimumVertexCover.h 23 lines

```
vector<int> cover(vector<vector<int>>& g, int n, int m) {
 //From maximum matrching de minimun vertex cover is the nodes not
      ↪ matched for the left side
  // and the nodes visited in the right part if we run a dfs from no
      ↪ matched nodes in the left j
  Bipartite_Matching BM(n,m,g);
  int res = BM.bipartite_matching();
  vector<int> match = BM.match;
 vector<bool> lfound(n, true), seen(m);
 for (int it : match) if (it != -1) lfound[it] = false;
 vector<int> q, cover;
 for(int i = 0;i<n;i++) if (lfound[i]) q.push_back(i);
 while (!q.empty()) {
  int i = q.back(); q.pop_back();
  lfound[i] = 1;
  for (int e : g[i]) if (!seen[e] && match[e] != -1) {
   seen[e] = true;
   q.push_back(match[e]);
  }
 }
 for(int i = 0;i<n;i++) if (!lfound[i]) cover.push_back(i);
 for(int i = 0;i<m;i++) if (seen[i]) cover.push_back(n+i);
 assert(cover.size() == res);
 return cover;
}
```

5.31 PushRelabel.h

Description: Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.
Time: $\mathcal{O}\left(V^2\sqrt{E}\right)$

PushRelabel.h 0ae1d4, 45 lines

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

```
      H[u] = 1e9;
      for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest]+1)
        H[u] = H[e.dest]+1, cur[u] = &e;
      if (++co[H[u]], !--co[hi] && hi < v)
        rep(i,0,v) if (hi < H[i] && H[i] < v)
          --co[H[i]], H[i] = v + 1;
      hi = H[u];
    } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
      addFlow(*cur[u], min(ec[u], cur[u]->c));
    else ++cur[u];
  }
 }
 bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

5.32 SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.

Usage: `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.

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

SCC.h 827511, 96 lines

```
vi val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F& f) {
 int low = val[j] = ++Time, x; z.push_back(j);
 for (auto e : g[j]) if (comp[e] < 0)
  low = min(low, val[e] ?: dfs(e,g,f));
 if (low == val[j]) {
  do {
   x = z.back(); z.pop_back();
   comp[x] = ncomps;
   cont.push_back(x);
  } while (x != j);
  f(cont); cont.clear();
  ncomps++;
 }
 return val[j] = low;
}
template<class G, class F> void scc(G& g, F f) {
 int n = sz(g);
 val.assign(n, 0); comp.assign(n, -1);
 Time = ncomps = 0;
 rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
#include <bits/stdc++.h>
using namespace std;
#define UNVISITED 0
vector<int> dfs_num, dfs_low, S, visited;
vector<vector<int>> AdjList;
int dfsNumberCounter;
int n, m, numSCC;
unordered_map<int, int> components;
void tarjanSCC(int u) {
   dfs_low[u] = dfs_num[u] = dfsNumberCounter++;
   S.push_back(u);
   visited[u] = 1;
   for(int j = 0; j < (int)AdjList[u].size(); j++) {
      auto v = AdjList[u][j];
      if(dfs_num[v] == UNVISITED)
         tarjanSCC(v);
      if(visited[v])
         dfs_low[u] = min(dfs_low[u], dfs_low[v]);
   }
   if(dfs_low[u] == dfs_num[u]) {
      numSCC++;
      while(1) {
         int v = S.back(); S.pop_back(); visited[v] = 0;
         components[v] = numSCC;
         if(u==v) break;
      }
   }
}
typedef long long ll;
typedef vector<ll> vll;
int main() {
   ios_base::sync_with_stdio(false);
   cin.tie(nullptr);
   cin>>n>>m;
   AdjList.resize(n + 1);
   for(int i = 0; i < m; i++) {
      int x, y;
      cin>>x>>y;
      AdjList[x].push_back(y);
   }
   dfs_num.assign(n+1, UNVISITED); dfs_low.assign(n+1, 0); visited.
       ↪ assign(n+1, 0);
   dfsNumberCounter = numSCC = 0;
   for(int i = 1; i <= n; i++) {
      //cout << i << ' ' << numSCC << endl;
      if(dfs_num[i] == UNVISITED) tarjanSCC(i);
   }
   vll in;
   vll out;
   in.resize(numSCC+1);
   out.resize(numSCC+1);
   for(int i = 1; i<= n; i++){
      //cout << i << endl;
      for(size_t j = 0; j < AdjList[i].size(); j++){
         ll pa = components[i];
         ll pb = components[AdjList[i][j]];
         //cout << i << ' ' << AdjList[i][j] << ' ' << pa << ' ' << pb
             ↪ << endl;
         if(pa == pb) continue;
         in[pb]++;
         out[pa]++;
      }
   }
   ll a1 = 0;
   ll a2 = 0;
   for(int i = 1; i < numSCC+1; i++){
      if(in[i] == 0) a1++;
      if(out[i] == 0) a2++;
   }
   if(numSCC == 1) {
      cout << 0 << endl;
      return 0;
   }
   cout << max(a1,a2) << endl;
}
```

5.33 SCCTarjan.cpp

SCCTarjan.cpp 52 lines

```
#include <bits/stdc++.h>
using namespace std;
#define __ ios_base::sync_with_stdio(false),cin.tie(NULL);
#define endl '\n'
const int maxn = 100007;
vector<int> graph[maxn], graph_rev[maxn];
vector<bool> used;
vector<int> order, component;
void dfs1(int v) {
   used[v] = true;
   for (auto u : graph[v])
      if (!used[u])
         dfs1(u);
   order.push_back(v);
}
void dfs2(int v) {
   used[v] = true;
   component.push_back(v);
   for (auto u : graph_rev[v])
      if (!used[u])
         dfs2(u);
```

```
}
signed main(){__
    int n,m,u,v;
    cin>>n>>m;
    vector<int> in(n);
    vector<int> out(n);
    for(int i = 0;i<m;i++){
        cin>>u>>v;
        u--,v--;
        graph[u].push_back(v);
        graph_rev[v].push_back(u);
    }
    used.assign(n, false);
    for (int i = 0; i < n; i++)
        if (!used[i])
        dfs1(i);
    used.assign(n, false);
    reverse(order.begin(), order.end());
    vector<int> roots(n, 0);
    vector<int> root_nodes;
    vector<vector<int>> graphC(n);
    for (auto v : order){
        if (!used[v]) {
            dfs2(v);
            int root = component.front();
            for (auto u : component) roots[u] = root;
            root_nodes.push_back(root);
            component.clear();
        }
    }
}
```

5.34 TopoSort.cpp

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

TopoSort.cpp 3966d8, 71 lines

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

5.35 WeightedMatching.h

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

WeightedMatching.h c97b86, 31 lines

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

```cpp
  }
  for(int j = 1;j<m;j++)if (p[j]) ans[p[j] - 1] = j - 1;
  return {-v[0], ans}; // min cost
}
```

5.36 articulationPoints.cpp

articulationPoints.cpp 54 lines

```cpp
struct graph{
  int n;
  vector<vector<int>> adj;
  graph(int n) : n(n), adj(n) {}
  void add_edge(int u, int v){
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  int add_node(){
    adj.push_back({});
    return n++;
  }
  vector<int>& operator[](int u) { return adj[u]; }
};
vector<vector<int>> biconnected_components(graph &adj){
  int n = adj.n;
  vector<int> num(n), low(n), art(n), stk;
  vector<vector<int>> comps;
  function<void(int, int, int&)> dfs = [&](int u, int p, int &t){
    num[u] = low[u] = ++t;
    stk.push_back(u);
    for (int v : adj[u]) if (v != p){
      if (!num[v]){
        dfs(v, u, t);
        low[u] = min(low[u], low[v]);
        if (low[v] >= num[u]){
          art[u] = (num[u] > 1 || num[v] > 2);
          comps.push_back({u});
          while (comps.back().back() != v)
            comps.back().push_back(stk.back()),
            stk.pop_back();
        }
      }
      else low[u] = min(low[u], num[v]);
    }
  };
  for (int u = 0, t; u < n; ++u){
    if (!num[u]) dfs(u, -1, t = 0);
  }
  function<graph()> build_tree = [&](){
    graph tree(0);
    vector<int> id(n);
    for (int u = 0; u < n; ++u)
      if (art[u]) id[u] = tree.add_node();
    for (auto &comp : comps){
      int node = tree.add_node();
      for (int u : comp)
        if (!art[u]) id[u] = node;
        else tree.add_edge(node, id[u]);
    }
    return tree;
  };
  return comps;
}
```

5.37 blockCutTree.cpp

blockCutTree.cpp 79 lines

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

```
}
```

5.38 bridgeTree.cpp

bridgeTree.cpp 70 lines

```
const lli maxn = 200007;
set<lli> graph[maxn];
set<lli> graph2[maxn];
vector<lli> low(maxn),d(maxn),label(maxn),bridge(maxn),vis(maxn),parent(maxn),sz(maxn);
lli idx;
void dfs(lli u,lli p = -1){
   d[u] =idx++;
   low[u] = d[u];
   vis[u] = true;
   sz[u]= 1;
   parent[u] = p;
   for(auto v:graph[u]){
      if(v == p)continue;
      if(!vis[v]){
         dfs(v,u);
         sz[u]+=sz[v];
         if(low[v]>d[u]) bridge[v] = true;
      }
      low[u] = min(low[u],low[v]);
   }
}
void dfs_label(lli u){
   vis[u] = 1;
   label[u] = idx;
   for(auto v : graph[u])
      if(!vis[v])
         dfs_label(v);
}
int main(){
 int t= 1,n,u,v;
   cin>>t;
   while(t--){
      cin>>n;
      for(lli i = 0;i<n;i++)graph[i].clear(),bridge[i] = false,vis[i] = false;
      for(lli i = 0;i<n;i++){
         cin>>u>>v;
         u--,v--;
         graph[u].insert(v);
         graph[v].insert(u);
      }
      dfs(0);
      vector<pair<lli,lli>> bridges;
      for(lli i = 0;i<n;i++){
         vis[i] = false;
         if(bridge[i]){
            graph[i].erase(parent[i]);
            graph[parent[i]].erase(i);
            if(i)bridges.push_back({i,parent[i]});
         }
      }
      idx = 0;
      for(lli i = 0;i<n;i++){
         if(!vis[i]){
            dfs_label(i);
            idx++;
         }
      }
      for(lli i = 0;i<n;i++)
         graph[i].clear(),vis[i] = false;
      for(auto v:bridges){
         lli a = label[v.first],b = label[v.second];
         graph[a].insert(b);
         graph[b].insert(a);
      }
      // Now graph has the bridge tree
      // use label to know the component of each node
      // use another vector if you want to keep the original graph
   }
   return 0;
}
```

5.39 cycle.cpp

cycle.cpp 44 lines

```
int n;
vector<vector<int>> adj;
vector<char> color;
vector<int> parent;
int cycle_start, cycle_end;
bool dfs(int v) {
   color[v] = 1;
   for (int u : adj[v]) {
      if (color[u] == 0) {
         parent[u] = v;
         if (dfs(u))
            return true;
      } else if (color[u] == 1) {
         cycle_end = v;
         cycle_start = u;
         return true;
      }
   }
   color[v] = 2;
   return false;
}
void find_cycle() {
   color.assign(n, 0);
   parent.assign(n, -1);
   cycle_start = -1;
   for (int v = 0; v < n; v++) {
      if (color[v] == 0 && dfs(v))
         break;
   }
   if (cycle_start == -1) {
      cout << "Acyclic" << endl;
   } else {
      vector<int> cycle;
      cycle.push_back(cycle_start);
      for (int v = cycle_end; v != cycle_start; v = parent[v])
         cycle.push_back(v);
      cycle.push_back(cycle_start);
      reverse(cycle.begin(), cycle.end());
      cout << "Cycle found: ";
      for (int v : cycle)
         cout << v << " ";
      cout << endl;
   }
}
```

5.40 dominatorTree.cpp

dominatorTree.cpp 85 lines

```
/*
 Dominator Tree (Lengauer-Tarjan)
 Tested: SPOJ EN
 Complexity: O(m log n)
*/
struct graph
{
 int n;
 vector<vector<int>> adj, radj;
 graph(int n) : n(n), adj(n), radj(n) {}
 void add_edge(int src, int dst)
 {
  adj[src].push_back(dst);
  radj[dst].push_back(src);
 }
 vector<int> rank, semi, low, anc;
 int eval(int v)
 {
```

```cpp
  if (anc[v] < n && anc[anc[v]] < n)
  {
    int x = eval(anc[v]);
    if (rank[semi[low[v]]] > rank[semi[x]])
      low[v] = x;
    anc[v] = anc[anc[v]];
  }
  return low[v];
}
vector<int> prev, ord;
void dfs(int u)
{
  rank[u] = ord.size();
  ord.push_back(u);
  for (auto v : adj[u])
  {
    if (rank[v] < n)
      continue;
    dfs(v);
    prev[v] = u;
  }
}
vector<int> idom; // idom[u] is an immediate dominator of u
void dominator_tree(int r)
{
  idom.assign(n, n);
  prev = rank = anc = idom;
  semi.resize(n);
  iota(semi.begin(), semi.end(), 0);
  low = semi;
  ord.clear();
  dfs(r);
  vector<vector<int>> dom(n);
  for (int i = (int) ord.size() - 1; i >= 1; --i)
  {
    int w = ord[i];
    for (auto v : radj[w])
    {
      int u = eval(v);
      if (rank[semi[w]] > rank[semi[u]])
        semi[w] = semi[u];
    }
    dom[semi[w]].push_back(w);
    anc[w] = prev[w];
    for (int v : dom[prev[w]])
    {
      int u = eval(v);
      idom[v] = (rank[prev[w]] > rank[semi[u]]
        ? u : prev[w]);
    }
    dom[prev[w]].clear();
  }
  for (int i = 1; i < (int) ord.size(); ++i)
  {
    int w = ord[i];
    if (idom[w] != semi[w])
      idom[w] = idom[idom[w]];
  }
}
vector<int> dominators(int u)
{
  vector<int> S;
  for (; u < n; u = idom[u])
    S.push_back(u);
  return S;
}
};
```

5.41 flowWithLowerBound.cpp

Description: Solves max flow problem with lower bound for capacities

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

flowWithLowerBound.cpp c9b219, 117 lines

```cpp
template<typename T>
struct dinic
{
  struct edge
  {
    int src, dst;
    T low, cap, flow;
    int rev;
  };
  int n;
  vector<vector<edge>> adj;
  dinic(int n) : n(n), adj(n + 2) {}
  void add_edge(int src, int dst, T low, T cap)
  {
    adj[src].push_back({ src, dst, low, cap, 0, (int) adj[dst].size() });
    if (src == dst)
      adj[src].back().rev++;
    adj[dst].push_back({ dst, src, 0, 0, 0, (int) adj[src].size() - 1 });
  }
  vector<int> level, iter;
  T augment(int u, int t, T cur)
  {
    if (u == t)
      return cur;
    for (int &i = iter[u]; i < (int) adj[u].size(); ++i)
    {
      edge &e = adj[u][i];
      if (e.cap - e.flow > 0 && level[u] > level[e.dst])
      {
        T f = augment(e.dst, t, min(cur, e.cap - e.flow));
        if (f > 0)
        {
          e.flow += f;
          adj[e.dst][e.rev].flow -= f;
          return f;
        }
      }
    }
    return 0;
  }
  int bfs(int s, int t)
  {
    level.assign(n + 2, n + 2);
    level[t] = 0;
    queue<int> Q;
    for (Q.push(t); !Q.empty(); Q.pop())
    {
      int u = Q.front();
      if (u == s)
        break;
      for (edge &e : adj[u])
      {
        edge &erev = adj[e.dst][e.rev];
        if (erev.cap - erev.flow > 0
          && level[e.dst] > level[u] + 1)
        {
          Q.push(e.dst);
          level[e.dst] = level[u] + 1;
        }
      }
    }
    return level[s];
  }
  const T oo = numeric_limits<T>::max();
  T max_flow(int source, int sink)
  {
    vector<T> delta(n + 2);
    for (int u = 0; u < n; ++u) // initialize
      for (auto &e : adj[u])
      {
        delta[e.src] -= e.low;
        delta[e.dst] += e.low;
        e.cap -= e.low;
        e.flow = 0;
```

```
    }
  T sum = 0;
  int s = n, t = n + 1;
  for (int u = 0; u < n; ++u)
  {
    if (delta[u] > 0)
    {
      add_edge(s, u, 0, delta[u]);
      sum += delta[u];
    }
    else if (delta[u] < 0)
      add_edge(u, t, 0, -delta[u]);
  }
  add_edge(sink, source, 0, oo);
  T flow = 0;
  while (bfs(s, t) < n + 2)
  {
    iter.assign(n + 2, 0);
    for (T f; (f = augment(s, t, oo)) > 0;)
      flow += f;
  }
  if (flow != sum)
    return -1; // no solution
  for (int u = 0; u < n; ++u)
    for (auto &e : adj[u])
    {
      e.cap += e.low;
      e.flow += e.low;
      edge &erev = adj[e.dst][e.rev];
      erev.cap -= e.low;
      erev.flow -= e.low;
    }
  adj[sink].pop_back();
  adj[source].pop_back();
  while (bfs(source, sink) < n + 2)
  {
    iter.assign(n + 2, 0);
    for (T f; (f = augment(source, sink, oo)) > 0;)
      flow += f;
  } // level[u] == n + 2 ==> s-side
  return flow;
 }
};
```

5.42 gabowEdmonds.cpp

gabowEdmonds.cpp 90 lines

```
/*
 Tested: Timus 1099
 Complexity: O(n^3)
*/
struct graph
{
 int n;
 vector<vector<int>> adj;
 graph(int n) : n(n), adj(n) {}
 void add_edge(int u, int v)
 {
  adj[u].push_back(v);
  adj[v].push_back(u);
 }
 queue<int> q;
 vector<int> label, mate, cycle;
 void rematch(int x, int y)
 {
  int m = mate[x];
  mate[x] = y;
  if (mate[m] == x)
  {
   if (label[x] < n)
    rematch(mate[m] = label[x], m);
   else
   {
    int s = (label[x] - n) / n, t = (label[x] - n) % n;
    rematch(s, t);
    rematch(t, s);
   }
  }
 }
 void traverse(int x)
 {
  vector<int> save = mate;
  rematch(x, x);
  for (int u = 0; u < n; ++u)
   if (mate[u] != save[u])
    cycle[u] ^= 1;
  save.swap(mate);
 }
 void relabel(int x, int y)
 {
  cycle = vector<int>(n, 0);
  traverse(x);
  traverse(y);
  for (int u = 0; u < n; ++u)
  {
   if (!cycle[u] || label[u] >= 0)
    continue;
   label[u] = n + x + y * n;
    q.push(u);
  }
 }
 int augment(int r)
 {
  label.assign(n, -2);
  label[r] = -1;
  q = queue<int>();
  for (q.push(r); !q.empty(); q.pop())
  {
   int x = q.front();
   for (int y : adj[x])
   {
    if (mate[y] < 0 && r != y)
    {
     rematch(mate[y] = x, y);
     return 1;
    }
    else if (label[y] >= -1)
     relabel(x, y);
    else if (label[mate[y]] < -1)
    {
     label[mate[y]] = x;
     q.push(mate[y]);
    }
   }
  }
  return 0;
 }
 int maximum_matching()
 {
  mate.assign(n, -2);
  int matching = 0;
  for (int u = 0; u < n; ++u)
   if (mate[u] < 0)
    matching += augment(u);
  return matching;
 }
};
```

5.43 gomoryHuTree.cpp

gomoryHuTree.cpp 26 lines

```
/*
 Gomory-Hu tree
 Tested: SPOj MCQUERY
 Complexity: O(n-1) max-flow call
*/
template<typename flow_type>
```

```cpp
struct edge
{
 int src, dst;
 flow_type cap;
};
template<typename flow_type>
vector<edge<flow_type>> gomory_hu(dinic<flow_type> &adj)
{
 int n = adj.n;
 vector<edge<flow_type>> tree;
 vector<int> parent(n);
 for (int u = 1; u < n; ++u)
 {
  tree.push_back({ u, parent[u], adj.max_flow(u, parent[u]) });
  for (int v = u + 1; v < n; ++v)
   if (adj.level[v] == -1 && parent[v] == parent[u])
    parent[v] = u;
 }
 return tree;
}
```

5.44 hopcroftKarp.h

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

hopcroftKarp.h b66c8a, 91 lines

```cpp
struct Bipartite_Matching{
   vector<vector<int>> es;
   vector<int> d, match;
   vector<bool> used, used2;
   const int n, m;
   Bipartite_Matching(int n, int m) : es(n), d(n), match(m), used(n),
        used2(n), n(n), m(m) {}
   void add_edge(int u, int v){
      es[u].push_back(v);
   }
   void bfs(){
      fill(begin(d), end(d), -1);
      queue<int> que;
      for(int i = 0; i < n; i++){
         if(!used[i]) que.emplace(i), d[i] = 0;
      }
      while(!que.empty()){
         int i = que.front(); que.pop();
         for(auto &e: es[i]){
            int j = match[e];
            if(j != -1 && d[j] == -1){
               que.emplace(j), d[j] = d[i]+1;
            }
         }
      }
   }
   bool dfs(int now){
      used2[now] = true;
      for(auto &e: es[now]){
         int u = match[e];
         if(u == -1 || (!used2[u] && d[u] == d[now]+1 && dfs(u))){
            match[e] = now, used[now] = true;
            return true;
         }
      }
      return false;
   }
   int bipartite_matching(){
      fill(begin(match), end(match), -1), fill(begin(used), end(used),
           false);
      int ret = 0;
      while(true){
         bfs();
         fill(begin(used2), end(used2), false);
         int flow = 0;
         for(int i = 0; i < n; i++){
            if(!used[i] && dfs(i)) flow++;
         }
         if(flow == 0) break;
         ret += flow;
      }
      return ret;
   }
};
// If you graph is not dividen in proper way (not divided in two sets L
    and R) call this function
int ConverLR(int n){
   vector<bool> vis(n);
   vector<int> color(n);
   auto bfsColor = [&](int s){
      vector<int> q;
      q.push_back(s);
      while(q.size()){
         int u = q.back();
         vis[u] = true;
         q.pop_back();
         for(auto v:graph[u]){
            if(!vis[v]){
               q.push_back(v);
               color[v] = color[u]^1;
            }
         }
      }
   };
   for(int i = 0;i<n;i++)
      if(!vis[i])
         bfsColor(i);
   map<int,int> mpR;
   int m= 0,key = 0;
   vector<vector<int>> g;
   for(int i = 0;i<n;i++){
      if(color[i]){
         g.push_back(vector<int>());
         for(auto d:graph[i]){
            if(!mpR.count(d))
               mpR[d] = key++;
            g.back().push_back(mpR[d]);
         }
      }
      else m++;
   }
   Bipartite_matching BM(n-m,m,g);
   return BM.bipartite_matching();
}
```

5.45 kShortestPaths.cpp

kShortestPaths.cpp 74 lines

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

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

5.46 kuhn-munkras.cpp

kuhn-munkras.cpp — 59 lines

```
template<typename T>
struct KuhnMunkras { // n for left, m for right
  int n, m, match[maxM];
  T g[maxN][maxM], lx[maxN], ly[maxM], slack[maxM];
  bool vx[maxN], vy[maxM];
  void init(int n_, int m_) {
    mset0(g); n = n_, m = m_;
  }
  void add(int u, int v, T w) {
    g[u][v] = w;
  }
  bool find(int x) {
    vx[x] = true;
    for (int y = 1; y <= m; ++y) {
      if (!vy[y]) {
        T delta = lx[x] + ly[y] - g[x][y];
        if (equalT(delta, T(0))) {
          vy[y] = true;
          if (match[y] == 0 || find(match[y])) {
            match[y] = x;
            return true;
          }
        } else slack[y] = min(slack[y], delta);
      }
    }
    return false;
  }
  T matching() { // maximum weight matching
    fill(lx + 1, lx + 1 + n, numeric_limits<T>::lowest());
    mset0(ly); mset0(match);
    for (int i = 1; i <= n; ++i) {
      for (int j = 1; j <= m; ++j) lx[i] = max(lx[i], g[i][j]);
    }
    for (int k = 1; k <= n; ++k) {
      fill(slack + 1, slack + 1 + m, numeric_limits<T>::max());
      while (true) {
        mset0(vx); mset0(vy);
        if (find(k)) break;
        else {
          T delta = numeric_limits<T>::max();
          for (int i = 1; i <= m; ++i) {
            if (!vy[i]) delta = min(delta, slack[i]);
          }
          for (int i = 1; i <= n; ++i) {
            if (vx[i]) lx[i] -= delta;
          }
          for (int i = 1; i <= m; ++i) {
            if (vy[i]) ly[i] += delta;
            if (!vy[i]) slack[i] -= delta;
          }
        }
      }
    }
    T result = 0;
    for (int i = 1; i <= n; ++i) result += lx[i];
    for (int i = 1; i <= m; ++i) result += ly[i];
    return result;
  }
};
```

5.47 maxFlowDinic.cpp

Description: Solves max flow problem

Time: Approximately $\mathcal{O}\left(v^2E\right)$ but faster in practice , also for bipartite matching complexity is :$\mathcal{O}\left(E\sqrt{V}\right)$

maxFlowDinic.cpp — c3e9a0, 68 lines

```
template<typename flow_type>
struct dinic
{
  struct edge
  {
    size_t src, dst, rev;
    flow_type flow, cap;
  };
  int n;
  vector<vector<edge>> adj;
  dinic(int n) : n(n), adj(n), level(n), q(n), it(n) {}
  void add_edge(size_t src, size_t dst, flow_type cap, flow_type rcap = 0)
  {
    adj[src].push_back({src, dst, adj[dst].size(), 0, cap});
    if (src == dst) adj[src].back().rev++;
    adj[dst].push_back({dst, src, adj[src].size() - 1, 0, rcap});
  }
  vector<int> level, q, it;
  bool bfs(int source, int sink)
```

```
  {
    fill(level.begin(), level.end(), -1);
    for (int qf = level[q[0] = sink] = 0, qb = 1; qf < qb; ++qf)
    {
      sink = q[qf];
      for (edge &e : adj[sink])
      {
        edge &r = adj[e.dst][e.rev];
        if (r.flow < r.cap && level[e.dst] == -1)
          level[q[qb++] = e.dst] = 1 + level[sink];
      }
    }
    return level[source] != -1;
  }
  flow_type augment(int source, int sink, flow_type flow)
  {
    if (source == sink) return flow;
    for (; it[source] != adj[source].size(); ++it[source])
    {
      edge &e = adj[source][it[source]];
      if (e.flow < e.cap && level[e.dst] + 1 == level[source])
      {
        flow_type delta = augment(e.dst, sink,
              min(flow, e.cap - e.flow));
        if (delta > 0)
        {
          e.flow += delta;
          adj[e.dst][e.rev].flow -= delta;
          return delta;
        }
      }
    }
    return 0;
  }
  flow_type max_flow(int source, int sink)
  {
    for (int u = 0; u < n; ++u)
      for (edge &e : adj[u]) e.flow = 0;
    flow_type flow = 0;
    flow_type oo = numeric_limits<flow_type>::max();
    while (bfs(source, sink))
    {
      fill(it.begin(), it.end(), 0);
      for (flow_type f; (f = augment(source, sink, oo)) > 0;)
        flow += f;
    } // level[u] = -1 => source side of min cut
    return flow;
  }
};
```

5.48 maxFlowPushRelabel.cpp

maxFlowPushRelabel.cpp 93 lines

```
/*
  Maximum Flow (Goldberg-Tarjan)
  Complexity: O(n^3) faster than Dinic in most cases
  Tested: http://www.spoj.com/problems/FASTFLOW/
*/
template<typename flow_type>
struct goldberg_tarjan
{
  struct edge
  {
    size_t src, dst, rev;
    flow_type flow, cap;
  };
  int n;
  vector<vector<edge>> adj;
  goldberg_tarjan(int n) : n(n), adj(n) {}
  void add_edge(size_t src, size_t dst, flow_type cap, flow_type rcap =
      0)
  {
    adj[src].push_back({ src, dst, adj[dst].size(), 0, cap });
    if (src == dst) adj[src].back().rev++;
    adj[dst].push_back({ dst, src, adj[src].size() - 1, 0, rcap });
  }
  flow_type max_flow(int source, int sink)
  {
    vector<flow_type> excess(n);
    vector<int> dist(n), active(n), count(2 * n);
    queue<int> q;
    auto enqueue = [&](int v)
    {
      if (!active[v] && excess[v] > 0)
      {
        active[v] = true;
        q.push(v);
      }
    };
    auto push = [&](edge &e)
    {
      flow_type f = min(excess[e.src], e.cap - e.flow);
      if (dist[e.src] <= dist[e.dst] || f == 0) return;
      e.flow += f;
      adj[e.dst][e.rev].flow -= f;
      excess[e.dst] += f;
      excess[e.src] -= f;
      enqueue(e.dst);
    };
    dist[source] = n;
    active[source] = active[sink] = true;
    count[0] = n - 1;
    count[n] = 1;
    for (int u = 0; u < n; ++u)
      for (edge &e : adj[u]) e.flow = 0;
    for (edge &e : adj[source])
    {
      excess[source] += e.cap;
      push(e);
    }
    for (int u; !q.empty(); q.pop())
    {
      active[u = q.front()] = false;
      for (auto &e : adj[u]) push(e);
      if (excess[u] > 0)
      {
        if (count[dist[u]] == 1)
        {
          int k = dist[u]; // Gap Heuristics
          for (int v = 0; v < n; v++)
          {
            if (dist[v] < k)
              continue;
            count[dist[v]]--;
            dist[v] = max(dist[v], n + 1);
            count[dist[v]]++;
            enqueue(v);
          }
        }
        else
        {
          count[dist[u]]--; // Relabel
          dist[u] = 2 * n;
          for (edge &e : adj[u])
            if (e.cap > e.flow)
              dist[u] = min(dist[u], dist[e.dst] + 1);
          count[dist[u]]++;
          enqueue(u);
        }
      }
    }
    flow_type flow = 0;
    for (edge e : adj[source])
      flow += e.flow;
    return flow;
  }
};
```

5.49 minCostMaxFlow.cpp

minCostMaxFlow.cpp 87 lines

```cpp
/*
  Minimum Cost Flow (Tomizawa, Edmonds-Karp)
  Complexity: O(F m log n), where F is the amount of maximum flow
  Tested: Codeforces [http://codeforces.com/problemset/problem/717/G]
*/
template<typename flow_type, typename cost_type>
struct min_cost_max_flow
{
  struct edge
  {
    size_t src, dst, rev;
    flow_type flow, cap;
    cost_type cost;
  };
  int n;
  vector<vector<edge>> adj;
  min_cost_max_flow(int n) : n(n), adj(n), potential(n), dist(n), back(n)
      {}
  void add_edge(size_t src, size_t dst, flow_type cap, cost_type cost)
  {
    adj[src].push_back({src, dst, adj[dst].size(), 0, cap, cost});
    if (src == dst)
      adj[src].back().rev++;
    adj[dst].push_back({dst, src, adj[src].size() - 1, 0, 0, -cost});
  }
  vector<cost_type> potential;
  inline cost_type rcost(const edge &e)
  {
    return e.cost + potential[e.src] - potential[e.dst];
  }
  void bellman_ford(int source)
  {
    for (int k = 0; k < n; ++k)
      for (int u = 0; u < n; ++u)
        for (edge &e : adj[u])
          if (e.cap > 0 && rcost(e) < 0)
            potential[e.dst] += rcost(e);
  }
  const cost_type oo = numeric_limits<cost_type>::max();
  vector<cost_type> dist;
  vector<edge*> back;
  cost_type dijkstra(int source, int sink)
  {
    fill(dist.begin(), dist.end(), oo);
    typedef pair<cost_type, int> node;
    priority_queue<node, vector<node>, greater<node>> pq;
    for (pq.push({dist[source] = 0, source}); !pq.empty();)
    {
      node p = pq.top(); pq.pop();
      if (dist[p.second] < p.first) continue;
      if (p.second == sink) break;
      for (edge &e : adj[p.second])
        if (e.flow < e.cap &&
          dist[e.dst] > dist[e.src] + rcost(e))
        {
          back[e.dst] = &e;
          pq.push({dist[e.dst] = dist[e.src] + rcost(e),
              e.dst});
        }
    }
    return dist[sink];
  }
  pair<flow_type, cost_type> max_flow(int source, int sink)
  {
    flow_type flow = 0;
    cost_type cost = 0;
    for (int u = 0; u < n; ++u)
      for (edge &e : adj[u]) e.flow = 0;
    potential.assign(n, 0);
    dist.assign(n, 0);
    back.assign(n, nullptr);
    bellman_ford(source); // remove negative costs
    while (dijkstra(source, sink) < oo)
    {
      for (int u = 0; u < n; ++u)
        if (dist[u] < dist[sink])
          potential[u] += dist[u] - dist[sink];
      flow_type f = numeric_limits<flow_type>::max();
      for (edge *e = back[sink]; e; e = back[e->src])
        f = min(f, e->cap - e->flow);
      for (edge *e = back[sink]; e; e = back[e->src])
        e->flow += f, adj[e->dst][e->rev].flow -= f;
      flow += f;
      cost += f * (potential[sink] - potential[source]);
    }
    return {flow, cost};
  }
};
```

5.50 stoerWagner.cpp

stoerWagner.cpp 46 lines

```cpp
/*
  Tested: ZOJ 2753
  Complexity: O(n^3)
*/
template<typename T>
pair<T, vector<int>> stoer_wagner(vector<vector<T>> &weights)
{
  int n = weights.size();
  vector<int> used(n), cut, best_cut;
  T best_weight = -1;
  for (int phase = n - 1; phase >= 0; --phase)
  {
    vector<T> w = weights[0];
    vector<int> added = used;
    int prev, last = 0;
    for (int i = 0; i < phase; ++i)
    {
      prev = last;
      last = -1;
      for (int j = 1; j < n; ++j)
        if (!added[j] && (last == -1 || w[j] > w[last]))
          last = j;
      if (i == phase - 1)
      {
        for (int j = 0; j < n; ++j)
          weights[prev][j] += weights[last][j];
        for (int j = 0; j < n; ++j)
          weights[j][prev] = weights[prev][j];
        used[last] = true;
        cut.push_back(last);
        if (best_weight == -1 || w[last] < best_weight)
        {
          best_cut = cut;
          best_weight = w[last];
        }
      }
      else
      {
        for (int j = 0; j < n; j++)
          w[j] += weights[last][j];
        added[last] = true;
      }
    }
  }
  return make_pair(best_weight, best_cut);
}
```

5.51 topTree.cpp

Description: The ultimate link cut tree supporting all path/subtree lazy updates and aggregates. Also known as AAA tree. Nodes are 1-indexed. Note that lazy style is different from my usual style, lazy on node means value is already updated instead of pending update. Only modifications required are: 1. Data struct, 2. Set MAX to 2n, 3. Populate freeList with n nodes at the start.

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

topTree.cpp 1259b1, 279 lines

```cpp
const int MAX = 2e5 + 5;
struct Data {
	int sum, mn, mx, sz;
	Data() : sum(0), mn(INT_MAX), mx(INT_MIN), sz(0) {}
	Data(int val) : sum(val), mn(val), mx(val), sz(1) {}
	Data(int _sum, int _mn, int _mx, int _sz) : sum(_sum), mn(_mn), mx(_mx), sz(_sz) {}
};
struct Lazy {
	int a, b;
	Lazy(int _a = 1, int _b = 0) : a(_a), b(_b) {}
	bool lazy() {
		return a != 1 || b != 0;
	}
};
Data operator + (const Data &a, const Data &b) {
	return Data(a.sum + b.sum, min(a.mn, b.mn), max(a.mx, b.mx), a.sz + b.sz);
}
Data& operator += (Data &a, const Data &b) {
	return a = a + b;
}
Lazy& operator += (Lazy &a, const Lazy &b) {
	return a = Lazy(a.a * b.a, a.b * b.a + b.b);
}
int& operator += (int &a, const Lazy &b) {
	return a = a * b.a + b.b;
}
Data& operator += (Data &a, const Lazy &b) {
	return a.sz ? a = Data(a.sum * b.a + a.sz * b.b, a.mn * b.a + b.b, a.mx * b.a + b.b, a.sz) : a;
}
struct Node {
	int p, ch[4], val;
	Data path, sub, all;
	Lazy plazy, slazy;
	bool flip, fake;
	Node() : p(0), ch(), path(), sub(), all(), plazy(), slazy(), flip(false), fake(true) {}
	Node(int _val) : Node() {
		val = _val;
		path = all = Data(val);
		fake = false;
	}
} tr[MAX];
vector<int> freeList;
void pushFlip(int u) {
	if (!u)
		return;
	swap(tr[u].ch[0], tr[u].ch[1]);
	tr[u].flip ^= true;
}
void pushPath(int u, const Lazy &lazy) {
	if (!u || tr[u].fake)
		return;
	tr[u].val += lazy;
	tr[u].path += lazy;
	tr[u].all = tr[u].path + tr[u].sub;
	tr[u].plazy += lazy;
}
void pushSub(int u, bool o, const Lazy &lazy) {
	if (!u)
		return;
	tr[u].sub += lazy;
	tr[u].slazy += lazy;
	if (!tr[u].fake && o)
		pushPath(u, lazy);
	else
		tr[u].all = tr[u].path + tr[u].sub;
}
void push(int u) {
	if (!u)
		return;
	if (tr[u].flip) {
		pushFlip(tr[u].ch[0]);
		pushFlip(tr[u].ch[1]);
		tr[u].flip = false;
	}
	if (tr[u].plazy.lazy()) {
		pushPath(tr[u].ch[0], tr[u].plazy);
		pushPath(tr[u].ch[1], tr[u].plazy);
		tr[u].plazy = Lazy();
	}
	if (tr[u].slazy.lazy()) {
		pushSub(tr[u].ch[0], false, tr[u].slazy);
		pushSub(tr[u].ch[1], false, tr[u].slazy);
		pushSub(tr[u].ch[2], true, tr[u].slazy);
		pushSub(tr[u].ch[3], true, tr[u].slazy);
		tr[u].slazy = Lazy();
	}
}
void pull(int u) {
	if (!tr[u].fake)
		tr[u].path = tr[tr[u].ch[0]].path + tr[tr[u].ch[1]].path + tr[u].val;
	tr[u].sub = tr[tr[u].ch[0]].sub + tr[tr[u].ch[1]].sub + tr[tr[u].ch[2]].all + tr[tr[u].ch[3]].all;
	tr[u].all = tr[u].path + tr[u].sub;
}
void attach(int u, int d, int v) {
	tr[u].ch[d] = v;
	tr[v].p = u;
	pull(u);
}
int dir(int u, int o) {
	int v = tr[u].p;
	return tr[v].ch[o] == u ? o : tr[v].ch[o+1] == u ? o + 1 : -1;
}
void rotate(int u, int o) {
	int v = tr[u].p, w = tr[v].p, du = dir(u, o), dv = dir(v, o);
	if (dv == -1 && o == 0)
		dv = dir(v, 2);
	attach(v, du, tr[u].ch[du^1]);
	attach(u, du ^1, v);
	if (~dv)
		attach(w, dv, u);
	else
		tr[u].p = w;
}
void splay(int u, int o) {
	push(u);
	while (~dir(u, o) && (o == 0 || tr[tr[u].p].fake)) {
		int v = tr[u].p, w = tr[v].p;
		push(w);
		push(v);
		push(u);
		int du = dir(u, o), dv = dir(v, o);
		if (~dv && (o == 0 || tr[w].fake))
			rotate(du == dv ? v : u, o);
		rotate(u, o);
	}
}
void add(int u, int v) {
	if (!v)
		return;
	for (int i=2; i<4; i++)
		if (!tr[u].ch[i]) {
			attach(u, i, v);
			return;
		}
	int w = freeList.back();
	freeList.pop_back();
	attach(w, 2, tr[u].ch[2]);
	attach(w, 3, v);
```

```
    attach(u, 2, w);
}
void recPush(int u) {
    if (tr[u].fake)
        recPush(tr[u].p);
    push(u);
}
void rem(int u) {
    int v = tr[u].p;
    recPush(v);
    if (tr[v].fake) {
        int w = tr[v].p;
        attach(w, dir(v, 2), tr[v].ch[dir(u, 2) ^1]);
        if (tr[w].fake)
            splay(w, 2);
        freeList.push_back(v);
    } else {
        attach(v, dir(u, 2), 0);
    }
    tr[u].p = 0;
}
int par(int u) {
    int v = tr[u].p;
    if (!tr[v].fake)
        return v;
    splay(v, 2);
    return tr[v].p;
}
int access(int u) {
    int v = u;
    splay(u, 0);
    add(u, tr[u].ch[1]);
    attach(u, 1, 0);
    while (tr[u].p) {
        v = par(u);
        splay(v, 0);
        rem(u);
        add(v, tr[v].ch[1]);
        attach(v, 1, u);
        splay(u, 0);
    }
    return v;
}
void reroot(int u) {
    access(u);
    pushFlip(u);
}
void link(int u, int v) {
    reroot(u);
    access(v);
    add(v, u);
}
void cut(int u, int v) {
    reroot(u);
    access(v);
    tr[v].ch[0] = tr[u].p = 0;
    pull(v);
}
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int n, m;
    cin >> n >> m;
    vector<pair<int, int>> edges;
    for (int i=0; i<n-1; i++) {
        int x, y;
        cin >> x >> y;
        edges.emplace_back(x, y);
    }
    for (int i=1; i<=n; i++) {
        int w;
        cin >> w;
        tr[i] = Node(w);
    }
    for (int i=n+1; i<MAX; i++)
        freeList.push_back(i);
    for (auto [x, y] : edges)
        link(x, y);
    int rt;
    cin >> rt;
    while (m--) {
        int k;
        cin >> k;
        if (k == 1) { // change root
            cin >> rt;
        } else if (k == 3 || k == 4 || k == 11) { // subtree query
            int x;
            cin >> x;
            reroot(rt);
            access(x);
            Data ret = tr[x].val;
            for (int i=2; i<4; i++)
                ret += tr[tr[x].ch[i]].all;
            if (k == 3)
                cout << ret.mn << "\n";
            else if (k == 4)
                cout << ret.mx << "\n";
            else
                cout << ret.sum << "\n";
        } else if (k == 0 || k == 5) { // subtree update
            int x, y;
            cin >> x >> y;
            reroot(rt);
            access(x);
            Lazy lazy(k == 5, y);
            tr[x].val += lazy;
            for (int i=2; i<4; i++)
                pushSub(tr[x].ch[i], true, lazy);
        } else if (k == 7 || k == 8 || k == 10) { // path query
            int x, y;
            cin >> x >> y;
            reroot(x);
            access(y);
            Data ret = tr[y].path;
            if (k == 7)
                cout << ret.mn << "\n";
            else if (k == 8)
                cout << ret.mx << "\n";
            else
                cout << ret.sum << "\n";
        } else if (k == 2 || k == 6) { // path update
            int x, y, z;
            cin >> x >> y >> z;
            reroot(x);
            access(y);
            pushPath(y, Lazy(k == 6, z));
        } else { // change parent
            int x, y;
            cin >> x >> y;
            reroot(rt);
            access(y);
            if (access(x) != x) {
                tr[x].ch[0] = tr[tr[x].ch[0]].p = 0;
                pull(x);
                access(y);
                add(y, x);
            }
        }
    }
    return 0;
}
```

5.52 treeIsomorphism.cpp

treeIsomorphism.cpp 115 lines

```
/*
 Tested: SPOJ TREEISO
```

```
  Complexity: O(n log n)
*/
#define all(c) (c).begin(), (c).end()
struct tree
{
  int n;
  vector<vector<int>> adj;
  tree(int n) : n(n), adj(n) {}
  void add_edge(int src, int dst)
  {
    adj[src].push_back(dst);
    adj[dst].push_back(src);
  }
  vector<int> centers()
  {
    vector<int> prev;
    int u = 0;
    for (int k = 0; k < 2; ++k)
    {
      queue<int> q;
      prev.assign(n, -1);
      for (q.push(prev[u] = u); !q.empty(); q.pop())
      {
        u = q.front();
        for (auto v : adj[u])
        {
          if (prev[v] >= 0)
            continue;
          q.push(v);
          prev[v] = u;
        }
      }
    }
    vector<int> path = { u };
    while (u != prev[u])
      path.push_back(u = prev[u]);
    int m = path.size();
    if (m % 2 == 0)
      return {path[m/2-1], path[m/2]};
    else
      return {path[m/2]};
  }
  vector<vector<int>> layer;
  vector<int> prev;
  int levelize(int r)
  {
    prev.assign(n, -1);
    prev[r] = n;
    layer = {{r}};
    while (1)
    {
      vector<int> next;
      for (int u : layer.back())
        for (int v : adj[u])
        {
          if (prev[v] >= 0)
            continue;
          prev[v] = u;
          next.push_back(v);
        }
      if (next.empty())
        break;
      layer.push_back(next);
    }
    return layer.size();
  }
};
bool isomorphic(tree S, int s, tree T, int t)
{
  if (S.n != T.n)
    return false;
  if (S.levelize(s) != T.levelize(t))
    return false;
  vector<vector<int>> longcodeS(S.n + 1), longcodeT(T.n + 1);
  vector<int> codeS(S.n), codeT(T.n);
  for (int h = (int) S.layer.size() - 1; h >= 0; --h)
  {
    map<vector<int>, int> bucket;
    for (int u : S.layer[h])
    {
      sort(all(longcodeS[u]));
      bucket[longcodeS[u]] = 0;
    }
    for (int u : T.layer[h])
    {
      sort(all(longcodeT[u]));
      bucket[longcodeT[u]] = 0;
    }
    int id = 0;
    for (auto &p : bucket)
      p.second = id++;
    for (int u : S.layer[h])
    {
      codeS[u] = bucket[longcodeS[u]];
      longcodeS[S.prev[u]].push_back(codeS[u]);
    }
    for (int u : T.layer[h])
    {
      codeT[u] = bucket[longcodeT[u]];
      longcodeT[T.prev[u]].push_back(codeT[u]);
    }
  }
  return codeS[s] == codeT[t];
}
bool isomorphic(tree S, tree T)
{
  auto x = S.centers(), y = T.centers();
  if (x.size() != y.size())
    return false;
  if (isomorphic(S, x[0], T, y[0]))
    return true;
  return x.size() > 1 && isomorphic(S, x[1], T, y[0]);
}
```

geometry (6)

6.1 2d-base.cpp

2d-base.cpp 117 lines

```
struct point_t {
  double x, y;
  point_t() { }
  point_t(double tx, double ty) : x(tx), y(ty) { }
  point_t operator-(const point_t &r) const { return point_t(x - r.x, y -
      r.y); }
  point_t operator+(const point_t &r) const { return point_t(x + r.x, y +
      r.y); }
  point_t operator*(double r) const { return point_t(x * r, y * r); }
  point_t operator/(double r) const { return point_t(x / r, y / r); }
  point_t rot90() const { return point_t(-y, x); }
  double l() const { return sqrt(x * x + y * y); }
  void read() { scanf("%lf%lf", &x, &y); }
};
int dblcmp(double x) {
  return (x < -eps ? -1 : x > eps);
}
double dist(point_t p1, point_t p2) {
  return (p2 - p1).l();
}
double cross(point_t p1, point_t p2) {
  return p1.x * p2.y - p2.x * p1.y;
}
double dot(point_t p1, point_t p2) {
  return p1.x * p2.x + p1.y * p2.y;
}
// count-clock wise is positive direction
```

```
double angle(point_t p1, point_t p2) {
  double x1 = p1.x, y1 = p1.y, x2 = p2.x, y2 = p2.y;
  double a1 = atan2(y1, x1), a2 = atan2(y2, x2);
  double a = a2 - a1;
  while (a < -pi) a += 2 * pi;
  while (a ≥ pi) a -= 2 * pi;
  return a;
}
bool onSeg(point_t p, point_t a, point_t b) {
  return dblcmp(cross(a - p, b - p)) == 0 && dblcmp(dot(a - p, b - p)) ≤
      ↪ 0;
}
// 1 normal intersected, -1 denormal intersected, 0 not intersected
int testSS(point_t a, point_t b, point_t c, point_t d) {
  if (dblcmp(max(a.x, b.x) - min(c.x, d.x)) < 0) return 0;
  if (dblcmp(max(c.x, d.x) - min(a.x, b.x)) < 0) return 0;
  if (dblcmp(max(a.y, b.y) - min(c.y, d.y)) < 0) return 0;
  if (dblcmp(max(c.y, d.y) - min(a.y, b.y)) < 0) return 0;
  int d1 = dblcmp(cross(c - a, b - a));
  int d2 = dblcmp(cross(d - a, b - a));
  int d3 = dblcmp(cross(a - c, d - c));
  int d4 = dblcmp(cross(b - c, d - c));
  if ((d1 * d2 < 0) && (d3 * d4 < 0)) return 1;
  if ((d1 * d2 ≤ 0 && d3 * d4 == 0) || (d1 * d2 == 0 && d3 * d4 ≤ 0))
      ↪ return -1;
  return 0;
}
vector<point_t> isLL(point_t a, point_t b, point_t c, point_t d) {
  point_t p1 = b - a, p2 = d - c;
  vector<point_t> ret;
  double a1 = p1.y, b1 = -p1.x, c1;
  double a2 = p2.y, b2 = -p2.x, c2;
  if (dblcmp(a1 * b2 - a2 * b1) == 0) return ret; // colined ≤> a1*c2-a2
      ↪ *c1=0 && b1*c2-b2*c1=0
  else {
    c1 = a1 * a.x + b1 * a.y;
    c2 = a2 * c.x + b2 * c.y;
    ret.push_back(point_t((c1 * b2 - c2 * b1) / (a1 * b2 - a2 * b1), (c1
      ↪ * a2 - c2 * a1) / (b1 * a2 - b2 * a1)));
    return ret;
  }
}
point_t angle_bisector(point_t p0, point_t p1, point_t p2) {
  point_t v1 = p1 - p0, v2 = p2 - p0;
  v1 = v1 / dist(v1) * dist(v2);
  return v1 + v2 + p0;
}
point_t perpendicular_bisector(point_t p1, point_t p2) {
  point_t v = p2 - p1;
  swap(v.x, v.y);
  v.x = -v.x;
  return v + (p1 + p2) / 2;
}
point_t circumcenter(point_t p0, point_t p1, point_t p2) {
  point_t v1 = perpendicular_bisector(p0, p1);
  point_t v2 = perpendicular_bisector(p1, p2);
  return isLL((p0 + p1) / 2, v1, (p1 + p2) / 2, v2);
}
point_t incenter(point_t p0, point_t p1, point_t p2) {
  point_t v1 = angle_bisector(p0, p1, p2);
  point_t v2 = angle_bisector(p1, p2, p0);
  return isLL(p0, v1, p1, v2);
}
point_t orthocenter(point_t p0, point_t p1, point_t p2) {
  return p0 + p1 + p2 - circumcenter(p0, p1, p2) * 2;
}
// count-clock wise is positive direction
point_t rotate(point_t p, double a) {
  double s = sin(a), c = cos(a);
  return point_t(p.x * c - p.y * s, p.y * c + p.x * s);
}
bool insidePoly(point_t *p, int n, point_t t) {
  p[0] = p[n];
  for (int i = 0; i < n; ++i) if (onSeg(t, p[i], p[i + 1])) return true;
  point_t r = point_t(2353456.663, 5326546.243); // random point
  int cnt = 0;
  for (int i = 0; i < n; ++i) {
    if (testSS(t, r, p[i], p[i + 1]) ≠ 0) ++cnt;
  }
  return cnt & 1;
}
bool insideConvex(point_t *convex, int n, point_t t) { // O(logN),
    ↪ convex polygen, cross(p[2] - p[1], p[3] - p[1]) > 0
  if (n == 2) return onSeg(t, convex[1], convex[2]);
  int l = 2, r = n;
  while (l < r) {
    int mid = (l + r) / 2 + 1;
    int side = dblcmp(cross(convex[mid] - convex[1], t - convex[1]));
    if (side == 1) l = mid;
    else r = mid - 1;
  }
  int s = dblcmp(cross(convex[l] - convex[1], t - convex[1]));
  if (s == -1 || l == n) return false;
  point_t v = convex[l + 1] - convex[l];
  if (dblcmp(cross(v, t - convex[l])) ≥ 0) return true;
  return false;
}
```

6.2 3dHull.h

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

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

3dHull.h c114bf, 46 lines

```
#include "Point3D.h"
typedef Point3D<double> P3;
struct PR {
  void ins(int x) { (a == -1 ? a : b) = x; }
  void rem(int x) { (a == x ? a : b) = -1; }
  int cnt() { return (a ≠ -1) + (b ≠ -1); }
  int a, b;
};
struct F { P3 q; int a, b, c; };
vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) ≥ 4);
  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);
  rep(i,4,sz(A)) {
    rep(j,0,sz(FS)) {
      F f = FS[j];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();
      }
    }
    int nw = sz(FS);
    rep(j,0,nw) {
      F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() ≠ 2) mf(f.a, f.b, i, f.c);
      C(a, b, c); C(a, c, b); C(b, c, a);
    }
  }
  for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
    A[it.c] - A[it.a]).dot(it.q) ≤ 0) swap(it.c, it.b);
```

```
  return FS;
};
```

6.3 AllGeometry.cpp

AllGeometry.cpp 820 lines

```
#include <bits/stdc++.h>
using namespace std;
using ld = long double;
const ld eps = 1e-9, inf = numeric_limits<ld>::max(), pi = acos(-1);
// For use with integers, just set eps=0 and everything remains the same
bool geq(ld a, ld b){return a-b >= -eps;} //a >= b
bool leq(ld a, ld b){return b-a >= -eps;} //a <= b
bool ge(ld a, ld b){return a-b > eps;} //a > b
bool le(ld a, ld b){return b-a > eps;} //a < b
bool eq(ld a, ld b){return abs(a-b) <= eps;} //a == b
bool neq(ld a, ld b){return abs(a-b) > eps;} //a != b
struct point{
  ld x, y;
  point(): x(0), y(0){}
  point(ld x, ld y): x(x), y(y){}
  point operator+(const point & p) const{return point(x + p.x, y + p.y);}
  point operator-(const point & p) const{return point(x - p.x, y - p.y);}
  point operator*(const ld & k) const{return point(x * k, y * k);}
  point operator/(const ld & k) const{return point(x / k, y / k);}
  point operator+=(const point & p){*this = *this + p; return *this;}
  point operator-=(const point & p){*this = *this - p; return *this;}
  point operator*=(const ld & p){*this = *this * p; return *this;}
  point operator/=(const ld & p){*this = *this / p; return *this;}
  point rotate(const ld & a) const{return point(x*cos(a) - y*sin(a), x*sin(a) + y*cos(a));}
  point perp() const{return point(-y, x);}
  ld ang() const{
    ld a = atan2l(y, x); a += le(a, 0) ? 2*pi : 0; return a;
  }
  ld dot(const point & p) const{return x * p.x + y * p.y;}
  ld cross(const point & p) const{return x * p.y - y * p.x;}
  ld norm() const{return x * x + y * y;}
  ld length() const{return sqrtl(x * x + y * y);}
  point unit() const{return (*this) / length();}
  bool operator==(const point & p) const{return eq(x, p.x) && eq(y, p.y);}
  bool operator!=(const point & p) const{return !(*this == p);}
  bool operator<(const point & p) const{return le(x, p.x) || (eq(x, p.x) && le(y, p.y));}
  bool operator>(const point & p) const{return ge(x, p.x) || (eq(x, p.x) && ge(y, p.y));}
  bool half(const point & p) const{return le(p.cross(*this), 0) || (eq(p.cross(*this), 0) && le(p.dot(*this), 0));}
};
istream &operator>>(istream &is, point & p){return is >> p.x >> p.y;}
ostream &operator<<(ostream &os, const point & p){return os << "(" << p.x << ", " << p.y << ")";}
int sgn(ld x){
  if(ge(x, 0)) return 1;
  if(le(x, 0)) return -1;
  return 0;
}
void polarSort(vector<point> & P, const point & o, const point & v){
  //sort points in P around o, taking the direction of v as first angle
  sort(P.begin(), P.end(), [&](const point & a, const point & b){
    return point((a - o).half(v), 0) < point((b - o).half(v), (a - o).cross(b - o));
  });
}
bool pointInLine(const point & a, const point & v, const point & p){
  //line a+tv, point p
  return eq((p - a).cross(v), 0);
}
bool pointInSegment(const point & a, const point & b, const point & p){
  //segment ab, point p
  return pointInLine(a, b - a, p) && leq((a - p).dot(b - p), 0);
}
int intersectLinesInfo(const point & a1, const point & v1, const point & a2, const point & v2){
  //lines a1+tv1 and a2+tv2
  ld det = v1.cross(v2);
  if(eq(det, 0)){
    if(eq((a2 - a1).cross(v1), 0)){
      return -1; //infinity points
    }else{
      return 0; //no points
    }
  }else{
    return 1; //single point
  }
}
point intersectLines(const point & a1, const point & v1, const point & a2, const point & v2){
  //lines a1+tv1, a2+tv2
  //assuming that they intersect
  ld det = v1.cross(v2);
  return a1 + v1 * ((a2 - a1).cross(v2) / det);
}
int intersectLineSegmentInfo(const point & a, const point & v, const point & c, const point & d){
  //line a+tv, segment cd
  point v2 = d - c;
  ld det = v.cross(v2);
  if(eq(det, 0)){
    if(eq((c - a).cross(v), 0)){
      return -1; //infinity points
    }else{
      return 0; //no point
    }
  }else{
    return sgn(v.cross(c - a)) != sgn(v.cross(d - a)); //1: single point, 0: no point
  }
}
int intersectSegmentsInfo(const point & a, const point & b, const point & c, const point & d){
  //segment ab, segment cd
  point v1 = b - a, v2 = d - c;
  int t = sgn(v1.cross(c - a)), u = sgn(v1.cross(d - a));
  if(t == u){
    if(t == 0){
      if(pointInSegment(a, b, c) || pointInSegment(a, b, d) || pointInSegment(c, d, a) || pointInSegment(c, d, b)){
        return -1; //infinity points
      }else{
        return 0; //no point
      }
    }else{
      return 0; //no point
    }
  }else{
    return sgn(v2.cross(a - c)) != sgn(v2.cross(b - c)); //1: single point, 0: no point
  }
}
ld distancePointLine(const point & a, const point & v, const point & p){
  //line: a + tv, point p
  return abs(v.cross(p - a)) / v.length();
}
ld perimeter(vector<point> & P){
  int n = P.size();
  ld ans = 0;
  for(int i = 0; i < n; i++){
    ans += (P[i] - P[(i + 1) % n]).length();
  }
  return ans;
}
ld area(vector<point> & P){
  int n = P.size();
  ld ans = 0;
  for(int i = 0; i < n; i++){
```

```
    ans += P[i].cross(P[(i + 1) % n]);
  }
  return abs(ans / 2);
}
vector<point> convexHull(vector<point> P){
  sort(P.begin(), P.end());
  vector<point> L, U;
  for(int i = 0; i < P.size(); i++){
    while(L.size() >= 2 && leq((L[L.size() - 2] - P[i]).cross(L[L.size() - 1] - P[i]), 0)){
      L.pop_back();
    }
    L.push_back(P[i]);
  }
  for(int i = P.size() - 1; i >= 0; i--){
    while(U.size() >= 2 && leq((U[U.size() - 2] - P[i]).cross(U[U.size() - 1] - P[i]), 0)){
      U.pop_back();
    }
    U.push_back(P[i]);
  }
  L.pop_back();
  U.pop_back();
  L.insert(L.end(), U.begin(), U.end());
  return L;
}
bool pointInPerimeter(const vector<point> & P, const point & p){
  int n = P.size();
  for(int i = 0; i < n; i++){
    if(pointInSegment(P[i], P[(i + 1) % n], p)){
      return true;
    }
  }
  return false;
}
bool crossesRay(const point & a, const point & b, const point & p){
  return (geq(b.y, p.y) - geq(a.y, p.y)) * sgn((a - p).cross(b - p)) > 0;
}
int pointInPolygon(const vector<point> & P, const point & p){
  if(pointInPerimeter(P, p)){
    return -1; //point in the perimeter
  }
  int n = P.size();
  int rays = 0;
  for(int i = 0; i < n; i++){
    rays += crossesRay(P[i], P[(i + 1) % n], p);
  }
  return rays & 1; //0: point outside, 1: point inside
}
//point in convex polygon in O(log n)
//make sure that P is convex and in ccw
//before the queries, do the preprocess on P:
// rotate(P.begin(), min_element(P.begin(), P.end()), P.end());
// int right = max_element(P.begin(), P.end()) - P.begin();
//returns 0 if p is outside, 1 if p is inside, -1 if p is in the perimeter
int pointInConvexPolygon(const vector<point> & P, const point & p, int right){
  if(p < P[0] || P[right] < p) return 0;
  int orientation = sgn((P[right] - P[0]).cross(p - P[0]));
  if(orientation == 0){
    if(p == P[0] || p == P[right]) return -1;
    return (right == 1 || right + 1 == P.size()) ? -1 : 1;
  }else if(orientation < 0){
    auto r = lower_bound(P.begin() + 1, P.begin() + right, p);
    int det = sgn((p - r[-1]).cross(r[0] - r[-1])) - 1;
    if(det == -2) det = 1;
    return det;
  }else{
    auto l = upper_bound(P.rbegin(), P.rend() - right - 1, p);
    int det = sgn((p - l[0]).cross((l == P.rbegin() ? P[0] : l[-1]) - l[0])) - 1;
    if(det == -2) det = 1;
    return det;
  }
}
vector<point> cutPolygon(const vector<point> & P, const point & a, const point & v){
  //returns the part of the convex polygon P on the left side of line a+tv
  int n = P.size();
  vector<point> lhs;
  for(int i = 0; i < n; ++i){
    if(geq(v.cross(P[i] - a), 0)){
      lhs.push_back(P[i]);
    }
    if(intersectLineSegmentInfo(a, v, P[i], P[(i+1)%n]) == 1){
      point p = intersectLines(a, v, P[i], P[(i+1)%n] - P[i]);
      if(p != P[i] && p != P[(i+1)%n]){
        lhs.push_back(p);
      }
    }
  }
  return lhs;
}
point centroid(vector<point> & P){
  point num;
  ld den = 0;
  int n = P.size();
  for(int i = 0; i < n; ++i){
    ld cross = P[i].cross(P[(i + 1) % n]);
    num += (P[i] + P[(i + 1) % n]) * cross;
    den += cross;
  }
  return num / (3 * den);
}
vector<pair<int, int>> antipodalPairs(vector<point> & P){
  vector<pair<int, int>> ans;
  int n = P.size(), k = 1;
  auto f = [&](int u, int v, int w){return abs((P[v%n]-P[u%n]).cross(P[w%n]-P[u%n]));};
  while(ge(f(n-1, 0, k+1), f(n-1, 0, k))) ++k;
  for(int i = 0, j = k; i <= k && j < n; ++i){
    ans.emplace_back(i, j);
    while(j < n-1 && ge(f(i, i+1, j+1), f(i, i+1, j)))
      ans.emplace_back(i, ++j);
  }
  return ans;
}
pair<ld, ld> diameterAndWidth(vector<point> & P){
  int n = P.size(), k = 0;
  auto dot = [&](int a, int b){return (P[(a+1)%n]-P[a]).dot(P[(b+1)%n]-P[b]);};
  auto cross = [&](int a, int b){return (P[(a+1)%n]-P[a]).cross(P[(b+1)%n]-P[b]);};
  ld diameter = 0;
  ld width = inf;
  while(ge(dot(0, k), 0)) k = (k+1) % n;
  for(int i = 0; i < n; ++i){
    while(ge(cross(i, k), 0)) k = (k+1) % n;
    //pair: (i, k)
    diameter = max(diameter, (P[k] - P[i]).length());
    width = min(width, distancePointLine(P[i], P[(i+1)%n] - P[i], P[k]));
  }
  return {diameter, width};
}
pair<ld, ld> smallestEnclosingRectangle(vector<point> & P){
  int n = P.size();
  auto dot = [&](int a, int b){return (P[(a+1)%n]-P[a]).dot(P[(b+1)%n]-P[b]);};
  auto cross = [&](int a, int b){return (P[(a+1)%n]-P[a]).cross(P[(b+1)%n]-P[b]);};
  ld perimeter = inf, area = inf;
  for(int i = 0, j = 0, k = 0, m = 0; i < n; ++i){
    while(ge(dot(i, j), 0)) j = (j+1) % n;
    if(!i) k = j;
    while(ge(cross(i, k), 0)) k = (k+1) % n;
```

```
  if(!i) m = k;
  while(le(dot(i, m), 0)) m = (m+1) % n;
  //pairs: (i, k) , (j, m)
  point v = P[(i+1)%n] - P[i];
  ld h = distancePointLine(P[i], v, P[k]);
  ld w = distancePointLine(P[j], v.perp(), P[m]);
  perimeter = min(perimeter, 2 * (h + w));
  area = min(area, h * w);
 }
 return {area, perimeter};
}
ld distancePointCircle(const point & c, ld r, const point & p){
 //point p, circle with center c and radius r
 return max((ld)0, (p - c).length() - r);
}
point projectionPointCircle(const point & c, ld r, const point & p){
 //point p (outside the circle), circle with center c and radius r
 return c + (p - c).unit() * r;
}
pair<point, point> pointsOfTangency(const point & c, ld r, const point & p){
 //point p (outside the circle), circle with center c and radius r
 point v = (p - c).unit() * r;
 ld d2 = (p - c).norm(), d = sqrt(d2);
 point v1 = v * (r / d), v2 = v.perp() * (sqrt(d2 - r*r) / d);
 return {c + v1 - v2, c + v1 + v2};
}
vector<point> intersectLineCircle(const point & a, const point & v, const point & c, ld r){
 //line a+tv, circle with center c and radius r
 ld h2 = r*r - v.cross(c - a) * v.cross(c - a) / v.norm();
 point p = a + v * v.dot(c - a) / v.norm();
 if(eq(h2, 0)) return {p}; //line tangent to circle
 else if(le(h2, 0)) return {}; //no intersection
 else{
  point u = v.unit() * sqrt(h2);
  return {p - u, p + u}; //two points of intersection (chord)
 }
}
vector<point> intersectSegmentCircle(const point & a, const point & b, const point & c, ld r){
 //segment ab, circle with center c and radius r
 vector<point> P = intersectLineCircle(a, b - a, c, r), ans;
 for(const point & p : P){
  if(pointInSegment(a, b, p)) ans.push_back(p);
 }
 return ans;
}
pair<point, ld> getCircle(const point & m, const point & n, const point & p){
 //find circle that passes through points p, q, r
 point c = intersectLines((n + m) / 2, (n - m).perp(), (p + n) / 2, (p - n).perp());
 ld r = (c - m).length();
 return {c, r};
}
vector<point> intersectionCircles(const point & c1, ld r1, const point & c2, ld r2){
 //circle 1 with center c1 and radius r1
 //circle 2 with center c2 and radius r2
 point d = c2 - c1;
 ld d2 = d.norm();
 if(eq(d2, 0)) return {}; //concentric circles
 ld pd = (d2 + r1*r1 - r2*r2) / 2;
 ld h2 = r1*r1 - pd*pd/d2;
 point p = c1 + d*pd/d2;
 if(eq(h2, 0)) return {p}; //circles touch at one point
 else if(le(h2, 0)) return {}; //circles don't intersect
 else{
  point u = d.perp() * sqrt(h2/d2);
  return {p - u, p + u};
 }
}
int circleInsideCircle(const point & c1, ld r1, const point & c2, ld r2) {
 //test if circle 2 is inside circle 1
 //returns "-1" if 2 touches internally 1, "1" if 2 is inside 1, "0" if they overlap
 ld l = r1 - r2 - (c1 - c2).length();
 return (ge(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
}
int circleOutsideCircle(const point & c1, ld r1, const point & c2, ld r2 ){
 //test if circle 2 is outside circle 1
 //returns "-1" if they touch externally, "1" if 2 is outside 1, "0" if they overlap
 ld l = (c1 - c2).length() - (r1 + r2);
 return (ge(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
}
int pointInCircle(const point & c, ld r, const point & p){
 //test if point p is inside the circle with center c and radius r
 //returns "0" if it's outside, "-1" if it's in the perimeter, "1" if it's inside
 ld l = (p - c).length() - r;
 return (le(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
}
vector<vector<point>> tangents(const point & c1, ld r1, const point & c2, ld r2, bool inner){
 //returns a vector of segments or a single point
 if(inner) r2 = -r2;
 point d = c2 - c1;
 ld dr = r1 - r2, d2 = d.norm(), h2 = d2 - dr*dr;
 if(eq(d2, 0) || le(h2, 0)) return {};
 point v = d*dr/d2;
 if(eq(h2, 0)) return {{c1 + v*r1}};
 else{
  point u = d.perp()*sqrt(h2)/d2;
  return {{c1 + (v - u)*r1, c2 + (v - u)*r2}, {c1 + (v + u)*r1, c2 + (v + u)*r2}};
 }
}
ld signed_angle(const point & a, const point & b){
 return sgn(a.cross(b)) * acosl(a.dot(b) / (a.length() * b.length()));
}
ld intersectPolygonCircle(const vector<point> & P, const point & c, ld r ){
 //Gets the area of the intersection of the polygon with the circle
 int n = P.size();
 ld ans = 0;
 for(int i = 0; i < n; ++i){
  point p = P[i], q = P[(i+1)%n];
  bool p_inside = (pointInCircle(c, r, p) != 0);
  bool q_inside = (pointInCircle(c, r, q) != 0);
  if(p_inside && q_inside){
   ans += (p - c).cross(q - c);
  }else if(p_inside && !q_inside){
   point s1 = intersectSegmentCircle(p, q, c, r)[0];
   point s2 = intersectSegmentCircle(c, q, c, r)[0];
   ans += (p - c).cross(s1 - c) + r*r * signed_angle(s1 - c, s2 - c);
  }else if(!p_inside && q_inside){
   point s1 = intersectSegmentCircle(c, p, c, r)[0];
   point s2 = intersectSegmentCircle(p, q, c, r)[0];
   ans += (s2 - c).cross(q - c) + r*r * signed_angle(s1 - c, s2 - c);
  }else{
   auto info = intersectSegmentCircle(p, q, c, r);
   if(info.size() <= 1){
    ans += r*r * signed_angle(p - c, q - c);
   }else{
    point s2 = info[0], s3 = info[1];
    point s1 = intersectSegmentCircle(c, p, c, r)[0];
    point s4 = intersectSegmentCircle(c, q, c, r)[0];
    ans += (s2 - c).cross(s3 - c) + r*r * (signed_angle(s1 - c, s2 - c) + signed_angle(s3 - c, s4 - c));
   }
  }
```

```
  }
  return abs(ans)/2;
}
pair<point, ld> mec2(vector<point> & S, const point & a, const point & b
    ↪ , int n){
  ld hi = inf, lo = -hi;
  for(int i = 0; i < n; ++i){
    ld si = (b - a).cross(S[i] - a);
    if(eq(si, 0)) continue;
    point m = getCircle(a, b, S[i]).first;
    ld cr = (b - a).cross(m - a);
    if(le(si, 0)) hi = min(hi, cr);
    else lo = max(lo, cr);
  }
  ld v = (ge(lo, 0) ? lo : le(hi, 0) ? hi : 0);
  point c = (a + b) / 2 + (b - a).perp() * v / (b - a).norm();
  return {c, (a - c).norm()};
}
pair<point, ld> mec(vector<point> & S, const point & a, int n){
  random_shuffle(S.begin(), S.begin() + n);
  point b = S[0], c = (a + b) / 2;
  ld r = (a - c).norm();
  for(int i = 1; i < n; ++i){
    if(ge((S[i] - c).norm(), r)){
      tie(c, r) = (n == S.size() ? mec(S, S[i], i) : mec2(S, a, S[i], i));
    }
  }
  return {c, r};
}
pair<point, ld> smallestEnclosingCircle(vector<point> S){
  assert(!S.empty());
  auto r = mec(S, S[0], S.size());
  return {r.first, sqrt(r.second)};
}
bool comp1(const point & a, const point & b){
  return le(a.y, b.y);
}
pair<point, point> closestPairOfPoints(vector<point> P){
  sort(P.begin(), P.end(), comp1);
  set<point> S;
  ld ans = inf;
  point p, q;
  int pos = 0;
  for(int i = 0; i < P.size(); ++i){
    while(pos < i && geq(P[i].y - P[pos].y, ans)){
      S.erase(P[pos++]);
    }
    auto lower = S.lower_bound({P[i].x - ans - eps, -inf});
    auto upper = S.upper_bound({P[i].x + ans + eps, -inf});
    for(auto it = lower; it != upper; ++it){
      ld d = (P[i] - *it).length();
      if(le(d, ans)){
        ans = d;
        p = P[i];
        q = *it;
      }
    }
    S.insert(P[i]);
  }
  return {p, q};
}
struct vantage_point_tree{
  struct node
  {
    point p;
    ld th;
    node *l, *r;
  }*root;
  vector<pair<ld, point>> aux;
  vantage_point_tree(vector<point> &ps){
    for(int i = 0; i < ps.size(); ++i)
      aux.push_back({ 0, ps[i] });
    root = build(0, ps.size());
  }
  node *build(int l, int r){
    if(l == r)
      return 0;
    swap(aux[l], aux[l + rand() % (r - l)]);
    point p = aux[l++].second;
    if(l == r)
      return new node({ p });
    for(int i = l; i < r; ++i)
      aux[i].first = (p - aux[i].second).dot(p - aux[i].second);
    int m = (l + r) / 2;
    nth_element(aux.begin() + l, aux.begin() + m, aux.begin() + r);
    return new node({ p, sqrt(aux[m].first), build(l, m), build(m, r) });
  }
  priority_queue<pair<ld, node*>> que;
  void k_nn(node *t, point p, int k){
    if(!t)
      return;
    ld d = (p - t->p).length();
    if(que.size() < k)
      que.push({ d, t });
    else if(ge(que.top().first, d)){
      que.pop();
      que.push({ d, t });
    }
    if(!t->l && !t->r)
      return;
    if(le(d, t->th)){
      k_nn(t->l, p, k);
      if(leq(t->th - d, que.top().first))
        k_nn(t->r, p, k);
    }else{
      k_nn(t->r, p, k);
      if(leq(d - t->th, que.top().first))
        k_nn(t->l, p, k);
    }
  }
  vector<point> k_nn(point p, int k){
    k_nn(root, p, k);
    vector<point> ans;
    for(; !que.empty(); que.pop())
      ans.push_back(que.top().second->p);
    reverse(ans.begin(), ans.end());
    return ans;
  }
};
vector<point> minkowskiSum(vector<point> A, vector<point> B){
  int na = (int)A.size(), nb = (int)B.size();
  if(A.empty() || B.empty()) return {};
  rotate(A.begin(), min_element(A.begin(), A.end()), A.end());
  rotate(B.begin(), min_element(B.begin(), B.end()), B.end());
  int pa = 0, pb = 0;
  vector<point> M;
  while(pa < na && pb < nb){
    M.push_back(A[pa] + B[pb]);
    ld x = (A[(pa + 1) % na] - A[pa]).cross(B[(pb + 1) % nb] - B[pb]);
    if(leq(x, 0)) pb++;
    if(geq(x, 0)) pa++;
  }
  while(pa < na) M.push_back(A[pa++] + B[0]);
  while(pb < nb) M.push_back(B[pb++] + A[0]);
  return M;
}
//Delaunay triangulation in O(n log n)
const point inf_pt(inf, inf);
struct QuadEdge{
  point origin;
  QuadEdge* rot = nullptr;
  QuadEdge* onext = nullptr;
  bool used = false;
  QuadEdge* rev() const{return rot->rot;}
  QuadEdge* lnext() const{return rot->rev()->onext->rot;}
  QuadEdge* oprev() const{return rot->onext->rot;}
  point dest() const{return rev()->origin;}
```

```
};
QuadEdge* make_edge(const point & from, const point & to){
  QuadEdge* e1 = new QuadEdge;
  QuadEdge* e2 = new QuadEdge;
  QuadEdge* e3 = new QuadEdge;
  QuadEdge* e4 = new QuadEdge;
  e1->origin = from;
  e2->origin = to;
  e3->origin = e4->origin = inf_pt;
  e1->rot = e3;
  e2->rot = e4;
  e3->rot = e2;
  e4->rot = e1;
  e1->onext = e1;
  e2->onext = e2;
  e3->onext = e4;
  e4->onext = e3;
  return e1;
}
void splice(QuadEdge* a, QuadEdge* b){
  swap(a->onext->rot->onext, b->onext->rot->onext);
  swap(a->onext, b->onext);
}
void delete_edge(QuadEdge* e){
  splice(e, e->oprev());
  splice(e->rev(), e->rev()->oprev());
  delete e->rot;
  delete e->rev()->rot;
  delete e;
  delete e->rev();
}
QuadEdge* connect(QuadEdge* a, QuadEdge* b){
  QuadEdge* e = make_edge(a->dest(), b->origin);
  splice(e, a->lnext());
  splice(e->rev(), b);
  return e;
}
bool left_of(const point & p, QuadEdge* e){
  return ge((e->origin - p).cross(e->dest() - p), 0);
}
bool right_of(const point & p, QuadEdge* e){
  return le((e->origin - p).cross(e->dest() - p), 0);
}
ld det3(ld a1, ld a2, ld a3, ld b1, ld b2, ld b3, ld c1, ld c2, ld c3) {
  return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3) + a3 * (b1 * c2 - c1 * b2);
}
bool in_circle(const point & a, const point & b, const point & c, const point & d) {
  ld det = -det3(b.x, b.y, b.norm(), c.x, c.y, c.norm(), d.x, d.y, d.norm());
  det += det3(a.x, a.y, a.norm(), c.x, c.y, c.norm(), d.x, d.y, d.norm());
  det -= det3(a.x, a.y, a.norm(), b.x, b.y, b.norm(), d.x, d.y, d.norm());
  det += det3(a.x, a.y, a.norm(), b.x, b.y, b.norm(), c.x, c.y, c.norm());
  return ge(det, 0);
}
pair<QuadEdge*, QuadEdge*> build_tr(int l, int r, vector<point> & P){
  if(r - l + 1 == 2){
    QuadEdge* res = make_edge(P[l], P[r]);
    return {res, res->rev()};
  }
  if(r - l + 1 == 3){
    QuadEdge *a = make_edge(P[l], P[l + 1]), *b = make_edge(P[l + 1], P[r]);
    splice(a->rev(), b);
    int sg = sgn((P[l + 1] - P[l]).cross(P[r] - P[l]));
    if(sg == 0)
      return {a, b->rev()};
    QuadEdge* c = connect(b, a);
    if(sg == 1)
      return {a, b->rev()};
    else
      return {c->rev(), c};
  }
  int mid = (l + r) / 2;
  QuadEdge *ldo, *ldi, *rdo, *rdi;
  tie(ldo, ldi) = build_tr(l, mid, P);
  tie(rdi, rdo) = build_tr(mid + 1, r, P);
  while(true){
    if(left_of(rdi->origin, ldi)){
      ldi = ldi->lnext();
      continue;
    }
    if(right_of(ldi->origin, rdi)){
      rdi = rdi->rev()->onext;
      continue;
    }
    break;
  }
  QuadEdge* basel = connect(rdi->rev(), ldi);
  auto valid = [&basel](QuadEdge* e){return right_of(e->dest(), basel);};
  if(ldi->origin == ldo->origin)
    ldo = basel->rev();
  if(rdi->origin == rdo->origin)
    rdo = basel;
  while(true){
    QuadEdge* lcand = basel->rev()->onext;
    if(valid(lcand)){
      while(in_circle(basel->dest(), basel->origin, lcand->dest(), lcand->onext->dest())){
        QuadEdge* t = lcand->onext;
        delete_edge(lcand);
        lcand = t;
      }
    }
    QuadEdge* rcand = basel->oprev();
    if(valid(rcand)){
      while(in_circle(basel->dest(), basel->origin, rcand->dest(), rcand->oprev()->dest())){
        QuadEdge* t = rcand->oprev();
        delete_edge(rcand);
        rcand = t;
      }
    }
    if(!valid(lcand) && !valid(rcand))
      break;
    if(!valid(lcand) || (valid(rcand) && in_circle(lcand->dest(), lcand->origin, rcand->origin, rcand->dest())))
      basel = connect(rcand, basel->rev());
    else
      basel = connect(basel->rev(), lcand->rev());
  }
  return {ldo, rdo};
}
vector<tuple<point, point, point>> delaunay(vector<point> & P){
  sort(P.begin(), P.end());
  auto res = build_tr(0, (int)P.size() - 1, P);
  QuadEdge* e = res.first;
  vector<QuadEdge*> edges = {e};
  while(le((e->dest() - e->onext->dest()).cross(e->origin - e->onext->dest()), 0))
    e = e->onext;
  auto add = [&P, &e, &edges](){
    QuadEdge* curr = e;
    do{
      curr->used = true;
      P.push_back(curr->origin);
      edges.push_back(curr->rev());
      curr = curr->lnext();
    }while(curr != e);
  };
  add();
  P.clear();
  int kek = 0;
```

```
  while(kek < (int)edges.size())
   if(!(e = edges[kek++])->used)
    add();
  vector<tuple<point, point, point>> ans;
  for(int i = 0; i < (int)P.size(); i += 3){
   ans.emplace_back(P[i], P[i + 1], P[i + 2]);
  }
  return ans;
}
struct circ{
  point c;
  ld r;
  circ() {}
  circ(const point & c, ld r): c(c), r(r) {}
  set<pair<ld, ld>> ranges;
  void disable(ld l, ld r){
   ranges.emplace(l, r);
  }
  auto getActive() const{
   vector<pair<ld, ld>> ans;
   ld maxi = 0;
   for(const auto & dis : ranges){
    ld l, r;
    tie(l, r) = dis;
    if(l > maxi){
     ans.emplace_back(maxi, l);
    }
    maxi = max(maxi, r);
   }
   if(!eq(maxi, 2*pi)){
    ans.emplace_back(maxi, 2*pi);
   }
   return ans;
  }
};
ld areaUnionCircles(const vector<circ> & circs){
  vector<circ> valid;
  for(const circ & curr : circs){
   if(eq(curr.r, 0)) continue;
   circ nuevo = curr;
   for(circ & prev : valid){
    if(circleInsideCircle(prev.c, prev.r, nuevo.c, nuevo.r)){
     nuevo.disable(0, 2*pi);
    }else if(circleInsideCircle(nuevo.c, nuevo.r, prev.c, prev.r)){
     prev.disable(0, 2*pi);
    }else{
     auto cruce = intersectionCircles(prev.c, prev.r, nuevo.c, nuevo.r)
         ↪ ;
     if(cruce.size() == 2){
      ld a1 = (cruce[0] - prev.c).ang();
      ld a2 = (cruce[1] - prev.c).ang();
      ld b1 = (cruce[1] - nuevo.c).ang();
      ld b2 = (cruce[0] - nuevo.c).ang();
      if(a1 < a2){
       prev.disable(a1, a2);
      }else{
       prev.disable(a1, 2*pi);
       prev.disable(0, a2);
      }
      if(b1 < b2){
       nuevo.disable(b1, b2);
      }else{
       nuevo.disable(b1, 2*pi);
       nuevo.disable(0, b2);
      }
     }
    }
   }
   valid.push_back(nuevo);
  }
  ld ans = 0;
  for(const circ & curr : valid){
   for(const auto & range : curr.getActive()){
    ld l, r;
    tie(l, r) = range;
    ans += curr.r*(curr.c.x * (sin(r) - sin(l)) - curr.c.y * (cos(r) -
         ↪ cos(l))) + curr.r*curr.r*(r-l);
   }
  }
  return ans/2;
};
struct plane{
  point a, v;
  plane(): a(), v(){}
  plane(const point& a, const point& v): a(a), v(v){}
  point intersect(const plane& p) const{
   ld t = (p.a - a).cross(p.v) / v.cross(p.v);
   return a + v*t;
  }
  bool outside(const point& p) const{ // test if point p is strictly
      ↪ outside
   return le(v.cross(p - a), 0);
  }
  bool inside(const point& p) const{ // test if point p is inside or in
      ↪ the boundary
   return geq(v.cross(p - a), 0);
  }
  bool operator<(const plane& p) const{ // sort by angle
   auto lhs = make_tuple(v.half({1, 0}), ld(0), v.cross(p.a - a));
   auto rhs = make_tuple(p.v.half({1, 0}), v.cross(p.v), ld(0));
   return lhs < rhs;
  }
  bool operator==(const plane& p) const{ // paralell and same directions,
      ↪ not really equal
   return eq(v.cross(p.v), 0) && ge(v.dot(p.v), 0);
  }
};
vector<point> halfPlaneIntersection(vector<plane> planes){
  planes.push_back({{0, -inf}, {1, 0}});
  planes.push_back({{inf, 0}, {0, 1}});
  planes.push_back({{0, inf}, {-1, 0}});
  planes.push_back({{-inf, 0}, {0, -1}});
  sort(planes.begin(), planes.end());
  planes.erase(unique(planes.begin(), planes.end()), planes.end());
  deque<plane> ch;
  deque<point> poly;
  for(const plane& p : planes){
   while(ch.size() ≥ 2 && p.outside(poly.back())) ch.pop_back(), poly.
       ↪ pop_back();
   while(ch.size() ≥ 2 && p.outside(poly.front())) ch.pop_front(), poly
       ↪ .pop_front();
   if(p.v.half({1, 0}) && poly.empty()) return {};
   ch.push_back(p);
   if(ch.size() ≥ 2) poly.push_back(ch[ch.size()-2].intersect(ch[ch.
       ↪ size()-1]));
  }
  while(ch.size() ≥ 3 && ch.front().outside(poly.back())) ch.pop_back(),
      ↪ poly.pop_back();
  while(ch.size() ≥ 3 && ch.back().outside(poly.front())) ch.pop_front()
      ↪ , poly.pop_front();
  poly.push_back(ch.back().intersect(ch.front()));
  return vector<point>(poly.begin(), poly.end());
}
vector<point> halfPlaneIntersectionRandomized(vector<plane> planes){
  point p = planes[0].a;
  int n = planes.size();
  random_shuffle(planes.begin(), planes.end());
  for(int i = 0; i < n; ++i){
   if(planes[i].inside(p)) continue;
   ld lo = -inf, hi = inf;
   for(int j = 0; j < i; ++j){
    ld A = planes[j].v.cross(planes[i].v);
    ld B = planes[j].v.cross(planes[j].a - planes[i].a);
    if(ge(A, 0)){
     lo = max(lo, B/A);
    }else if(le(A, 0)){
     hi = min(hi, B/A);
```

```
    }else{
      if(ge(B, 0)) return {};
    }
    if(ge(lo, hi)) return {};
  }
  p = planes[i].a + planes[i].v*lo;
 }
 return {p};
}
```

6.4 Angle.h

Angle.h 34 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)b.y);
}
// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

6.5 CircleIntersection.h

CircleIntersection.h 12 lines

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

6.6 CircleLine.h

CircleLine.h 10 lines

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

6.7 CirclePolygonIntersection.h

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

CirclePolygonIntersection.h 7d8898, 20 lines

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

6.8 CircleTangents.h

CircleTangents.h 14 lines

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

6.9 ClosestPair.h

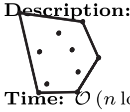
Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$

ClosestPair.h ac9ec9, 18 lines

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

6.10 ConvexHull.h

Description: Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.



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

ConvexHull.h ec8c60, 14 lines

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

6.11 DelaunayTriangulation.h

Description: Computes the Delaunay triangulation of a set of points. Each circumcircle contains none of the input points. If any three points are collinear or any four are on the same circle, behavior is undefined.

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

DelaunayTriangulation.h 3d8be0, 12 lines

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

6.12 FastDelaunay.h

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

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

FastDelaunay.h e8d11f, 83 lines

```
#include "Point.h"
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point
struct Quad {
  Q rot, o; P p = arb; bool mark;
  P& F() { return r()->p; }
  Q& r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
} *H;
bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
  r->p = orig; r->F() = dest;
  return r;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}
pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }
#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
      (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;
#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e->o = H; H = e; e = t; \
    }
  for (;;) {
    DEL(LC, base->r(), o); DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, base->r());
    else
      base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}
vector<P> triangulate(vector<P> pts) {
  sort(all(pts)); assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

6.13 HullDiameter.h

HullDiameter.h 13 lines

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

```
  return res.second;
}
```

6.14 InsidePolygon.h

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

InsidePolygon.h ded0d2, 14 lines

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

6.15 LineHullIntersection.h

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

LineHullIntersection.h 32bb13, 39 lines

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

6.16 LineProjectionReflection.h

LineProjectionReflection.h 6 lines

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

6.17 ManhattanMST.h

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

ManhattanMST.h c264e8, 24 lines

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

6.18 MinimumEnclosingCircle.h

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

MinimumEnclosingCircle.h 0b9ff5, 18 lines

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

6.19 OnSegment.h

OnSegment.h 4 lines

```
#include "Point.h"
```

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

6.20 Point.h

Point.h 28 lines

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

6.21 Point3D.h

Point3D.h 32 lines

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

6.22 PointInsideHull.h

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

PointInsideHull.h 1710a2, 16 lines

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

6.23 PolygonArea.h

PolygonArea.h 7 lines

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

6.24 PolygonCenter.h

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

PolygonCenter.h 6906c1, 10 lines

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

6.25 PolygonCut.h

PolygonCut.h 15 lines

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

6.26 PolygonUnion.h

Description: Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)

Time: $\mathcal{O}\left(N^2\right)$, where N is the total number of points

PolygonUnion.h 6ab2ad, 35 lines

```
#include "Point.h"
#include "sideOf.h"
typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
  double ret = 0;
  rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
    P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
    vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
    rep(j,0,sz(poly)) if (i ≠ j) {
      rep(u,0,sz(poly[j])) {
        P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
        int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
        if (sc ≠ sd) {
          double sa = C.cross(D, A), sb = C.cross(D, B);
          if (min(sc, sd) < 0)
            segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
        } else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>0){
          segs.emplace_back(rat(C - A, B - A), 1);
          segs.emplace_back(rat(D - A, B - A), -1);
        }
      }
    }
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
    double sum = 0;
    int cnt = segs[0].second;
    rep(j,1,sz(segs)) {
      if (!cnt) sum += segs[j].first - segs[j - 1].first;
      cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
  }
  return ret / 2;
}
```

6.27 PolyhedronVolume.h

PolyhedronVolume.h 6 lines

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

6.28 SegmentDistance.h

SegmentDistance.h 7 lines

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

6.29 SegmentIntersection.h

SegmentIntersection.h 15 lines

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

6.30 antipodalPoints.cpp

antipodalPoints.cpp 40 lines

```
/*
  Antipodal points
  Tested: AIZU(judge.u-aizu.ac.jp) CGL.4B
  Complexity: O(n)
*/
vector<pair<int, int>> antipodal(const polygon &P)
{
  vector<pair<int, int>> ans;
  int n = P.size();
  if (P.size() == 2)
    ans.push_back({ 0, 1 });
  if (P.size() < 3)
    return ans;
  int q0 = 0;
  while (abs(area2(P[n - 1], P[0], P[NEXT(q0)]))
     > abs(area2(P[n - 1], P[0], P[q0])))
    ++q0;
  for (int q = q0, p = 0; q ≠ 0 && p ≤ q0; ++p)
  {
    ans.push_back({ p, q });
    while (abs(area2(P[p], P[NEXT(p)], P[NEXT(q)]))
       > abs(area2(P[p], P[NEXT(p)], P[q])))
    {
      q = NEXT(q);
      if (p ≠ q0 || q ≠ 0)
        ans.push_back({ p, q });
      else
        return ans;
    }
    if (abs(area2(P[p], P[NEXT(p)], P[NEXT(q)]))
      == abs(area2(P[p], P[NEXT(p)], P[q])))
    {
      if (p ≠ q0 || q ≠ n - 1)
        ans.push_back({ p, NEXT(q) });
      else
        ans.push_back({ NEXT(p), q });
    }
  }
  return ans;
}
```

6.31 basics.cpp

basics.cpp 31 lines

```
typedef complex<double> point;
typedef vector<point> polygon;
#define NEXT(i) (((i) + 1) % n)
struct circle { point p; double r; };
struct line { point p, q; };
using segment = line;
const double eps = 1e-9;
// fix comparations on doubles with this two functions
int sign(double x) { return x < -eps ? -1 : x > eps; }
int dblcmp(double x, double y) { return sign(x - y); }
double dot(point a, point b) { return real(conj(a) * b); }
double cross(point a, point b) { return imag(conj(a) * b); }
double area2(point a, point b, point c) { return cross(b - a, c - a); }
int ccw(point a, point b, point c)
{
  b -= a; c -= a;
```

```
  if (cross(b, c) > 0) return +1; // counter clockwise
  if (cross(b, c) < 0) return -1; // clockwise
  if (dot(b, c) < 0) return +2; // c--a--b on line
  if (dot(b, b) < dot(c, c)) return -2; // a--b--c on line
  return 0;
}
namespace std
{
 bool operator<(point a, point b)
 {
  if (a.real() != b.real())
   return a.real() < b.real();
  return a.imag() < b.imag();
 }
}
```

6.32 circle.cpp

circle.cpp 119 lines

```
/*
 Circles
 Tested: AIZU
*/
// circle-circle intersection
vector<point> intersect(circle C, circle D)
{
 double d = abs(C.p - D.p);
 if (sign(d - C.r - D.r) > 0) return {}; // too far
 if (sign(d - abs(C.r - D.r)) < 0) return {}; // too close
 double a = (C.r*C.r - D.r*D.r + d*d) / (2*d);
 double h = sqrt(C.r*C.r - a*a);
 point v = (D.p - C.p) / d;
 if (sign(h) == 0) return {C.p + v*a}; // touch
 return {C.p + v*a + point(0,1)*v*h, // intersect
    C.p + v*a - point(0,1)*v*h};
}
// circle-line intersection
vector<point> intersect(line L, circle C)
{
 point u = L.p - L.q, v = L.p - C.p;
 double a = dot(u, u), b = dot(u, v), c = dot(v, v) - C.r*C.r;
 double det = b*b - a*c;
 if (sign(det) < 0) return {};    // no solution
 if (sign(det) == 0) return {L.p - b/a*u}; // touch
 return {L.p + (-b + sqrt(det))/a*u,
    L.p + (-b - sqrt(det))/a*u};
}
// circle tangents through point
vector<point> tangent(point p, circle C)
{
 // not tested enough
 double D = abs(p - C.p);
 if (D + eps < C.r) return {};
 point t = C.p - p;
 double theta = asin( C.r / D );
 double d = cos(theta) * D;
 t = t / abs(t) * d;
 if ( abs(D - C.r) < eps ) return {p + t};
 point rot( cos(theta), sin(theta) );
 return {p + t * rot, p + t * conj(rot)};
}
bool incircle(point a, point b, point c, point p)
{
 a -= p; b -= p; c -= p;
 return norm(a) * cross(b, c)
    + norm(b) * cross(c, a)
    + norm(c) * cross(a, b) >= 0;
    // < : inside, = cocircular, > outside
}
point three_point_circle(point a, point b, point c)
{
 point x = 1.0 / conj(b - a), y = 1.0 / conj(c - a);
 return (y - x) / (conj(x) * y - x * conj(y)) + a;
}
/*
  Get the center of the circles that pass through p0 and p1
  and has ratio r.
  Be careful with epsilon.
*/
vector<point> two_point_ratio_circle(point p0, point p1, double r){
  if (abs(p1 - p0) > 2 * r + eps) // Points are too far.
    return {};
  point pm = (p1 + p0) / 2.0l;
  point pv = p1 - p0;
  pv = point(-pv.imag(), pv.real());
  double x1 = p1.real(), y1 = p1.imag();
  double xm = pm.real(), ym = pm.imag();
  double xv = pv.real(), yv = pv.imag();
  double A = (sqr(xv) + sqr(yv));
  double C = sqr(xm - x1) + sqr(ym - y1) - sqr(r);
  double D = sqrt( - 4 * A * C );
  double t = D / 2.0 / A;
  if (abs(t) <= eps)
    return {pm};
  return {c1, c2};
}
/*
 Area of the intersection of a circle with a polygon
 Circle's center lies in (0, 0)
 Polygon must be given counterclockwise
 Tested: LightOJ 1358
 Complexity: O(n)
*/
#define x(_t) (xa + (_t) * a)
#define y(_t) (ya + (_t) * b)
double radian(double xa, double ya, double xb, double yb)
{
 return atan2(xa * yb - xb * ya, xa * xb + ya * yb);
}
double part(double xa, double ya, double xb, double yb, double r)
{
 double l = sqrt((xa - xb) * (xa - xb) + (ya - yb) * (ya - yb));
 double a = (xb - xa) / l, b = (yb - ya) / l, c = a * xa + b * ya;
 double d = 4.0 * (c * c - xa * xa - ya * ya + r * r);
 if (d < eps)
  return radian(xa, ya, xb, yb) * r * r * 0.5;
 else
 {
  d = sqrt(d) * 0.5;
  double s = -c - d, t = -c + d;
  if (s < 0.0) s = 0.0;
  else if (s > l) s = l;
  if (t < 0.0) t = 0.0;
  else if (t > l) t = l;
  return (x(s) * y(t) - x(t) * y(s)
    + (radian(xa, ya, x(s), y(s))
    + radian(x(t), y(t), xb, yb)) * r * r) * 0.5;
 }
}
double intersection_circle_polygon(const polygon &P, double r)
{
 double s = 0.0;
 int n = P.size();;
 for (int i = 0; i < n; i++)
  s += part(P[i].real(), P[i].imag(),
   P[NEXT(i)].real(), P[NEXT(i)].imag(), r);
 return fabs(s);
}
```

6.33 circumcircle.h

circumcircle.h 10 lines

```
#include "Point.h"
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
 return (B-A).dist()*(C-B).dist()*(A-C).dist()/
   abs((B-A).cross(C-A))/2;
```

```
}
P ccCenter(const P& A, const P& B, const P& C) {
 P b = C-A, c = B-A;
 return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

6.34 geometryComplex.cpp

geometryComplex.cpp 246 lines

```
#include <bits/stdc++.h>
using namespace std;
#define _ ios_base::sync_with_stdio(false),cin.tie(NULL);
#define endl '\n'
#define x real()
#define y imag()
typedef double T;
typedef complex<T> pt;
//Basics
T dot(pt v, pt w) {return (conj(v)*w).x;} // v.x*w.x + v.y*w.y;
T cross(pt v, pt w) {return (conj(v)*w).y;} // {v.x*w.y - v.y*w.x;}
T sq(pt p) {return p.x*p.x + p.y*p.y;}
double abs(pt p) {return sqrt(sq(p));}
// 0 if is colinear (or are in the line AB), >0 if is left turn , <0 if is a right turn
T orient(pt a, pt b, pt c) {return cross(b-a,c-a);}
//Translations
pt translate(pt v,pt p){return p+v;}
pt scale(pt c ,double factor,pt p){return c + (p-c)*factor;}
pr rotate(pt p,double a){return p * polar(1.0,a);}
pt perp(pt p){return {-p.y,p.x};} //rotate 90°
pt linearTransfo(pt p, pt q, pt r, pt fp, pt fq) {
   return fp + (r-p) * (fq-fp) / (q-p);
}
bool isPerp(pt v, pt w) {return dot(v,w) == 0;}
double angle(pt v, pt w) {
   double cosTheta = dot(v,w) / abs(v) / abs(w);
   return acos(max(-1.0, min(1.0, cosTheta)));
}
// Angle betwen the lines AB and AC in oriented way
double orientedAngle(pt a, pt b, pt c) {
   if (orient(a,b,c) >= 0)
   return angle(b-a, c-a);
   else
   return 2*M_PI - angle(b-a, c-a);
}
// Check if a point is betwen the angle formed by the lines AB ,AC
bool inAngle(pt a, pt b, pt c, pt p) {
   assert(orient(a,b,c) != 0);
   if (orient(a,b,c) < 0) swap(b,c);
   return orient(a,b,p) >= 0 && orient(a,c,p) <= 0;
}
// If we want some vector v to be the first angle
pt v = {1e9,0}; // diferent from {0,0}
bool half(pt p) {
   return cross(v,p) < 0 || (cross(v,p) == 0 && dot(v,p) <0);
}
// bool half(pt p) { // true if in blue half
//   assert(p.x != 0 || p.y != 0); // the argument of (0,0) is undefined
//   return p.y > 0 || (p.y == 0 && p.x < 0);
// }
void polarSortAround(pt o, vector<pt> &v) {
   sort(v.begin(), v.end(), [](pt v, pt w) {
      return make_tuple(half(v-o), 0) <make_tuple(half(w-o), cross(v-o, w-o));
   });
}
struct line {
   pt v; T c;
   pt p,q;
   // T a,b,c;
   // From direction vector v and offset c
   line(pt v, T c) : v(v), c(c) {}
   // From equation ax+by=c
   line(T a, T b, T c) : v({b,-a}), c(c) {}
   // From points P and Q
   line(pt p, pt q) : v(q-p), c(cross(v,p)) ,p(p),q(q){}
   // Will be defined later:
   // - these work with T = int
   T side(pt p) {return cross(v,p)-c;}
   double dist(pt p) {return abs(side(p)) / abs(v);}
   double sqDist(pt p) {return side(p)*side(p) / (double)sq(v);}
   line perpThrough(pt p) {return {p, p + perp(v)};}
   bool cmpProj(pt p, pt q) {
      return dot(v,p) < dot(v,q);
   }
   line translate(pt t) {return {v, c + cross(v,t)};}
   // - these require T = double
   line shiftLeft(double dist) {return {v, c + dist*abs(v)};}
   pt proj(pt p) {return p - perp(v)*side(p)/sq(v);}
   pt refl(pt p) {return p - perp(v)*2*side(p)/sq(v);}
}
//Mapping pendients
map<pair<int,int>,map<int,set<int>> mp;
void add(pt a,pt b,int ida,int idb){
   int A = a.y-b.y;
   int B = a.x-b.x;
   int gcd = __gcd(A,B);
   A/=gcd;
   B/=gcd;
   C = (a.x*A) - (a.y*B);
   mp[{A,B}][C].insert(ida);
   mp[{A,B}][C].insert(idb);
}
bool intersect(line l1, line l2, pt &out) {
   T d = cross(l1.v, l2.v);
   if (d == 0) return false;
   out = (l2.v*l1.c - l1.v*l2.c) / d; // requires floating-point coordinates
   return true;
}
// Is a line that forms equal angles with l1 and l2
line bisector(line l1, line l2, bool interior) {
   assert(cross(l1.v, l2.v) != 0); // l1 and l2 cannot be parallel!
   double sign = interior ? 1 : -1;
   return {l2.v/abs(l2.v) + l1.v/abs(l1.v) * sign,
   l2.c/abs(l2.v) + l1.c/abs(l1.v) * sign};
}
// Segments
// Check if a point are int the circle of radius AB , if the angle is less of 90° is inside
bool pointInDisk(pt a, pt b, pt p) {
   return dot(a-p, b-p) <= 0;
}
bool pointInSegment(pt a, pt b, pt p) {
   return orient(a,b,p) == 0 && pointInDisk(a,b,p);
}
bool properInter(pt a, pt b, pt c, pt d, pt &out) {
   double oa = orient(c,d,a),
   ob = orient(c,d,b),
   oc = orient(a,b,c),
   od = orient(a,b,d);
   if (oa*ob < 0 && oc*od < 0) {
   out = (a*ob - b*oa) / (ob-oa);
   return true;
   }
   return false;
}
struct cmpX {
   bool operator()(pt a, pt b) {
      return make_pair(a.x, a.y) < make_pair(b.x, b.y);
   }
};
set<pt,cmpX> inters(pt a, pt b, pt c, pt d) {
   pt out;
   if (properInter(a,b,c,d,out)) return {out};
   set<pt,cmpX> s;
   if (onSegment(c,d,a)) s.insert(a);
```

```
  if (onSegment(c,d,b)) s.insert(b);
  if (onSegment(a,b,c)) s.insert(c);
  if (onSegment(a,b,d)) s.insert(d);
  return s;
}
double closestDistanceSegmentPoint(pt a, pt b, pt p) {
  if (a != b) {
    line l(a,b);
    if (l.cmpProj(a,p) && l.cmpProj(p,b)) // if closest to projection
    return l.dist(p); // output distance to line
  }
  return min(abs(p-a), abs(p-b)); // otherwise distance to A or B
}
double closestDistanceSegmentSegment(pt a, pt b, pt c, pt d) {
  pt dummy;
  if (properInter(a,b,c,d,dummy))
  return 0;
  return min({segPoint(a,b,c), segPoint(a,b,d),
  segPoint(c,d,a), segPoint(c,d,b)});
}
/** Polygons */
double areaTriangle(pt a, pt b, pt c) {
  return abs(cross(b-a, c-a)) / 2.0;
}
double areaPolygon(vector<pt> p) {
  double area = 0.0;
  for (int i = 0, n = p.size(); i < n; i++) {
  area += cross(p[i], p[(i+1)%n]); // wrap back to 0 if i == n-1
  }
  return abs(area) / 2.0;
}
bool above(pt a, pt p) {
  return p.y >= a.y;
}
// check if [PQ] crosses ray from A
bool crossesRay(pt a, pt p, pt q) {
  return (above(a,q) - above(a,p)) * orient(a,p,q) > 0;
}
// if strict, returns false when A is on the boundary
bool inPolygon(vector<pt> p, pt a, bool strict = true) {
  int numCrossings = 0;
  for (int i = 0, n = p.size(); i < n; i++) {
    if (pointInSegment(p[i], p[(i+1)%n], a))
      return !strict;
    numCrossings += crossesRay(a, p[i], p[(i+1)%n]);
  }
  return numCrossings & 1; // inside if odd number of crossings
}
// amplitude travelled around point A, from P to Q
double angleTravelled(pt a, pt p, pt q) {
  double ampli = angle(p-a, q-a);
  if (orient(a,p,q) > 0) return ampli;
  else return -ampli;
}
double angleTravelled(pt a, pt p, pt q) {
  // remainder ensures the value is in [-pi,pi]
  return remainder(arg(q-a) - arg(p-a), 2*M_PI);
}
int windingNumber(vector<pt> p, pt a) {
  double ampli = 0;
  for (int i = 0, n = p.size(); i < n; i++)
  ampli += angleTravelled(a, p[i], p[(i+1)%n]);
  return round(ampli / (2*M_PI));
}
/** Circles */
pt circumCenter(pt a, pt b, pt c) {
  b = b-a, c = c-a; // consider coordinates relative to A
  assert(cross(b,c) != 0); // no circumcircle if A,B,C aligned
  return a + perp(b*sq(c) - c*sq(b))/cross(b,c)/2;
}
int circleLine(pt o, double r, line l, pair<pt,pt> &out) {
  double h2 = r*r - l.sqDist(o);
  if (h2 >= 0) { // the line touches the circle
  pt p = l.proj(o); // point P
  pt h = l.v*sqrt(h2)/abs(l.v); // vector parallel to l, of
  length h
  out = {p-h, p+h};
  }
  return 1 + sgn(h2);
}
int circleCircle(pt o1, double r1, pt o2, double r2, pair<pt,pt> &out) {
  pt d=o2-o1; double d2=sq(d);
  if (d2 == 0) {assert(r1 != r2); return 0;} // concentric circles
  double pd = (d2 + r1*r1 - r2*r2)/2; // = |O_1P| * d
  double h2 = r1*r1 - pd*pd/d2; // = ^h2
  if (h2 >= 0) {
  pt p = o1 + d*pd/d2, h = perp(d)*sqrt(h2/d2);
  out = {p-h, p+h};
  }
  return 1 + sgn(h2);
}
int tangents(pt o1, double r1, pt o2, double r2, bool inner, vector<pair<pt,pt>> &out) {
  if (inner) r2 = -r2;
  pt d = o2-o1;
  double dr = r1-r2, d2 = sq(d), h2 = d2-dr*dr;
  if (d2 == 0 || h2 < 0) {assert(h2 != 0); return 0;}
  for (double sign : {-1,1}) {
  pt v = (d*dr + perp(d)*sqrt(h2)*sign)/d2;
  out.push_back({o1 + v*r1, o2 + v*r2});
  }
  return 1 + (h2 > 0);
}
int main(){
  pt p{3,-4};
  cout<<p.x<<" "<<p.y<<endl;
  cout<<p<<endl;
  pt a{3,1}, b{1,-2};
  a += 2.0*b;
  cout<<a<<endl;
  return 0;
}
```

6.35 kdTree.h

kdTree.h 55 lines

```
#include "Point.h"
typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();
bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }
struct Node {
 P pt; // if this is a leaf, the single point in it
 T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
 Node *first = 0, *second = 0;
 T distance(const P& p) { // min squared distance to a point
  T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
  T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
  return (P(x,y) - p).dist2();
 }
 Node(vector<P>&& vp) : pt(vp[0]) {
  for (P p : vp) {
   x0 = min(x0, p.x); x1 = max(x1, p.x);
   y0 = min(y0, p.y); y1 = max(y1, p.y);
  }
  if (vp.size() > 1) {
   // split on x if width >= height (not ideal ...)
   sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
   // divide by taking half the array for each child (not
   // best performance with many duplicates in the middle)
   int half = sz(vp)/2;
   first = new Node({vp.begin(), vp.begin() + half});
   second = new Node({vp.begin() + half, vp.end()});
  }
 }
};
```

```
struct KDTree {
  Node* root;
  KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}
  pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
      // uncomment if we should not find the point itself:
      // if (p == node->pt) return {INF, P()};
      return make_pair((p - node->pt).dist2(), node->pt);
    }
    Node *f = node->first, *s = node->second;
    T bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);
    // search closest side first, other side if needed
    auto best = search(f, p);
    if (bsec < best.first)
      best = min(best, search(s, p));
    return best;
  }
  // find nearest point to a point, and its squared distance
  // (requires an arbitrary operator< for Point)
  pair<T, P> nearest(const P& p) {
    return search(root, p);
  }
};
```

6.36 lineDistance.h

lineDistance.h 5 lines

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

6.37 lineIntersection.h

lineIntersection.h 9 lines

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

6.38 lineSegmentSntersections.cpp

lineSegmentSntersections.cpp 77 lines

```
/*
 Line and segments predicates
 Tested: AIZU(judge.u-aizu.ac.jp) CGL
*/
bool intersectLL(const line &l, const line &m)
{
 return abs(cross(l.q - l.p, m.q - m.p)) > eps || // non-parallel
     abs(cross(l.q - l.p, m.p - l.p)) < eps; // same line
}
bool intersectLS(const line &l, const segment &s)
{
 return cross(l.q - l.p, s.p - l.p) * // s[0] is left of l
     cross(l.q - l.p, s.q - l.p) < eps; // s[1] is right of l
}
bool intersectLP(const line &l, const point &p)
{
 return abs(cross(l.q - p, l.p - p)) < eps;
}
bool intersectSS(const segment &s, const segment &t)
{
 return ccw(s.p, s.q, t.p) * ccw(s.p, s.q, t.q) ≤ 0
     && ccw(t.p, t.q, s.p) * ccw(t.p, t.q, s.q) ≤ 0;
}
bool intersectSP(const segment &s, const point &p)
{
 return abs(s.p - p) + abs(s.q - p) - abs(s.q - s.p) < eps;
 // triangle inequality
 return min(real(s.p), real(s.q)) ≤ real(p)
     && real(p) ≤ max(real(s.p), real(s.q))
     && min(imag(s.p), imag(s.q)) ≤ imag(p)
     && imag(p) ≤ max(imag(s.p), imag(s.q))
     && cross(s.p - p, s.q - p) == 0;
}
point projection(const line &l, const point &p)
{
 double t = dot(p - l.p, l.p - l.q) / norm(l.p - l.q);
 return l.p + t * (l.p - l.q);
}
point reflection(const line &l, const point &p)
{
 return p + 2.0 * (projection(l, p) - p);
}
double distanceLP(const line &l, const point &p)
{
 return abs(p - projection(l, p));
}
double distanceLL(const line &l, const line &m)
{
 return intersectLL(l, m) ? 0 : distanceLP(l, m.p);
}
double distanceLS(const line &l, const line &s)
{
 if (intersectLS(l, s)) return 0;
 return min(distanceLP(l, s.p), distanceLP(l, s.q));
}
double distanceSP(const segment &s, const point &p)
{
 const point r = projection(s, p);
 if (intersectSP(s, r)) return abs(r - p);
 return min(abs(s.p - p), abs(s.q - p));
}
double distanceSS(const segment &s, const segment &t)
{
 if (intersectSS(s, t)) return 0;
 return min(min(distanceSP(s, t.p), distanceSP(s, t.q)),
     min(distanceSP(t, s.p), distanceSP(t, s.q)));
}
point crosspoint(const line &l, const line &m)
{
 double A = cross(l.q - l.p, m.q - m.p);
 double B = cross(l.q - l.p, l.q - m.p);
 if (abs(A) < eps && abs(B) < eps)
  return m.p; // same line
 if (abs(A) < eps)
  assert(false); // !!!PRECONDITION NOT SATISFIED!!!
 return m.p + B / A * (m.q - m.p);
}
```

6.39 linearTransformation.h

linearTransformation.h 7 lines

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

6.40 polygon-area-union.cpp

polygon-area-union.cpp 130 lines

```
// modified from syntax_error's code
bool operator<(const point_t &a, const point_t &b) {
  if (dblcmp(a.x - b.x) == 0) return a.y < b.y;
```

```
  return a.x < b.x;
}
bool operator==(const point_t &a, const point_t &b) {
  return dblcmp(a.x - b.x) == 0 && dblcmp(a.y - b.y) == 0;
}
struct segment_t {
  point_t a, b;
  segment_t() { a = b = point_t(); }
  segment_t(point_t ta, point_t tb) : a(ta), b(tb) { }
  double len() const { return dist(a, b); }
  double k() const { return (a.y - b.y) / (a.x - b.x); }
  double l() const { return a.y - k() * a.x; }
};
struct line_t {
  double a, b, c;
  line_t(point_t p) { a = p.x, b = -1.0, c = -p.y; }
  line_t(point_t p, point_t q) {
    a = p.y - q.y;
    b = q.x - p.x;
    c = a * p.x + b * p.y;
  }
};
bool ccutl(line_t p, line_t q) {
  if (dblcmp(p.a * q.b - q.a * p.b) == 0) return false;
  return true;
}
point_t cutl(line_t p, line_t q) {
  double x = (p.c * q.b - q.c * p.b) / (p.a * q.b - q.a * p.b);
  double y = (p.c * q.a - q.c * p.a) / (p.b * q.a - q.b * p.a);
  return point_t(x, y);
}
bool onseg(point_t p, segment_t s) {
  if (dblcmp(p.x - min(s.a.x, s.b.x)) < 0 || dblcmp(p.x - max(s.a.x, s.b.
      ↪ x)) > 0) return false;
  if (dblcmp(p.y - min(s.a.y, s.b.y)) < 0 || dblcmp(p.y - max(s.a.y, s.b.
      ↪ y)) > 0) return false;
  return true;
}
bool ccut(segment_t p, segment_t q) {
  if (!ccutl(line_t(p.a, p.b), line_t(q.a, q.b))) return false;
  point_t r = cutl(line_t(p.a, p.b), line_t(q.a, q.b));
  if (!onseg(r, p) || !onseg(r, q)) return false;
  return true;
}
point_t cut(segment_t p, segment_t q) {
  return cutl(line_t(p.a, p.b), line_t(q.a, q.b));
}
struct event_t {
  double x;
  int type;
  event_t() { x = 0, type = 0; }
  event_t(double _x, int _t) : x(_x), type(_t) { }
  bool operator<(const event_t &r) const {
    return x < r.x;
  }
};
vector<segment_t> s;
double solve(const vector<segment_t> &v, const vector<int> &sl) {
  double ret = 0;
  vector<point_t> lines;
  for (int i = 0; i < v.size(); ++i) lines.push_back(point_t(v[i].k(), v[
      ↪ i].l()));
  sort(lines.begin(), lines.end());
  lines.erase(unique(lines.begin(), lines.end()), lines.end());
  for(int i = 0; i < lines.size(); ++i) {
    vector<event_t> e;
    vector<int>::const_iterator it = sl.begin();
    for(int j = 0; j < s.size(); j += *it++) {
      bool touch = false;
      for (int k = 0; k < *it; ++k) if (lines[i] == point_t(s[j + k].k(),
          ↪ s[j + k].l())) touch = true;
      if (touch) continue;
      vector<point_t> cuts;
      for (int k = 0; k < *it; ++k) {
        if (!ccutl(line_t(lines[i]), line_t(s[j + k].a, s[j + k].b)))
            ↪ continue;
        point_t r = cutl(line_t(lines[i]), line_t(s[j + k].a, s[j + k].b))
            ↪ ;
        if (onseg(r, s[j + k])) cuts.push_back(r);
      }
      sort(cuts.begin(), cuts.end());
      cuts.erase(unique(cuts.begin(), cuts.end()), cuts.end());
      if (cuts.size() == 2) {
        e.push_back(event_t(cuts[0].x, 0));
        e.push_back(event_t(cuts[1].x, 1));
      }
    }
    for (int j = 0; j < v.size(); ++j) {
      if (lines[i] == point_t(v[j].k(), v[j].l())) {
        e.push_back(event_t(min(v[j].a.x, v[j].b.x), 2));
        e.push_back(event_t(max(v[j].a.x, v[j].b.x), 3));
      }
    }
    sort(e.begin(), e.end());
    double last = e[0].x;
    int cntg = 0, cntb = 0;
    for (int j = 0; j < e.size(); ++j) {
      double y0 = lines[i].x * last + lines[i].y;
      double y1 = lines[i].x * e[j].x + lines[i].y;
      if (cntb == 0 && cntg) ret += (y0 + y1) * (e[j].x - last) / 2;
      last = e[j].x;
      if (e[j].type == 0) ++cntb;
      if (e[j].type == 1) --cntb;
      if (e[j].type == 2) ++cntg;
      if (e[j].type == 3) --cntg;
    }
  }
  return ret;
}
double polyUnion(vector<vector<point_t>> polys) {
  s.clear();
  vector<segment_t> A, B;
  vector<int> sl;
  for (int i = 0; i < polys.size(); ++i) {
    double area = 0;
    int tot = polys[i].size();
    for (int j = 0; j < tot; ++j) {
      area += cross(polys[i][j], polys[i][(j + 1) % tot]);
    }
    if (dblcmp(area) > 0) reverse(polys[i].begin(), polys[i].end());
    if (dblcmp(area) != 0) {
      sl.push_back(tot);
      for (int j = 0; j < tot; ++j) s.push_back(segment_t(polys[i][j],
          ↪ polys[i][(j + 1) % tot]));
    }
  }
  for (int i = 0; i < s.size(); ++i) {
    int sgn = dblcmp(s[i].a.x - s[i].b.x);
    if (sgn == 0) continue;
    else if (sgn < 0) A.push_back(s[i]);
    else B.push_back(s[i]);
  }
  return solve(A, sl) - solve(B, sl);
}
```

6.41 rectangleUnion.cpp

rectangleUnion.cpp 58 lines

```
/*
 Tested: MIT 2008 Team Contest 1 (Rectangles)
 Complexity: O(n log n)
*/
typedef long long ll;
struct rectangle
{
 ll xl, yl, xh, yh;
};
```

```cpp
ll rectangle_area(vector<rectangle> &rs)
{
  vector<ll> ys; // coordinate compression
  for (auto r : rs)
  {
    ys.push_back(r.yl);
    ys.push_back(r.yh);
  }
  sort(ys.begin(), ys.end());
  ys.erase(unique(ys.begin(), ys.end()), ys.end());
  int n = ys.size(); // measure tree
  vector<ll> C(8 * n), A(8 * n);
  function<void(int, int, int, int, int, int)> aux =
      [&](int a, int b, int c, int l, int r, int k)
      {
        if ((a = max(a,l)) >= (b = min(b,r))) return;
        if (a == l && b == r) C[k] += c;
        else
        {
          aux(a, b, c, l, (l+r)/2, 2*k+1);
          aux(a, b, c, (l+r)/2, r, 2*k+2);
        }
        if (C[k]) A[k] = ys[r] - ys[l];
        else A[k] = A[2*k+1] + A[2*k+2];
      };
  struct event
  {
    ll x, l, h, c;
  };
  // plane sweep
  vector<event> es;
  for (auto r : rs)
  {
    int l = lower_bound(ys.begin(), ys.end(), r.yl) - ys.begin();
    int h = lower_bound(ys.begin(), ys.end(), r.yh) - ys.begin();
    es.push_back({ r.xl, l, h, +1 });
    es.push_back({ r.xh, l, h, -1 });
  }
  sort(es.begin(), es.end(), [](event a, event b)
      {return a.x != b.x ? a.x < b.x : a.c > b.c;});
  ll area = 0, prev = 0;
  for (auto &e : es)
  {
    area += (e.x - prev) * A[0];
    prev = e.x;
    aux(e.l, e.h, e.c, 0, n, 0);
  }
  return area;
}
```

6.42 rectilinearMst.cpp

rectilinearMst.cpp 56 lines

```cpp
/*
  Tested: USACO OPEN08 (Cow Neighborhoods)
  Complexity: O(n log n)
*/
typedef long long ll;
typedef complex<ll> point;
ll rectilinear_mst(vector<point> ps)
{
  vector<int> id(ps.size());
  iota(id.begin(), id.end(), 0);
  struct edge
  {
    int src, dst;
    ll weight;
  };
  vector<edge> edges;
  for (int s = 0; s < 2; ++s)
  {
    for (int t = 0; t < 2; ++t)
    {
      sort(id.begin(), id.end(), [&](int i, int j)
      {
        return real(ps[i] - ps[j]) < imag(ps[j] - ps[i]);
      });
      map<ll, int> sweep;
      for (int i : id)
      {
        for (auto it = sweep.lower_bound(-imag(ps[i]));
            it != sweep.end(); sweep.erase(it++))
        {
          int j = it->second;
          if (imag(ps[j] - ps[i]) < real(ps[j] - ps[i]))
            break;
          ll d = abs(real(ps[i] - ps[j]))
              + abs(imag(ps[i] - ps[j]));
          edges.push_back({ i, j, d });
        }
        sweep[-imag(ps[i])] = i;
      }
      for (auto &p : ps)
        p = point(imag(p), real(p));
    }
    for (auto &p : ps)
      p = point(-real(p), imag(p));
  }
  ll cost = 0;
  sort(edges.begin(), edges.end(), [](edge a, edge b)
  {
    return a.weight < b.weight;
  });
  union_find uf(ps.size());
  for (edge e : edges)
    if (uf.join(e.src, e.dst))
      cost += e.weight;
  return cost;
}
```

6.43 semiplaneIntersection.cpp

semiplaneIntersection.cpp 54 lines

```cpp
/*
  Check wether there is a point in the intersection of
  several semi-planes. if p lies in the border of some
  semiplane it is considered to belong to the semiplane.
  Expected Running time: linear
  Tested on Triathlon [Cuban Campament Contest]
*/
bool intersect( vector<line> semiplane ){
  function<bool(line&,point&)> side = [](line &l, point &p){
    // IMPORTANT: point p belongs to semiplane defined by l
    // iff p it's clockwise respect to segment < l.p, l.q >
    // i.e. (non negative cross product)
    return cross( l.q - l.p, p - l.p ) >= 0;
  };
  function<bool(line&, line&, point&)> crosspoint = [](const line &l,
      const line &m, point &x){
    double A = cross(l.q - l.p, m.q - m.p);
    double B = cross(l.q - l.p, l.q - m.p);
    if (abs(A) < eps) return false;
    x = m.p + B / A * (m.q - m.p);
    return true;
  };
  int n = (int)semiplane.size();
  random_shuffle( semiplane.begin(), semiplane.end() );
  point cent(0, 1e9);
  for (int i = 0; i < n; ++i){
    line &S = semiplane[ i ];
    if (side(S, cent)) continue;
    point d = S.q - S.p; d /= abs( d );
    point A = S.p - d * 1e8, B = S.p + d * 1e8;
    for (int j = 0; j < i; ++j){
      point x;
      line &T = semiplane[j];
      if ( crosspoint(T, S, x) ){
        int cnt = 0;
        if (!side(T, A)){
```

```
        A = x;
        cnt++;
      }
      if (!side(T, B)){
        B = x;
        cnt++;
      }
      if (cnt == 2)
        return false;
    }
    else{
      if (!side(T, A)) return false;
    }
  }
  if (imag(B) > imag(A)) swap(A, B);
  cent = A;
 }
 return true;
}
```

6.44 sideOf.h

sideOf.h 9 lines

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

6.45 sphericalDistance.h

sphericalDistance.h 8 lines

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

6.46 triangle-circle-area-intersection.cpp

triangle-circle-area-intersection.cpp 33 lines

```
double areaTC(point_t ct, double r, point_t p1, point_t p2) { //
    ↪ intersected area
 double a, b, c, x, y, s = cross(p1 - ct, p2 - ct) / 2;
 a = dist(ct, p2), b = dist(ct, p1), c = dist(p1, p2);
 double hr2 = r * r / 2, cr2 = c * c * r * r;
 if (a ≤ r && b ≤ r) {
  return s;
 } else if (a < r && b ≥ r) {
  x = (dot(p1 - p2, ct - p2) + sqrt(cr2 - sqr(cross(p1 - p2, ct - p2)))
     ↪ ) / c;
  return asin(s * (c - x) * 2 / c / b / r) * hr2 + s * x / c;
 } else if (a ≥ r && b < r) {
  y = (dot(p2 - p1, ct - p1) + sqrt(cr2 - sqr(cross(p2 - p1, ct - p1)))
     ↪ ) / c;
  return asin(s * (c - y) * 2 / c / a / r) * hr2 + s * y / c;
 } else {
  if (fabs(2 * s) ≥ r * c || dot(p2 - p1, ct - p1) ≤ 0 || dot(p1 - p2
     ↪ , ct - p2) ≤ 0) {
   if (dot(p1 - ct, p2 - ct) < 0) {
    if (cross(p1 - ct, p2 - ct) < 0) {
     return (-pi - asin(s * 2 / a / b)) * hr2;
    } else {
     return (pi - asin(s * 2 / a / b)) * hr2;
    }
   } else {
    return asin(s * 2 / a / b) * hr2;
   }
  } else {
   x = (dot(p1 - p2, ct - p2) + sqrt(cr2 - sqr(cross(p1 - p2, ct - p2))
      ↪ )) / c;
   y = (dot(p2 - p1, ct - p1) + sqrt(cr2 - sqr(cross(p2 - p1, ct - p1))
      ↪ )) / c;
   return (asin(s * (1 - x / c) * 2 / r / b) + asin(s * (1 - y / c) * 2
      ↪  / r / a)) * hr2 + s * ((y + x) / c - 1);
  }
 }
}
double areaTC(point_t ct, double r, point_t p1, point_t p2, point_t p3)
    ↪ {
 return areaTC(ct, r, p1, p2) + areaTC(ct, r, p2, p3) + areaTC(ct, r, p3
    ↪ , p1);
}
```

BitMask (7)

7.1 SubsetSubset.cpp

Description: Computing some information f from all the subsets of some subset
Time: $\mathcal{O}(3^n)$

SubsetSubset.cpp 1027f9, 35 lines

```
#include <bits/stdc++.h>
using namespace std;
int main(){
   int N = 4;
   // If you just want to iterate over all the subsets of a mask
   for(int i=0; i<(1<<N); ++i){
      bitset<8> n(i);
      cout<<"MASK: "<<n<<endl;
      cout<<"SUBMASK:"<<endl;
      for(int j = i; j; j = (j-1) & i){
         bitset<8> p(j);
         cout<<p<<endl;
      }
      cout<<endl;
   }
   // If you want to compute some information like the sum from all the
       ↪ subsets of some mask
   for(int mask = 0; mask < (1<<N); ++mask){
    dp[mask][-1] = A[mask]; //handle base case separately (leaf states)
    for(int i = 0;i < N; ++i){
     if(mask & (1<<i))
      dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
     else
      dp[mask][i] = dp[mask][i-1];
    }
    F[mask] = dp[mask][N-1];
   }
   // Memory optimization
   for(int i = 0; i<(1<<N); ++i)
    F[i] = A[i];
   for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
    if(mask & (1<<i))
     F[mask] += F[mask^(1<<i)];
   }
   return 0;
}
```

7.2 amountOfHamiltonianWalks.cpp

Description: Finding the number of Hamiltonian walks in the unweighted and directed graph G = (V, E).
Considerations: Let dp[msk][v] be the amount of Hamiltonian walks on the subgraph generated by vertices in msk that end in the vertex v.
Time: $\mathcal{O}\left(n^2 \times n^2\right)$

amountOfHamiltonianWalks.cpp 0f2078, 25 lines

```
#define BIT(n) (1 << n)
```

```cpp
const int MAXN = 20;
int n, m, u, v, g[MAXN], dp[BIT(MAXN)][MAXN], ans;
int main(){
  cin >> n >> m;
  for (int i = 0; i < m; ++i){
    cin >> u >> v;
    g[u] |= BIT(v);
  }
  for (int i = 0; i < n; ++i)
      dp[BIT(i)][i] = 1;
  for (int msk = 1; msk < BIT(n); ++msk){
    for (int i = 0; i < n; ++i) if (msk & BIT(i)){
      int tmsk = msk ^BIT(i);
      for (int j = 0; tmsk && j < n; ++j){
        if (g[j] & BIT(i))
            dp[msk][i] += dp[tmsk][j];
      }
    }
  }
  for (int i = 0; i < n; ++i)
      ans += dp[BIT(n) - 1][i];
  cout << ans << endl;
  return 0;
}
```

7.3 existenceOfHamiltonianCycle.cpp

Description: Check for existence of Hamiltonian cycle in a directed graph.
Considerations: Let dp[msk] be the mask of the subset consisting of those vertices j such that exist a Hamiltonian walk over the subset msk beginning in vertex 0 and ending in j.
Time: $\mathcal{O}\left(n^2 \times n\right)$

existenceOfHamiltonianCycle.cpp 3c918e, 23 lines

```cpp
#define  BIT(n) (1 << n)
const int MAXN  = 20;
int n, m, u, v, g[MAXN], dp[BIT(MAXN)];
int main()
{
  cin >> n >> m;
  for (int i = 0; i < m; ++i)
  {
    cin >> u >> v;
    g[v] |= BIT(u);
  }
  dp[1] = 1;
  for (int msk = 2; msk < BIT(n); ++msk)
  {
    for (int i = 0; i < n; ++i)
    {
      if ((msk & BIT(i)) && (dp[msk ^BIT(i)] & g[i]))
        dp[msk] |= BIT(i);
    }
  }
  cout << ((dp[BIT(n) - 1] & g[0]) != 0) << endl;
  return 0;
}
```

7.4 existenceOfHamiltonianWalk.cpp

Description: Check for existence of Hamiltonian walk in the directed graph G
Considerations: Let dp[msk] be the mask of the subset consisting of those vertices v for which exist a Hamiltonian walk over the subset msk ending in v.
Time: $\mathcal{O}\left(n^2 \times n\right)$

existenceOfHamiltonianWalk.cpp b53f91, 24 lines

```cpp
#define  BIT(n) (1 << n)
const int MAXN = 20;
int n, m, u, v, g[MAXN], dp[BIT(MAXN)];
int main()
{
  cin >> n >> m;
  for (int i = 0; i < m; ++i)
  {
    cin >> u >> v;
    g[v] |= BIT(u);
  }
  for (int i = 0; i < n; ++i)
    dp[BIT(i)] = BIT(i);
  for (int msk = 1; msk < BIT(n); ++msk)
  {
    for (int i = 0; i < n; ++i)
    {
      if ((msk & BIT(i)) && (dp[msk ^BIT(i)] & g[i]))
        dp[msk] |= BIT(i);
    }
  }
  cout << (dp[BIT(n) - 1] != 0) << endl;
  return 0;
}
```

7.5 findingTheNumberOfSimplePaths.cpp

Description: Finding the number of simple paths in the directed graph G = <V, E>.
Time: $\mathcal{O}\left(n^2 \times n\right)$

findingTheNumberOfSimplePaths.cpp 87424a, 22 lines

```cpp
#define  BIT(n) (1 << n)
const int MAXN = 20;
int n, m, u, v, ans, g[MAXN], dp[BIT(MAXN)][MAXN];
int main(){
  cin >> n >> m;
  for (int i = 0; i < m; ++i){
    cin >> u >> v;
    g[u] |= BIT(v);
  }
  for (int i = 0; i < n; ++i)
    dp[BIT(i)][i] = 1;
  for (int msk = 1; msk < BIT(n); ++msk){
    for (int i = 0; i < n; ++i) if (BIT(i) & msk){
      int tmsk = msk ^BIT(i);
      for (int j = 0; tmsk && j < n; ++j) if (g[j] & BIT(i))
        dp[msk][i] += dp[tmsk][j];
      ans += dp[msk][i];
    }
  }
  cout << ans - n << endl;
  return 0;
}
```

7.6 findingTheShortestHamiltonianCycle.cpp

Description: Search for the shortest Hamiltonian cycle. Let the directed graph G = (V, E) have n vertices, and each edge have weight d(i, j). We want to find a Hamiltonian cycle for which the sum of weights of its edges is minimal.
Time: $\mathcal{O}\left(n^2 \times n^2\right)$

findingTheShortestHamiltonianCycle.cpp 4355f0, 35 lines

```cpp
#define  BIT(n) (1 << n)
using namespace std;
const int MAXN = 20,
const int INF  = 1e9+7;
int g[MAXN][MAXN];
int dp[BIT(MAXN)][MAXN];
int main(){
  int n, m, u, v, w,
  int ans = INF;
  cin>>n>>m;
  for (int i = 0; i < n; ++i){
    for (int j = 0; j < n; ++j)
      g[i][j] = INF;
  }
  for (int i = 0; i < BIT(n); ++i){
    for (int j = 0; j < n; ++j)
      dp[i][j] = INF;
  }
  for (int i = 0; i < m; ++i){
    cin >> u >> v;
    cin >> g[u][v];
  }
```

```
  dp[1][0] = 0;
  for (int msk = 2; msk < BIT(n); ++msk){
    for (int i = 0; i < n; ++i) if (msk & BIT(i)){
      int tmsk = msk ^BIT(i);
      for (int j = 0; tmsk && j < n; ++j)
        dp[msk][i] = min(dp[msk][i], dp[tmsk][j] + g[j][i]);
    }
  }
  for (int i = 1; i < n; ++i)
    ans = min(ans, dp[BIT(n) - 1][i] + g[i][0]);
  cout << ans << endl;
  return 0;
}
```

combinatorial (8)

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		

8.1 Permutations

8.1.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
n	11	12	13	14	15	16	17			
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
n	20	25	30	40	50	100	150	171		
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

8.1.2 factoradic

El sistema factorádico es un sistema numérico de raíz mixta basado en factoriales en el que el n-ésimo dígito, empezando desde la derecha, debe ser multiplicado por n!

Hay una relación natural entre los enteros 0, ..., n! − 1 (o de manera equivalente los números factorádicos con n elementos) en orden lexicográfico, cuando los enteros son expresados en forma factorádica. Esta relación ha sido llamada código Lehmer o código Lucas-Lehmer (tabla invertida). Por ejemplo, con n = 3, dicha relación es

Decimal	Factoradic	Permutation
0_{10}	$0_2 0_1 0_0$	(0,1,2)
1_{10}	$0_2 1_1 0_0$	(0,2,1)
2_{10}	$1_2 0_1 0_0$	(1,0,2)
3_{10}	$1_2 1_1 0_0$	(1,2,0)
4_{10}	$2_2 0_1 0_0$	(2,0,1)
5_{10}	$2_2 1_1 0_0$	(2,1,0)

8.2 IntPerm.cpp

Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.
Time: $\mathcal{O}(n)$

IntPerm.cpp 044568, 6 lines

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

8.3 permutation.cpp

Description: Create the factoradic representation of a permutation or recover a permutation from its factoradic representation
Time: $(n \log n)$

permutation.cpp 8349c0, 64 lines

```
#include <bits/stdc++.h>
using namespace std;
vector<int> bit;
int n;
int sum(int idx){
    int ans = 0;
    for(++idx;idx>0 ;idx-= idx&-idx)ans+=bit[idx];
    return ans;
}
void add(int idx,int val){
    for(++idx;idx<n;idx+= idx&-idx)bit[idx]+=val;
}
int bit_search(int s){
    int sum = 0;
    int pos = 0;
    for(int i = ceil(log2(n));i>=0;i--){
        if((pos+(1<<i))<n && (sum+bit[pos+(1<<i)])<s){
            sum+=bit[pos+(1<<i)];
            pos+=(1<<i);
        }
    }
    return pos;
}
int main(){
    int x;
    cin>>n;
    vector<int> factoradicA(n);
    vector<int> factoradicB(n);
    bit.resize(n);
    for(int i = 0;i<n;i++)
        add(i,1);
    for(int i = 0;i<n;i++){
        cin>>x;
        factoradicA[i] = sum(x-1);
        add(x,-1);
    }
    bit.assign(n,0);
    for(int i = 0;i<n;i++)
        add(i,1);
    for(int i = 0;i<n;i++){
        cin>>x;
        factoradicB[i] = sum(x-1);
        add(x,-1);
    }
    vector<int> final(n);
    int carry= 0;
    for(int i = n-1;i>=0;i--){
        int fact = (n-1)-i;
        final[i] = (factoradicA[i]+factoradicB[i])+carry;
        if(final[i]>=fact+1){
            final[i]-=fact+1;
            carry = 1;
        }
        else carry = 0;
    }
    for(int i = 0;i<n;i++)add(i,1);
    for(int i = 0;i<n;i++){
        x = bit_search(final[i]+1);
        cout<<x<<" ";
        add(x,-1);
    }
    cout<<endl;
    return 0;
}
```

8.3.1 General

- Principio de las casillas: al colocar n objetos en k lugares hay al menos $\lceil \frac{n}{k} \rceil$ objetos en un mismo lugar.
- Número de funciones: sean A y B conjuntos con $m = |A|$ y $n = |B|$. Sea $f : A \to B$:

- Si $m \le n$, entonces hay $m!\binom{n}{m}$ funciones inyectivas f.
- Si $m = n$, entonces hay $n!$ funciones biyectivas f.
- Si $m \ge n$, entonces hay $n!\left\{{m \atop n}\right\}$ funciones suprayectivas f.

- Barras y estrellas: ¿cuántas soluciones en los enteros no negativos tiene la ecuación $\sum_{i=1}^{k} x_i = n$? Tiene $\binom{n+k-1}{k-1}$ soluciones.
- ¿Cuántas soluciones en los enteros positivos tiene la ecuación $\sum_{i=1}^{k} x_i = n$? Tiene $\binom{n-1}{k-1}$ soluciones.
- Desordenamientos: $a_0 = 1$, $a_1 = 0$, $a_n = (n-1)(a_{n-1} + a_{n-2}) = na_{n-1} + (-1)^n$.
- Sea $f(x)$ una función. Sea $g_n(x) = xg'_{n-1}(x)$ con $g_0(x) = f(x)$. Entonces $g_n(x) = \sum_{k=0}^{n} \left\{{n \atop k}\right\} x^k f^{(k)}(x)$.
- Supongamos que tenemos $m+1$ puntos: $(0, y_0)$, $(1, y_1)$, $\ldots$, (m, y_m). Entonces el polinomio $P(x)$ de grado m que pasa por todos ellos es:

$$P(x) = \left[\prod_{i=0}^{m}(x-i)\right](-1)^m \sum_{i=0}^{m} \frac{y_i(-1)^i}{(x-i)i!(m-i)!}$$

- Sea $a_0, a_1, \ldots$ una recurrencia lineal homogénea de grado d dada por $a_n = \sum_{i=1}^{d} b_i a_{n-i}$ para $n \ge d$ con términos iniciales $a_0, a_1, \ldots, a_{d-1}$. Sean $A(x)$ y $B(x)$ las funciones generadoras de las sucesiones a_n y b_n respectivamente, entonces se cumple que $A(x) = \frac{A_0(x)}{1 - B(x)}$, donde $A_0(x) = \sum_{i=0}^{d-1}\left[a_i - \sum_{j=0}^{i-1} a_j b_{i-j}\right] x^i$.
- Si queremos obtener otra recurrencia c_n tal que $c_n = a_{kn}$, las raíces del polinomio característico de c_n se obtienen al elevar todas las raíces del polinomio característico de a_n a la k-ésima potencia; y sus términos iniciales serán $a_0, a_k, \ldots, a_{k(d-1)}$.

8.3.2 Cycles

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

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

8.3.3 The twelvefold way

¿Cuántas funciones $f\colon N \to X$ hay?

N	X	Any f	Injective	Surjective
dist.	dist.	x^n	$(x)_n$	$x!\left\{{n \atop x}\right\}$
indist.	dist.	$\binom{x+n-1}{n}$	$\binom{x}{n}$	$\binom{n-1}{n-x}$
dist.	indist.	$\left\{{n \atop 1}\right\} + \ldots + \left\{{n \atop x}\right\}$	$[n \le x]$	$\left\{{n \atop k}\right\}$
indist.	indist.	$p_1(n) + \ldots p_x(n)$	$[n \le x]$	$p_x(n)$

Where $\binom{a}{b} = \frac{1}{b!}(a)_b$ and $p_x(n)$ is the number of ways to partition the integer n using x summands.

8.3.4 Derangements

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

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

$$!n = (n-1)(!(n-1)+!(n-2)); !1 = 0, !2 = 1$$

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

8.3.5 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

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

8.4 Partitions and subsets

8.4.1 Partition function

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

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

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

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

8.4.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + \ldots + n_1 p + n_0$ and $m = m_k p^k + \ldots + m_1 p + m_0$. Then

$$\binom{m}{n} \equiv \prod_{i=0}^{k}\binom{m_i}{k_i} \pmod{p}$$

$$m = \sum_{i=0}^{k} m_i p^i \quad , \quad n = \sum_{i=0}^{k} n_i p^i$$

$$0 \le m_i, n_i < p$$

8.4.3 Binomials

8.5 multinomial.cpp

multinomial.cpp — 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;
}
```

8.6 BinomialCoeficients.cpp

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

BinomialCoeficients.cpp e85f96, 82 lines

```
long binomial_Coeff_without_MOD(int n,int r){
   long ans = 1;
   for(int i = 1;i<=min(n-k,k);i++){
      ans = (ans* (n-(i-1)))/i;
   }
   return ans;
}
/* O(n) solutions
   Based in the prof of C(n,k) = C(n-1,k-1) + C(n-1,k)
   Also calc all C(n,i) for 0<=i<=n
*/
long binomial_Coeff(int n,int m){
  int i,j;
  long bc[MAXN][MAXN];
  for (i=0; i<=n; i++) bc[i][0] = 1;
  for (j=0; j<=n; j++) bc[j][j] = 1;
  for (i=1; i<=n; i++)
     for (j=1; j<i; j++)
        bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
  return bc[n][m];
}
/*
   O(k) solution
   Only calc C(n,k)
*/
int binomial_Coeff_2(int n, int k) {
   int res = 1;
   if ( k > n - k )
      k = n - k;
   for (int i = 0; i < k; ++i){
      res *= (n - i);
      res /= (i + 1);
   }
   return res;
}
/*  Factorial modulo P */
int factmod(int n, int p) {
   int res = 1;
   while (n > 1) {
      res = (res * ((n/p) % 2 ? p-1 : 1)) % p;
      for (int i = 2; i <= n%p; ++i)
         res = (res * i) % p;
      n /= p;
   }
   return res % p;
}
/** O(1) binomial coeficient with precalc in O(n) */
const int M = 1e6;
const lli mod = 986444681;
vector<lli> fact(M+1, 1), inv(M+1, 1), invfact(M+1, 1);
lli ncr(lli n, lli r){
  if(r < 0 || r > n) return 0;
  return fact[n] * invfact[r] % mod * invfact[n - r] % mod;
}
void calc(){
   for(int i = 2; i <= M; ++i){
   fact[i] = (lli)fact[i-1] * i % mod;
   inv[i] = mod - (lli)inv[mod % i] * (mod / i) % mod;
   invfact[i] = (lli)invfact[i-1] * inv[i] % mod;
  }
}
/** Lucas Theorem: Computes C(N,R)%p in O(log(n)) if P is prime */
/** call calc() first */
lli Lucas(lli N,lli R){
  if(R<0||R>N)
   return 0;
  if(R==0||R==N)
   return 1ll;
  if(N>=mod)
   return (1ll*Lucas(N/mod,R/mod)*Lucas(N%mod,R%mod))%mod;
  return fact[n] * invfact[r] % mod * invfact[n - r] % mod;
}
/* Using calc() we can also calculate P(n,k) (permutations) */
lli permutation(int n,int k){
   return (1ll*fact[n]* invfact[n-k])%mod;
}
/** Cayley's formula: Computes all posibles trees whit n nodes */
lli cayley(int n ,int k){
   if(n-k-1<0)
      return (1ll*k*modpow(n,mod-2))%mod;
   return (1ll*k*modpow(n,n-k-1))%mod;
}
```

8.7 General purpose numbers

8.7.1 Bernoulli numbers

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

Sums of powers:

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

Euler-Maclaurin formula for infinite sums:

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

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

8.7.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

8.7.3 Eulerian numbers

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

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

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

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

8.7.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

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

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

8.7.5 Bell numbers

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

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

8.7.6 Labeled unrooted trees

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

8.7.7 Catalan numbers

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

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

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

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

8.7.8 Números de Catalán

están definidos por la recurrencia:

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

8.7.9 Números de Stirling del primer tipo

$\left[{n \atop k}\right]$ representa el número de permutaciones de n elementos en exactamente k ciclos disjuntos.

$$\left[{0 \atop 0}\right] = 1$$

$$\left[{0 \atop n}\right] = \left[{n \atop 0}\right] = 0 \qquad , \quad n > 0$$

$$\left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right] \qquad , \quad k > 0$$

$$\sum_{k=0}^{n} \left[{n \atop k}\right] = n!$$

$$\sum_{k=0}^{\infty} \left[{n \atop k}\right] x^k = \prod_{k=0}^{n-1} (x+k)$$

8.7.10 Números de Stirling del segundo tipo

$\left\{{n \atop k}\right\}$ representa el número de formas de particionar un conjunto de n objetos distinguibles en k subconjuntos no vacíos.

$$\left\{{0 \atop 0}\right\} = 1$$

$$\left\{{0 \atop n}\right\} = \left\{{n \atop 0}\right\} = 0 \qquad , \quad n > 0$$

$$\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\} \qquad , \quad k > 0$$

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

8.7.11 Números de Euler

$\left\langle{n \atop k}\right\rangle$ representa el número de permutaciones de 1 a n en donde exactamente k números son mayores que el número anterior, es decir, cuántas permutaciones tienen k "ascensos".

$$\left\langle{1 \atop 0}\right\rangle = 1$$

$$\left\langle{n \atop k}\right\rangle = (n-k)\left\langle{n-1 \atop k-1}\right\rangle + (k+1)\left\langle{n-1 \atop k}\right\rangle \qquad , \quad n \geq 2$$

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

$$\sum_{k=0}^{n-1} \left\langle{n \atop k}\right\rangle = n!$$

8.7.12 Números de Catalan

$$C_0 = 1$$

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \sum_{j=0}^{n-1} C_j C_{n-1-j}$$

$$\sum_{n=0}^{\infty} C_n x^n = \frac{1 - \sqrt{1-4x}}{2x}$$

8.7.13 Números de Bell

B_n representa el número de formas de particionar un conjunto de n elementos.

$$B_n = \sum_{k=0}^{n} \left\{{n \atop k}\right\} = \sum_{k=0}^{n-1} \binom{n-1}{k} B_k$$

$$\sum_{n=0}^{\infty} \frac{B_n}{n!} x^n = e^{e^x - 1}$$

8.7.14 Números de Bernoulli

$$B_0{}^+ = 1$$

$$B_n{}^+ = 1 - \sum_{k=0}^{n-1} \binom{n}{k} \frac{B_k{}^+}{n-k+1}$$

$$\sum_{n=0}^{\infty} \frac{B_n{}^+ x^n}{n!} = \frac{x}{1-e^{-x}} = \frac{1}{\frac{1}{1!} - \frac{x}{2!} + \frac{x^2}{3!} - \frac{x^3}{4!} + \cdots}$$

8.8 BinomialCoeficients.cpp

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

BinomialCoeficients.cpp e85f96, 82 lines

```
long binomial_Coeff_without_MOD(int n,int r){
   long ans = 1;
   for(int i = 1;i<=min(n-k,k);i++){
      ans = (ans* (n-(i-1)))/i;
   }
   return ans;
}
/* O(n) solutions
   Based in the prof of C(n,k) = C(n-1,k-1) + C(n-1,k)
   Also calc all C(n,i) for 0<=i<=n
*/
long binomial_Coeff(int n,int m){
  int i,j;
  long bc[MAXN][MAXN];
  for (i=0; i<=n; i++) bc[i][0] = 1;
  for (j=0; j<=n; j++) bc[j][j] = 1;
  for (i=1; i<=n; i++)
     for (j=1; j<i; j++)
        bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
  return bc[n][m];
}
/*
```

```
   O(k) solution
   Only calc C(n,k)
*/
int binomial_Coeff_2(int n, int k) {
   int res = 1;
   if ( k > n - k )
      k = n - k;
   for (int i = 0; i < k; ++i){
      res *= (n - i);
      res /= (i + 1);
   }
   return res;
}
/*  Factorial modulo P */
int factmod(int n, int p) {
   int res = 1;
   while (n > 1) {
      res = (res * ((n/p) % 2 ? p-1 : 1)) % p;
      for (int i = 2; i <= n%p; ++i)
         res = (res * i) % p;
      n /= p;
   }
   return res % p;
}
/** O(1) binomial coeficient with precalc in O(n) */
const int M = 1e6;
const lli mod = 986444681;
vector<lli> fact(M+1, 1), inv(M+1, 1), invfact(M+1, 1);
lli ncr(lli n, lli r){
  if(r < 0 || r > n) return 0;
  return fact[n] * invfact[r] % mod * invfact[n - r] % mod;
}
void calc(){
   for(int i = 2; i <= M; ++i){
   fact[i] = (lli)fact[i-1] * i % mod;
   inv[i] = mod - (lli)inv[mod % i] * (mod / i) % mod;
   invfact[i] = (lli)invfact[i-1] * inv[i] % mod;
  }
}
/** Lucas Theorem: Computes C(N,R)%p in O(log(n)) if P is prime */
/** call calc() first */
lli Lucas(lli N,lli R){
  if(R<0||R>N)
   return 0;
  if(R==0||R==N)
   return 1ll;
  if(N>=mod)
   return (1ll*Lucas(N/mod,R/mod)*Lucas(N%mod,R%mod))%mod;
  return fact[n] * invfact[r] % mod * invfact[n - r] % mod;
}
/* Using calc() we can also calculate P(n,k) (permutations) */
lli permutation(int n,int k){
   return (1ll*fact[n]* invfact[n-k])%mod;
}
/** Cayley's formula: Computes all posibles trees whit n nodes */
lli cayley(int n ,int k){
   if(n-k-1<0)
      return (1ll*k*modpow(n,mod-2))%mod;
   return (1ll*k*modpow(n,n-k-1))%mod;
}
```

8.9 IntPerm.cpp

Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.
Time: $\mathcal{O}(n)$

IntPerm.cpp 044568, 6 lines

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

8.10 StirlingFirst.cpp

StirlingFirst.cpp 108 lines

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1 << 18;
const int mod = 998244353;
const int root = 3;
int lim, rev[N], w[N], wn[N], inv_lim;
void reduce(int &x) { x = (x + mod) % mod; }
int POW(int x, int y, int ans = 1) {
  for (; y; y >>= 1, x = (long long) x * x % mod) if (y & 1) ans = (long
      ↪ long) ans * x % mod;
  return ans;
}
void precompute(int len) {
  lim = wn[0] = 1; int s = -1;
  while (lim < len) lim <<= 1, ++s;
  for (int i = 0; i < lim; ++i) rev[i] = rev[i >> 1] >> 1 | (i & 1) << s;
  const int g = POW(root, (mod - 1) / lim);
  inv_lim = POW(lim, mod - 2);
  for (int i = 1; i < lim; ++i) wn[i] = (long long) wn[i - 1] * g % mod;
}
void ntt(vector<int> &a, int typ) {
  for (int i = 0; i < lim; ++i) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int i = 1; i < lim; i <<= 1) {
   for (int j = 0, t = lim / i / 2; j < i; ++j) w[j] = wn[j * t];
   for (int j = 0; j < lim; j += i << 1)
    for (int k = 0; k < i; ++k) {
     const int x = a[k + j], y = (long long) a[k + j + i] * w[k] % mod;
     reduce(a[k + j] += y - mod), reduce(a[k + j + i] = x - y);
    }
  }
  if (!typ) {
   reverse(a.begin() + 1, a.begin() + lim);
   for (int i = 0; i < lim; ++i) a[i] = (long long) a[i] * inv_lim % mod
       ↪ ;
  }
}
vector<int> multiply(vector<int> &f, vector<int> &g) {
  int n=(int)f.size() + (int)g.size() - 1;
  precompute(n);
  vector<int> a = f, b = g;
  a.resize(lim); b.resize(lim);
  ntt(a, 1), ntt(b, 1);
  for (int i = 0; i < lim; ++i) a[i] = (long long) a[i] * b[i] % mod;
  ntt(a, 0);
  //while((int)a.size() && a.back() == 0) a.pop_back();
  return a;
}
int fact[N], ifact[N];
vector<int> shift(vector<int> &f, int c) { //f(x + c)
  int n=(int)f.size();
  precompute(n + n - 1);
  vector<int> a = f; a.resize(lim);
  for (int i = 0; i < n; ++i) a[i] = (long long) a[i] * fact[i] % mod;
  reverse(a.begin(), a.begin()+n);
  vector<int> b; b.resize(lim); b[0] = 1;
  for (int i = 1; i < n; ++i) b[i] = (long long) b[i - 1] * c % mod;
  for (int i = 0; i < n; ++i) b[i] = (long long) b[i] * ifact[i] % mod;
  ntt(a, 1), ntt(b, 1);
  for (int i = 0; i < lim; ++i) a[i] = (long long) a[i] * b[i] % mod;
  ntt(a, 0), reverse(a.begin(), a.begin() + n);
  vector<int> g; g.resize(n);
  for (int i = 0; i < n; ++i) g[i] = (long long) a[i] * ifact[i] % mod;
  return g;
}
// (x+1)*(x+2)*(x+3) ... (x+n)
// O(n log n) only for ntt friendly primes
// otherwise use divide and conquer in O(n log^2 n)
vector<int> range_mul(int n) {
  if (n == 0) return vector<int>({1});
  if (n & 1) {
```

```
  vector<int> f = range_mul(n - 1);
  f.push_back(0);
  for (int i = (int)f.size()-1; i; --i) f[i] = (f[i - 1] + (long long)
      ↪ n * f[i]) % mod;
  f[0] = (long long) f[0] * n % mod;
  return f;
 }
 else {
  int n_ = n >> 1;
  vector<int> f = range_mul(n_);
  vector<int> tmp = shift(f, n_);
  f.resize(n_ + 1);
  tmp.resize(n_ + 1);
  return multiply(f, tmp);
 }
}
// returns stirling1st(n, i) for 0 <= i <= n
vector<int> stirling(int n) {
 if (n == 0) return {1};
 vector<int> ans = range_mul(n - 1);
 ans.resize(n + 1);
 for (int i = n - 1; i >= 0; i--) {
  ans[i + 1] = ans[i];
 }
 ans[0] = 0;
 return ans;
}
int32_t main() {
 ios_base::sync_with_stdio(0);
 cin.tie(0);
 fact[0] = 1;
 for (int i = 1; i < N; ++i) fact[i] = (long long) fact[i - 1] * i % mod
     ↪ ;
 ifact[N - 1] = POW(fact[N - 1], mod - 2);
 for (int i = N - 1; i; --i) ifact[i - 1] = (long long) ifact[i] * i %
     ↪ mod;
 int n; cin >> n;
 auto ans = stirling(n);
 for (int i = 0; i <= n; i++) {
  cout << ans[i] << ' ';
 }
 return 0;
}
```

8.11 multinomial.cpp

multinomial.cpp 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;
}
```

8.12 permutation.cpp

Description: Create the factoradic representation of a permutation or recover a permutation from its factoradic representation
Time: $(n \log n)$

permutation.cpp 8349c0, 64 lines

```
#include <bits/stdc++.h>
using namespace std;
vector<int> bit;
int n;
int sum(int idx){
    int ans = 0;
    for(++idx;idx>0 ;idx-= idx&-idx)ans+=bit[idx];
    return ans;
}
void add(int idx,int val){
    for(++idx;idx<n;idx+= idx&-idx)bit[idx]+=val;
}
int bit_search(int s){
    int sum = 0;
    int pos = 0;
    for(int i = ceil(log2(n));i>=0;i--){
        if((pos+(1<<i))<n && (sum+bit[pos+(1<<i)])<s){
            sum+=bit[pos+(1<<i)];
            pos+=(1<<i);
        }
    }
    return pos;
}
int main(){
    int x;
    cin>>n;
    vector<int> factoradicA(n);
    vector<int> factoradicB(n);
    bit.resize(n);
    for(int i = 0;i<n;i++)
        add(i,1);
    for(int i = 0;i<n;i++){
        cin>>x;
        factoradicA[i] = sum(x-1);
        add(x,-1);
    }
    bit.assign(n,0);
    for(int i = 0;i<n;i++)
        add(i,1);
    for(int i = 0;i<n;i++){
        cin>>x;
        factoradicB[i] = sum(x-1);
        add(x,-1);
    }
    vector<int> final(n);
    int carry= 0;
    for(int i = n-1;i>=0;i--){
        int fact = (n-1)-i;
        final[i] = (factoradicA[i]+factoradicB[i])+carry;
        if(final[i]>=fact+1){
            final[i]-=fact+1;
            carry = 1;
        }
        else carry = 0;
    }
    for(int i = 0;i<n;i++)add(i,1);
    for(int i = 0;i<n;i++){
        x = bit_search(final[i]+1);
        cout<<x<<" ";
        add(x,-1);
    }
    cout<<endl;
    return 0;
}
```

numerical (9)

9.1 BasisXor.cpp

Description: Structure to form a basis in Z2 that alows compute things around xor because xor is a sum in Z2 like the maxxor possible , minimum , number of diferent xor´s, kth possible xor
Time: $\mathcal{O}(\log N)$

BasisXor.cpp 396773, 117 lines

```
struct Basis {
    vector<int> a;
    void insert(int x) {
        for (auto &i: a) x = min(x, x ^i);
        if (!x) return;
        for (auto &i: a) if ((i ^x) < i) i ^= x;
        a.push_back(x);
        sort(a.begin(), a.end());
    }
    bool can(int x) {
        for (auto &i: a) x = min(x, x ^i);
        return !x;
    }
    int maxxor(int x = 0) {
        for (auto &i: a) x = max(x, x ^i);
```

```
        return x;
    }
    int minxor(int x = 0) {
        for (auto &i: a) x = min(x, x ^i);
        return x;
    }
    int kth(int k) { // 1st is 0
        int sz = (int)a.size();
        if (k > (1LL << sz)) return -1;
        k--; int ans = 0;
        for (int i = 0; i < sz; i++) if (k >> i & 1) ans ^= a[i];
        return ans;
    }
}t;
// Arbitrary size
const int sz = 500;
struct basisxor{
    bitset<sz> bs[sz];
    bitset<sz> index[sz];
    bitset<sz> zero;
    basisxor(){
        for(int i=0;i<sz;i++) bs[i]=zero;
    }
    bitset<sz> chk(bitset<sz> &b){
        bitset<sz> ans;
        int i;
        for(i = sz-1;i>=0;i--){
            if(b[i]== 0)continue;
            if(b.count()== 0)break;
            if(bs[i].count()==0)break;
            b^=bs[i];
            ans^=index[i];
        }
        if(b.count()==0) return ans;
        return zero;
    }
    bool add(bitset<sz> &b,int idx){
        int i;
        bitset<sz> x;
        x[idx]= 1;
        for(i = sz-1;i>=0;i--){
            if(b[i]== 0)continue;
            if(b.count()== 0)break;
            if(bs[i].count()==0)break;
            b^=bs[i];
            x^=index[i];
        }
        if(i ==-1)return false;
        bs[i] = b;
        index[i] = x;
        return true;
    }
    void print(){
        for(int i =sz-1;i>=0;i--){
            cout<<"I: "<<i<<" "<<bs[i]<<" idx: "<<endl;
        }
    }
};
int bin_pow(int a,int b){
    int x = 1;
    while(b){
        if(b&1) x*=a;
        a*=a;
        b>>=1;
    }
    return x;
}
// another
struct basisxor{
    int base[32];
    int sz = 0;
    basisxor(){
        for(int i = 0;i<32;i++)
            base[i] =0;
    }
    void add(int x){
        while(x != 0 && base[31-__builtin_clz(x)]!= 0){
            x^= base[31-__builtin_clz(x)];
        }
        if(!x)return;
        base[31-__builtin_clz(x)] = x;
        sz++;
    }
    int kth(int k){
        int total = bin_pow(2,sz);
        int val = 0;
        for(int i = 31;i>=0;i--){
            if(base[i] == 0)continue;
            if(k<total/2){
                if((val>>i)&1)
                    val^=base[i];
            }
            else{
                if(!((val>>i)&1))
                    val^=base[i];
                k-=total/2;
            }
            total>>=1;
        }
        return val;
    }
};
```

9.2 BerlekampMassey.cpp

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

BerlekampMassey.cpp cbc8b5, 20 lines

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

9.3 Determinant.cpp

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

Determinant.cpp bd5cec, 15 lines

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

```
    if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
   }
  }
  return res;
}
```

9.4 ExtendedPolynomial.cpp

ExtendedPolynomial.cpp 229 lines

```
const int N = 3e5 + 9, mod = 998244353;
struct base {
  double x, y;
  base() { x = y = 0; }
  base(double x, double y): x(x), y(y) { }
};
inline base operator + (base a, base b) { return base(a.x + b.x, a.y + b.y); }
inline base operator - (base a, base b) { return base(a.x - b.x, a.y - b.y); }
inline base operator * (base a, base b) { return base(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
inline base conj(base a) { return base(a.x, -a.y); }
int lim = 1;
vector<base> roots = {{0, 0}, {1, 0}};
vector<int> rev = {0, 1};
const double PI = acosl(- 1.0);
void ensure_base(int p) {
  if(p <= lim) return;
  rev.resize(1 << p);
  for(int i = 0; i < (1 << p); i++) rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (p - 1));
  roots.resize(1 << p);
  while(lim < p) {
    double angle = 2 * PI / (1 << (lim + 1));
    for(int i = 1 << (lim - 1); i < (1 << lim); i++) {
      roots[i << 1] = roots[i];
      double angle_i = angle * (2 * i + 1 - (1 << lim));
      roots[(i << 1) + 1] = base(cos(angle_i), sin(angle_i));
    }
    lim++;
  }
}
void fft(vector<base> &a, int n = -1) {
  if(n == -1) n = a.size();
  assert((n & (n - 1)) == 0);
  int zeros = __builtin_ctz(n);
  ensure_base(zeros);
  int shift = lim - zeros;
  for(int i = 0; i < n; i++) if(i < (rev[i] >> shift)) swap(a[i], a[rev[i] >> shift]);
  for(int k = 1; k < n; k <<= 1) {
    for(int i = 0; i < n; i += 2 * k) {
      for(int j = 0; j < k; j++) {
        base z = a[i + j + k] * roots[j + k];
        a[i + j + k] = a[i + j] - z;
        a[i + j] = a[i + j] + z;
      }
    }
  }
}
//eq = 0: 4 FFTs in total
//eq = 1: 3 FFTs in total
vector<int> multiply(vector<int> &a, vector<int> &b, int eq = 0) {
  int need = a.size() + b.size() - 1;
  int p = 0;
  while((1 << p) < need) p++;
  ensure_base(p);
  int sz = 1 << p;
  vector<base> A, B;
  if(sz > (int)A.size()) A.resize(sz);
  for(int i = 0; i < (int)a.size(); i++) {
    int x = (a[i] % mod + mod) % mod;
    A[i] = base(x & ((1 << 15) - 1), x >> 15);
  }
  fill(A.begin() + a.size(), A.begin() + sz, base{0, 0});
  fft(A, sz);
  if(sz > (int)B.size()) B.resize(sz);
  if(eq) copy(A.begin(), A.begin() + sz, B.begin());
  else {
    for(int i = 0; i < (int)b.size(); i++) {
      int x = (b[i] % mod + mod) % mod;
      B[i] = base(x & ((1 << 15) - 1), x >> 15);
    }
    fill(B.begin() + b.size(), B.begin() + sz, base{0, 0});
    fft(B, sz);
  }
  double ratio = 0.25 / sz;
  base r2(0, - 1), r3(ratio, 0), r4(0, - ratio), r5(0, 1);
  for(int i = 0; i <= (sz >> 1); i++) {
    int j = (sz - i) & (sz - 1);
    base a1 = (A[i] + conj(A[j])), a2 = (A[i] - conj(A[j])) * r2;
    base b1 = (B[i] + conj(B[j])) * r3, b2 = (B[i] - conj(B[j])) * r4;
    if(i != j) {
      base c1 = (A[j] + conj(A[i])), c2 = (A[j] - conj(A[i])) * r2;
      base d1 = (B[j] + conj(B[i])) * r3, d2 = (B[j] - conj(B[i])) * r4;
      A[i] = c1 * d1 + c2 * d2 * r5;
      B[i] = c1 * d2 + c2 * d1;
    }
    A[j] = a1 * b1 + a2 * b2 * r5;
    B[j] = a1 * b2 + a2 * b1;
  }
  fft(A, sz); fft(B, sz);
  vector<int> res(need);
  for(int i = 0; i < need; i++) {
    long long aa = A[i].x + 0.5;
    long long bb = B[i].x + 0.5;
    long long cc = A[i].y + 0.5;
    res[i] = (aa + ((bb % mod) << 15) + ((cc % mod) << 30))%mod;
  }
  return res;
}
template <int32_t MOD>
struct modint {
  int32_t value;
  modint() = default;
  modint(int32_t value_) : value(value_) {}
  inline modint<MOD> operator + (modint<MOD> other) const { int32_t c = this->value + other.value; return modint<MOD>(c >= MOD ? c - MOD : c); }
  inline modint<MOD> operator - (modint<MOD> other) const { int32_t c = this->value - other.value; return modint<MOD>(c < 0 ? c + MOD : c); }
  inline modint<MOD> operator * (modint<MOD> other) const { int32_t c = (int64_t)this->value * other.value % MOD; return modint<MOD>(c < 0 ? c + MOD : c); }
  inline modint<MOD> & operator += (modint<MOD> other) { this->value += other.value; if (this->value >= MOD) this->value -= MOD; return *this; }
  inline modint<MOD> & operator -= (modint<MOD> other) { this->value -= other.value; if (this->value < 0) this->value += MOD; return *this; }
  inline modint<MOD> & operator *= (modint<MOD> other) { this->value = (int64_t)this->value * other.value % MOD; if (this->value < 0) this->value += MOD; return *this; }
  inline modint<MOD> operator - () const { return modint<MOD>(this->value ? MOD - this->value : 0); }
  modint<MOD> pow(uint64_t k) const {
    modint<MOD> x = *this, y = 1;
    for (; k; k >>= 1) {
      if (k & 1) y *= x;
      x *= x;
    }
    return y;
  }
  modint<MOD> inv() const { return pow(MOD - 2); } // MOD must be a prime
```

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

9.5 FWHT.cpp

FWHT.cpp 71 lines

```
const int N = 3e5 + 9, mod = 1e9 + 7;
int POW(long long n, long long k) {
  int ans = 1 % mod; n %= mod; if (n < 0) n += mod;
  while (k) {
    if (k & 1) ans = (long long) ans * n % mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}
const int inv2 = (mod + 1) >> 1;
#define M (1 << 20)
#define OR 0
#define AND 1
#define XOR 2
struct FWHT{
  int P1[M], P2[M];
  void wt(int *a, int n, int flag = XOR) {
```

```cpp
    if (n == 0) return;
    int m = n / 2;
    wt(a, m, flag); wt(a + m, m, flag);
    for (int i = 0; i < m; i++){
      int x = a[i], y = a[i + m];
      if (flag == OR) a[i] = x, a[i + m] = (x + y) % mod;
      if (flag == AND) a[i] = (x + y) % mod, a[i + m] = y;
      if (flag == XOR) a[i] = (x + y) % mod, a[i + m] = (x - y + mod) % mod;
    }
  }
  void iwt(int* a, int n, int flag = XOR) {
    if (n == 0) return;
    int m = n / 2;
    iwt(a, m, flag); iwt(a + m, m, flag);
    for (int i = 0; i < m; i++){
      int x = a[i], y = a[i + m];
      if (flag == OR) a[i] = x, a[i + m] = (y - x + mod) % mod;
      if (flag == AND) a[i] = (x - y + mod) % mod, a[i + m] = y;
      if (flag == XOR) a[i] = 1LL * (x + y) * inv2 % mod, a[i + m] = 1LL * (x - y + mod) * inv2 % mod; // replace inv2 by >>1 if not required
    }
  }
  vector<int> multiply(int n, vector<int> A, vector<int> B, int flag = XOR) {
    assert(__builtin_popcount(n) == 1);
    A.resize(n); B.resize(n);
    for (int i = 0; i < n; i++) P1[i] = A[i];
    for (int i = 0; i < n; i++) P2[i] = B[i];
    wt(P1, n, flag); wt(P2, n, flag);
    for (int i = 0; i < n; i++) P1[i] = 1LL * P1[i] * P2[i] % mod;
    iwt(P1, n, flag);
    return vector<int> (P1, P1 + n);
  }
  vector<int> pow(int n, vector<int> A, long long k, int flag = XOR) {
    assert(__builtin_popcount(n) == 1);
    A.resize(n);
    for (int i = 0; i < n; i++) P1[i] = A[i];
    wt(P1, n, flag);
    for(int i = 0; i < n; i++) P1[i] = POW(P1[i], k);
    iwt(P1, n, flag);
    return vector<int> (P1, P1 + n);
  }
}t;
int32_t main() {
  int n; cin >> n;
  vector<int> a(M, 0);
  for(int i = 0; i < n; i++) {
    int k; cin >> k; a[k]++;
  }
  vector<int> v = t.pow(M, a, n, AND);
  int ans = 1;
  for(int i = 1; i < M; i++) ans += v[i] > 0;
  cout << ans << '\n';;
  return 0;
}
```

9.6 FastFourierTransform.h

Description: fft(a) computes $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all k. N must be a power of 2. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2)\log_2 N < 9 \cdot 10^{14}$ (in practice 10^{16}; higher for random inputs). Otherwise, use NTT/FFTMod.
Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ ($\sim$1s for $N = 2^{22}$)

FastFourierTransform.h 9db875, 163 lines

```cpp
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vector<complex<long double>> R(2, 1);
  static vector<C> rt(2, 1); // (^ 10% faster if double)
  for (static int k = 2; k < n; k *= 2) {
    R.resize(n); rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
      C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)
      a[i + j + k] = a[i + j] - z;
      a[i + j] += z;
    }
}
vd conv(const vd& a, const vd& b) {
  if (a.empty() || b.empty()) return {};
  vd res(sz(a) + sz(b) - 1);
  int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
  vector<C> in(n), out(n);
  copy(all(a), begin(in));
  rep(i,0,sz(b)) in[i].imag(b[i]);
  fft(in);
  for (C& x : in) x *= x;
  rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
  fft(out);
  rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
  return res;
}
//
// Get the sums of products of all subsets of size k
#include <bits/stdc++.h>
using namespace std;
const double PI = 4*atan(1);
typedef complex<double> base;
vector<base> FFT;
long long FFT_N;
#define endl '\n'
#define __ ios_base::sync_with_stdio(false);cin.tie(NULL);
void init_fft(long long n){
    FFT_N = n;
    FFT.resize(n);
    double angle = 2 * PI / n;
    for(int i = 0; i < n; i++)
    FFT[i] = base( cos(i * angle), sin(i * angle));
}
void fft (vector<base> & a){
    long long n = (long long) a.size();
    if (n == 1) return;
    long long half = n >> 1;
    vector<base> even (half), odd (half);
    for (int i=0, j=0; i<n; i+=2, ++j){
        even[j] = a[i];
        odd[j] = a[i+1];
    }
    fft (even), fft (odd);
    for (int i=0, fact = FFT_N/n; i < half; ++i){
        base twiddle = odd[i] * FFT[i * fact] ;
        a[i] = even[i] + twiddle;
        a[i+half] = even[i] - twiddle;
    }
}
void multiply (const vector<double> & a, const vector<double> & b, vector<double> & res){
    vector<base> fa (a.begin(), a.end()), fb (b.begin(), b.end());
    long long n = 1;
    while (n < 2*max (a.size(), b.size())) n <<= 1;
    fa.resize (n), fb.resize (n);
    init_fft(n);
    fft (fa), fft (fb);
    for (size_t i=0; i<n; ++i)
        fa[i] = conj( fa[i] * fb[i]);
    fft (fa);
    res.resize (n);
    for (size_t i=0; i<n; ++i){
```

```
    res[i] = (fa[i].real() / n );
  }
}
vector<double> getProb(double x,vector<int> &B,double d,double p){
  vector<double> P;
  for(int i = 0;i<B.size();i++){
    int b = B[i];
    double z = min(1.0,x/b);
    double y = min(1.0,d*(x/b));
    P.push_back((p*y)+((1-p)*z));
  }
  return P;
}
vector<double> getSums(vector<double> Probs){
  vector<double> inp,res;
  vector<vector<double>> calc[20];
  int n = Probs.size();
  for(int i=0;i<n;i++){
    inp.clear();
    inp.push_back(1);
    inp.push_back(Probs[i]);
    calc[0].push_back(inp);
  }
  int p = 0,l = n;
  while(l>1){
    p++;
    for(int i=0;i<l/2;i++){
      calc[p].push_back(res);
      multiply(calc[p-1][2*i],calc[p-1][2*i+1],calc[p][i]);
    }
    if(l%2)
      calc[p].push_back(calc[p-1][l-1]);
    l=calc[p].size();
  }
  return calc[p][0];
}
bool isEqual(double a,double b){
  return fabs(a-b)<1e-8;
}
int main(){__
  int n;
  double x,P,d,e;
  cin>>n;
  cin>>d>>P>>e;
  vector<int> B(n);
  for(auto &c:B)cin>>c;
  double l = 0,r = 10;
  int cont = 60;
  cout<<fixed<<setprecision(6);
  while(cont--){
    double m = (l+r)/2;
    auto Probs = getProb(m,B,d,P);
    int contOnes = 0;
    vector<double> P2;
    double prod = 1;
    double prodI = 1;
    for(auto &c:Probs){
      if(isEqual(c,1.0))
        contOnes++;
      else {
        P2.push_back(c/(1.0-c));
        prodI*=(1.0-c);
        prod*=c;
      }
    }
    double E = 0;
    if(contOnes== n){
      E = n;
    }
    else{
      auto sums = getSums(P2);
      for(int i = contOnes;i<n;i++){
        int need = i-contOnes;
        E+=i*(prodI*sums[need]);
      }
      E+=n*prod;
    }
    if(E>=e)r = m;
    else l = m;
  }
  cout<<r<<endl;
  return 0;
}
```

9.7 FastFourierTransformMod.h

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

FastFourierTransformMod.h — 26e1c6, 23 lines

```
#include "FastFourierTransform.h"
typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  vl res(sz(a) + sz(b) - 1);
  int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
  vector<C> L(n), R(n), outs(n), outl(n);
  rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
  rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
  fft(L), fft(R);
  rep(i,0,n) {
    int j = -i & (n - 1);
    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
  }
  fft(outl), fft(outs);
  rep(i,0,sz(res)) {
    ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
    ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
  }
  return res;
}
```

9.8 FastSubsetTransform.h

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

FastSubsetTransform.h — 464cf3, 16 lines

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

9.9 GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is *eps*. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.
Usage: `double func(double x) { return 4+x+.3*x*x; }`
`double xmin = gss(-1000,1000,func);`
Time: $\mathcal{O}\left(\log((b-a)/\epsilon)\right)$

GoldenSectionSearch.h — 31d45b, 14 lines

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

9.10 HillClimbing.h

HillClimbing.h — 13 lines

```
typedef array<double, 2> P;
template<class F> pair<double, P> hillClimb(P start, F f) {
  pair<double, P> cur(f(start), start);
  for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
    rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) {
      P p = cur.second;
      p[0] += dx*jmp;
      p[1] += dy*jmp;
      cur = min(cur, make_pair(f(p), p));
    }
  }
  return cur;
}
```

9.11 IntDeterminant.h

Description: Calculates determinant using modular arithmetics. Modulos can also be removed to get a pure-integer version.

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

IntDeterminant.h — 3313dc, 18 lines

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

9.12 Integrate.h

Integrate.h — 7 lines

```
template<class F>
double quad(double a, double b, F f, const int n = 1000) {
  double h = (b - a) / 2 / n, v = f(a) + f(b);
  rep(i,1,n*2)
    v += f(a + i*h) * (i&1 ? 4 : 2);
  return v * h / 3;
}
```

9.13 IntegrateAdaptive.h

IntegrateAdaptive.h — 14 lines

```
typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6
template <class F>
d rec(F& f, d a, d b, d eps, d S) {
  d c = (a + b) / 2;
  d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
  if (abs(T - S) <= 15 * eps || b - a < 1e-10)
    return T + (T - S) / 15;
  return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}
template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
  return rec(f, a, b, eps, S(a, b));
}
```

9.14 LinearRecurrence.h

Description: Generates the k'th term of an n-order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0 \ldots \geq n-1]$ and $tr[0 \ldots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.

Usage: `linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number`

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

LinearRecurrence.h — f4e444, 22 lines

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

9.15 MatrixInverse-mod.h

Description: Invert matrix A modulo a prime. Returns rank; result is stored in A unless singular (rank < n). For prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.

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

MatrixInverse-mod.h — c63129, 34 lines

```
#include "../number-theory/ModPow.h"
int matInv(vector<vector<ll>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<ll>> tmp(n, vector<ll>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;
  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n) if (A[j][k]) {
      r = j; c = k; goto found;
    }
    return i;
found:
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n) swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    ll v = modpow(A[i][i], mod - 2);
    rep(j,i+1,n) {
      ll f = A[j][i] * v % mod;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] = (A[j][k] - f*A[i][k]) % mod;
      rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) % mod;
```

```
  }
  rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
  rep(j,0,n) tmp[i][j] = tmp[i][j] * v % mod;
  A[i][i] = 1;
 }
 for (int i = n-1; i > 0; --i) rep(j,0,i) {
  ll v = A[j][i];
  rep(k,0,n) tmp[j][k] = (tmp[j][k] - v*tmp[i][k]) % mod;
 }
 rep(i,0,n) rep(j,0,n)
  A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] < 0 ? mod : 0);
 return n;
}
```

9.16 MatrixInverse.h

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

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

MatrixInverse.h — ebfff6, 32 lines

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

9.17 MatrixTemplate.cpp

MatrixTemplate.cpp — 379 lines

```
#define mod 1e9+7
#define INF INT_MAX
const double EPS = 1e-9;
typedef long long int lli;
template <typename T>
struct Matrix {
   vector < vector <T> > A;
   int r,c;
   Matrix(){
      this->r = 0;
      this->c = 0;
   }
   Matrix(int r,int c){
      this->r = r;
      this->c = c;
      A.assign(r , vector <T> (c));
   }
   Matrix(int r,int c,const T &val){
      this->r = r;
      this->c = c;
      A.assign(r , vector <T> (c , val));
   }
   Matrix(int n){
      this->r = this->c = n;
      A.assign(n , vector <T> (n));
      for(int i=0;i<n;i++)
         A[i][i] = (T)1;
   }
   Matrix operator * (const Matrix<T> &B){
//    Matrix <T> C(r,B.c,0);
//    for(int i=0 ; i<r ; i++)
//       for(int j=0 ; j<B.c ; j++)
//          for(int k=0 ; k<c ; k++)
//             C[i][j] = (C[i][j] + ( (long long )A[i][k] * (long long)B[k][j] ));
//    return C;
      Matrix<T> C(r,B.c,0);
      for(int i = 0;i<r;i++){
         for(int j = 0;j<B.c;j++){
            for(int k = 0;k<c;k++){
               C[i][j] = (C[i][j] + ((lli)A[i][k] * (lli)B[k][j]));
                  if(C[i][j] >= 8ll*mod*mod)
                     C[i][j]%=mod;
               }
            }
         }
      for(int i = 0;i<r;i++)for(int j = 0;j<c;j++)C[i][j]%=mod;
      return C;
   }
   Matrix operator + (const Matrix<T> &B){
      assert(r == B.r);
      assert(c == B.c);
      Matrix <T> C(r,c,0);
      int i,j;
      for(i=0;i<r;i++)
         for(j=0;j<c;j++)
            C[i][j] = ((A[i][j] + B[i][j]));
      return C;
   }
   Matrix operator*(int & c) {
      Matrix<T> C(r, c);
      for(int i = 0; i < r; i++)
         for(int j = 0; j < c; j++)
            C[i][j] = A[i][j] * c;
      return C;
   }
   Matrix operator - (){
      Matrix <T> C(r,c,0);
      int i,j;
      for(i=0;i<r;i++)
         for(j=0;j<c;j++)
            C[i][j] = -A[i][j];
      return C;
   }
   Matrix operator - (const Matrix<T> &B){
      assert(r == B.r);
      assert(c == B.c);
      Matrix <T> C(r,c,0);
      int i,j;
      for(i=0;i<r;i++)
         for(j=0;j<c;j++)
            C[i][j] = A[i][j] - B[i][j];
      return C;
   }
   Matrix operator ^(long long n){
      assert(r == c);
      int i,j;
      Matrix <T> C(r);
      Matrix <T> X(r,c,0);
      for(i=0;i<r;i++)
```

```cpp
        for(j=0;j<c;j++)
            X[i][j] = A[i][j];
    while(n){
        if(n&1)
            C *= X;
        X *= X;
        n /= 2;
    }
    return C;
}
vector<T>& operator [] (int i){
    assert(i < r);
    assert(i >= 0);
    return A[i];
}
const vector<T>& operator [] (int i) const{
    assert(i < r);
    assert(i >= 0);
    return A[i];
}
friend ostream& operator << (ostream &out,const Matrix<T> &M){
    for (int i = 0; i < M.r; ++i) {
        for (int j = 0; j < M.c; ++j) {
            out << M[i][j] << " ";
        }
        out << '\n';
    }
    return out;
}
void operator *= (const Matrix<T> &B){
    (*this) = (*this)*B;
}
void operator += (const Matrix<T> &B){
    (*this) = (*this)+B;
}
void operator -= (const Matrix<T> &B){
    (*this) = (*this)-B;
}
void operator ^= (long long n){
    (*this) =(*this)^n;
}
//Inverse
bool Inverse(Matrix<double> &inverse){
    if(this->detGauss() == 0)return false;
    int n = A[0].size();
    Matrix<double> temp(n,2*n);
    for(int i = 0;i<n;i++)
        for(int j = 0;j<n;j++)temp[i][j] = A[i][j];
    Matrix<double> ident(n);
    for(int i = 0;i<n;i++)
        for(int j = n;j<2*n;j++)temp[i][j] = ident[i][j-n];
    int m = n*2;
    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (temp[i][col]) > abs (temp[sel][col]))
                sel = i;
        if (abs (temp[sel][col]) < EPS)
            continue;
        for (int i=col; i<m; ++i)
            swap (temp[sel][i], temp[row][i]);
        where[col] = row;
        double div = temp[row][col];
        for(int i = 0;i<m;i++)
            if(fabs(temp[row][i])>EPS)temp[row][i] /=div;
        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = temp[i][col] / temp[row][col];
                for (int j=col; j<m; ++j)
                    temp[i][j] -= temp[row][j] * c;
            }
        ++row;
    }
    for(int i = 0;i<n;i++)
        for(int j = 0;j<n;j++)
            inverse[i][j] = temp[i][j+n];
    return true;
}
//Adjoint
Matrix<T> minor(int x, int y){
    Matrix<T> M(r-1, c-1);
    for(int i = 0; i < c-1; ++i)
        for(int j = 0; j < r-1; ++j)
            M[i][j] = A[i < x ? i : i+1][j < y ? j : j+1];
    return M;
}
T cofactor(int x, int y){
    T ans = minor(x, y).detGauss();
    if((x + y) % 2 == 1) ans *= -1;
    return ans;
}
Matrix<T> cofactorMatrix(){
    Matrix<T> C(r, c);
    for(int i = 0; i < c; i++)
        for(int j = 0; j < r; j++)
            C[i][j] = cofactor(i, j);
    return C;
}
Matrix<T> Adjunta(){
    int n = A[0].size();
    Matrix<int> adjoint(n);
    Matrix<double> inverse(n);
    this->Inverse(inverse);
    int determinante = this->detGauss();
    if(determinante){
        for(int i = 0;i<n;i++)
            for(int j = 0;j<n;j++)
                adjoint[i][j] = (T)round((inverse[i][j]*determinante));
    }
    else {
        adjoint = this->cofactorMatrix().transpose();
    }
    return adjoint;
}
//Transpuesta
Matrix transpose(){
    Matrix <T> C(c,r);
    int i,j;
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            C[j][i] = A[i][j];
    return C;
}
//Traza
T trace(){
    T sum = 0;
    for(int i = 0; i < min(r, c); i++)
        sum += A[i][i];
    return sum;
}
//Determinante
int determinant() {
    int n = r;
    Matrix<T> temp(n);
    temp.A = A;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            temp[i][j] %= mod;
    lli res = 1;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            for (; temp[j][i]; res = -res) {
                long long t = temp[i][i] / temp[j][i];
                for (int k = i; k < n; k++) {
                    temp[i][k] = (temp[i][k] - temp[j][k] * t) % mod;
                    std::swap(temp[j][k], temp[i][k]);
```

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

9.18 NumberTheoreticTransform.h

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

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

NumberTheoreticTransform.h f699dc, 61 lines

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

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

PolyInterpolate.h 08bf48, 13 lines

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

9.20 PolyRoots.h

Description: Finds the real roots to a polynomial.

Usage: `polyRoots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0`

Time: $\mathcal{O}\left(n^2 \log(1/\epsilon)\right)$

PolyRoots.h - "Polynomial.h" b00bfe, 23 lines

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

9.21 Polynomial.h

Polynomial.h 17 lines

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

9.22 PolynomialInt.cpp

PolynomialInt.cpp 488 lines

```
//
// Polynomial with integer coefficient (mod M)
//
// Implemented routines:
//  1) addition
//  2) subtraction
//  3) multiplication (naive O(n^2), Karatsuba O(n^1.5..), FFT O(n log n))
//  4) division (naive O(n^2), Newton O(M(n)))
//  5) gcd
//  6) multipoint evaluation (divide conquer: O(M(n) log |X|))
//  7) interpolation (naive O(n^2), divide conquer O(M(n) log n))
//  8) polynomial shift (naive, fast)
//
//  *) n! mod M in O(n^{1/2} log n) time
//
//
typedef long long ll;
ll add(ll a, ll b, ll M)
{
 return (a += b) >= M ? a - M : a;
}
ll sub(ll a, ll b, ll M)
{
 return (a -= b) < 0 ? a + M : a;
}
ll mul(ll a, ll b, ll M)
{
 ll q = (long double) a * (long double) b / (long double) M;
 ll r = a * b - q * M;
 return (r + 5 * M) % M;
}
// solve b x == a (mod M)
ll div(ll a, ll b, ll M)
{
 ll u = 1, x = 0, s = b, t = M;
 while (s)
 {
  ll q = t / s;
  swap(x -= u * q, u);
  swap(t -= s * q, s);
 }
 if (a % t)
  return -1; // infeasible
 return mul(x < 0 ? x + M : x, a / t, M); // b (xa/t) == a (mod M)
}
ll pow(ll a, ll b, ll M)
{
 ll x = 1;
 for (; b > 0; b >>= 1)
 {
  if (b & 1)
   x = (a * x) % M;
  a = (a * a) % M;
 }
 return x;
}
// p(x) = p[0] + p[1] x + ... + p[n-1] x^n-1
// assertion: p.back() != 0
typedef vector<ll> poly;
ostream& operator<<(ostream &os, const poly &p)
{
 bool head = true;
 for (int i = 0; i < p.size(); ++i)
 {
  if (p[i] == 0)
   continue;
  if (!head)
   os << " + ";
  os << p[i];
  head = false;
  if (i >= 1)
   os << " x";
  if (i >= 2)
   os << "^" << i;
 }
 return os;
}
poly add(poly p, const poly &q, ll M)
{
 if (p.size() < q.size())
  p.resize(q.size());
 for (int i = 0; i < q.size(); ++i)
  p[i] = add(p[i], q[i], M);
 while (!p.empty() && !p.back())
  p.pop_back();
 return p;
}
poly sub(poly p, const poly &q, ll M)
{
 if (p.size() < q.size())
  p.resize(q.size());
 for (int i = 0; i < q.size(); ++i)
  p[i] = sub(p[i], q[i], M);
 while (!p.empty() && !p.back())
  p.pop_back();
 return p;
}
// naive multiplication in O(n^2)
poly mul_n(const poly &p, const poly &q, ll M)
{
 if (p.empty() || q.empty())
  return {};
 poly r(p.size() + q.size() - 1);
 for (int i = 0; i < p.size(); ++i)
  for (int j = 0; j < q.size(); ++j)
   r[i + j] = add(r[i + j], mul(p[i], q[j], M), M);
 while (!r.empty() && !r.back())
  r.pop_back();
 return r;
}
// naive division (long division) in O(n^2)
pair<poly, poly> divmod_n(poly p, poly q, ll M)
{
 poly u(p.size() - q.size() + 1);
 ll inv = div(1, q.back(), M);
 for (int i = u.size() - 1; i >= 0; --i)
 {
  u[i] = mul(p.back(), inv, M);
  for (int j = 0; j < q.size(); ++j)
   p[j + p.size() - q.size()] = sub(p[j + p.size() - q.size()],
     mul(q[j], u[i], M), M);
  p.pop_back();
 }
 return {u, p};
}
// Karatsuba multiplication; this works correctly for M in [long long]
poly mul_k(poly p, poly q, ll M)
{
 int n = max(p.size(), q.size()), m = p.size() + q.size() - 1;
 for (int k : { 1, 2, 4, 8, 16 })
  n |= (n >> k);
 ++n; // n is power of two
 p.resize(n);
 q.resize(n);
 poly r(6 * n);
 function<void(ll*, ll*, int, ll*)> rec = [&](ll *p0, ll *q0, int n, ll *r0)
 {
  if (n <= 4)
  {
   // 4 is the best threshold
   fill_n(r0, 2*n, 0);
   for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
     r0[i+j] = add(r0[i+j], mul(p0[i], q0[j], M), M);
```

```
    return;
  }
  ll *p1=p0+n/2,*q1=q0+n/2,*r1=r0+n/2,*r2=r0+n,*u=r0+5*n,*v=u+n/2,*w=r0
      ↪ +2*n;
  for (int i = 0; i < n/2; ++i)
  {
    u[i] = add(p0[i], p1[i], M);
    v[i] = add(q0[i], q1[i], M);
  }
  rec(p0, q0, n/2, r0);
  rec(p1, q1, n/2, r2);
  rec( u, v, n/2, w);
  for (int i = 0; i < n; ++i) w[i] = sub(w[i], add(r0[i], r2[i], M), M)
      ↪ ;
  for (int i = 0; i < n; ++i) r1[i] = add(r1[i], w[i], M);
 };
 rec(&p[0], &q[0], n, &r[0]);
 r.resize(m);
 return r;
}
// FFT-based multiplication: this works correctly for M in [int]
// assume: size of a/b is power of two, mod is predetermined
template<int mod, int sign>
void fmt(vector<ll>& x)
{
 const int n = x.size();
 int h = pow(3, (mod - 1) / n, mod);
 if (sign < 0)
  h = div(1, h, mod);
 for (int i = 0, j = 1; j < n - 1; ++j)
 {
  for (int k = n >> 1; k > (i ^= k); k >>= 1);
  if (j < i) swap(x[i], x[j]);
 }
 for (int m = 1; m < n; m *= 2)
 {
  ll w = 1, wk = pow(h, n / (2 * m), mod);
  for (int i = 0; i < m; ++i)
  {
   for (int s = i; s < n; s += 2 * m)
   {
    ll u = x[s], d = x[s + m] * w % mod;
    if ((x[s] = u + d) >= mod)
     x[s] -= mod;
    if ((x[s + m] = u - d) < 0)
     x[s + m] += mod;
   }
   w = w * wk % mod;
  }
 }
 if (sign < 0)
 {
  ll inv = div(1, n, mod);
  for (auto &a : x)
   a = a * inv % mod;
 }
}
// assume: size of a/b is power of two, mod is predetermined
template<int mod>
vector<ll> conv(vector<ll> a, vector<ll> b)
{
 fmt<mod, +1>(a);
 fmt<mod, +1>(b);
 for (int i = 0; i < a.size(); ++i)
  a[i] = a[i] * b[i] % mod;
 fmt<mod, -1>(a);
 return a;
}
// general convolution where mod < 2^31.
vector<ll> conv(vector<ll> a, vector<ll> b, ll mod)
{
 int n = a.size() + b.size() - 1;
 for (int k : { 1, 2, 4, 8, 16 })
  n |= (n >> k);
 ++n;
 a.resize(n);
 b.resize(n);
 const int A = 167772161, B = 469762049, C = 1224736769, D = (ll) (A) *
     ↪ B % mod;
 vector<ll> x = conv<A>(a, b), y = conv<B>(a, b), z = conv<C>(a, b);
 for (int i = 0; i < x.size(); ++i)
 {
  ll X = (y[i] - x[i]) * 104391568;
  if ((X %= B) < 0)
   X += B;
  ll Y = (z[i] - (x[i] + A * X) % C) * 721017874;
  if ((Y %= C) < 0)
   Y += C;
  x[i] += A * X + D * Y;
  if ((x[i] %= mod) < 0)
   x[i] += mod;
 }
 x.resize(n);
 return x;
}
poly mul(poly p, poly q, ll M)
{
 poly pq = conv(p, q, M);
 pq.resize(p.size() + q.size() - 1);
 while (!pq.empty() && !pq.back())
  pq.pop_back();
 return pq;
}
// Newton division: O(M(n)); M is the complexity of multiplication
// fast when FFT multiplication is used
//
// Note: complexity = M(n) + M(n/2) + M(n/4) + ... <= 2 M(n).
pair<poly, poly> divmod(poly p, poly q, ll M)
{
 if (p.size() < q.size())
  return {{}, p};
 reverse(p.begin(), p.end());
 reverse(q.begin(), q.end());
 poly t = { div(1, q[0], M) };
 if (t[0] < 0)
  return {{},{}}; // infeasible
 for (int k = 1; k <= 2 * (p.size() - q.size() + 1); k *= 2)
 {
  poly s = mul(mul(t, q, M), t, M);
  t.resize(k);
  for (int i = 0; i < k; ++i)
   t[i] = sub(2 * t[i], s[i], M);
 }
 t.resize(p.size() - q.size() + 1);
 t = mul(t, p, M);
 t.resize(p.size() - q.size() + 1);
 reverse(t.begin(), t.end());
 reverse(p.begin(), p.end());
 reverse(q.begin(), q.end());
 while (!t.empty() && !t.back())
  t.pop_back();
 return {t, sub(p, mul(q, t, M), M)};
}
// polynomial GCD: O(M(n) log n);
poly gcd(poly p, poly q, ll M)
{
 for (; !p.empty(); swap(p, q = divmod(q, p, M).second));
 return p;
}
// value of p(x)
ll eval(poly p, ll x, ll M)
{
 ll ans = 0;
 for (int i = p.size() - 1; i >= 0; --i)
  ans = add(mul(ans, x, M), p[i], M);
 return ans;
}
```

```
//
// faster multipoint evaluation
// fast if |x| >= 10000.
//
// algo:
//  evaluate(p, {x[0], ... , x[n-1]})
//  = evaluate(p mod (X-x[0]) ... (X-x[n/2-1]), {x[0], ... , x[n/2-1]}),
//  + evaluate(p mod (X-x[n/2]) ... (X-x[n-1]), {x[n/2], ... , x[n-1]}),
//
// f(n) = 2 f(n/2) + M(n) ==> O(M(n) log n)
//
vector<ll> evaluate(poly p, vector<ll> x, ll M)
{
 vector<poly> prod(8 * x.size()); // segment tree
 function<poly(int, int, int)> run = [&](int i, int j, int k)
 {
  if (i == j) return prod[k] = (poly) {1};
  if (i+1 == j) return prod[k] = (poly) { M-x[i], 1};
  return prod[k] = mul(run(i,(i+j)/2,2*k+1), run((i+j)/2,j,2*k+2), M);
 };
 run(0, x.size(), 0);
 vector<ll> y(x.size());
 function<void(int, int, int, poly)> rec = [&](int i, int j, int k, poly
      p)
 {
  if (j - i <= 8)
  {
   for (; i < j; ++i) y[i] = eval(p, x[i], M);
  }
  else
  {
   rec(i, (i+j)/2, 2*k+1, divmod(p, prod[2*k+1], M).second);
   rec((i+j)/2, j, 2*k+2, divmod(p, prod[2*k+2], M).second);
  }
 };
 rec(0, x.size(), 0, p);
 return y;
}
poly interpolate_n(vector<ll> x, vector<ll> y, ll M)
{
 int n = x.size();
 vector<ll> dp(n + 1);
 dp[0] = 1;
 for (int i = 0; i < n; ++i)
 {
  for (int j = i; j >= 0; --j)
  {
   dp[j + 1] = add(dp[j + 1], dp[j], M);
   dp[j] = mul(dp[j], M - x[i], M);
  }
 }
 poly r(n);
 for (int i = 0; i < n; ++i)
 {
  ll den = 1, res = 0;
  for (int j = 0; j < n; ++j)
   if (i != j)
    den = mul(den, sub(x[i], x[j], M), M);
  den = div(1, den, M);
  for (int j = n - 1; j >= 0; --j)
  {
   res = add(dp[j + 1], mul(res, x[i], M), M);
   r[j] = add(r[j], mul(res, mul(den, y[i], M), M), M);
  }
 }
 while (!r.empty() && !r.back())
  r.pop_back();
 return r;
}
//
// faster algo to find a poly p such that
//  p(x[i]) = y[i] for each i
//
// see http://people.mpi-inf.mpg.de/~csaha/lectures/lec6.pdf
//
poly interpolate(vector<ll> x, vector<ll> y, ll M)
{
 vector<poly> prod(8 * x.size()); // segment tree
 function<poly(int, int, int)> run = [&](int i, int j, int k)
 {
  if (i == j) return prod[k] = (poly) {1};
  if (i+1 == j) return prod[k] = (poly) {M-x[i], 1};
  return prod[k] = mul(run(i,(i+j)/2,2*k+1), run((i+j)/2,j,2*k+2), M);
 };
 run(0, x.size(), 0); // preprocessing in O(n log n) time
 poly H = prod[0]; // newton polynomial
 for (int i = 1; i < H.size(); ++i)
  H[i - 1] = mul(H[i], i, M);
 do
  H.pop_back();
 while (!H.empty() && !H.back());
 vector<ll> u(x.size());
 function<void(int, int, int, poly)> rec = [&](int i, int j, int k, poly
      p)
 {
  if (j - i <= 8)
  {
   for (; i < j; ++i) u[i] = eval(p, x[i], M);
  }
  else
  {
   rec(i, (i+j)/2, 2*k+1, divmod(p, prod[2*k+1], M).second);
   rec((i+j)/2, j, 2*k+2, divmod(p, prod[2*k+2], M).second);
  }
 };
 rec(0, x.size(), 0, H); // multipoint evaluation
 for (int i = 0; i < x.size(); ++i)
  u[i] = div(y[i], u[i], M);
 function<poly(int, int, int)> f = [&](int i, int j, int k)
 {
  if (i >= j) return poly();
  if (i+1 == j) return (poly) {u[i]};
  return add(mul(f(i,(i+j)/2,2*k+1), prod[2*k+2], M),
     mul(f((i+j)/2,j,2*k+2), prod[2*k+1], M), M);
 };
 return f(0, x.size(), 0);
}
//
// return p(x+a)
//
poly shift_n(poly p, ll a, ll M)
{
 poly q(p.size());
 for (int i = p.size() - 1; i >= 0; --i)
 {
  for (int j = p.size() - i - 1; j >= 1; --j)
   q[j] = add(mul(q[j], a, M), q[j - 1], M);
  q[0] = add(mul(q[0], a, M), p[i], M);
 }
 return q;
}
//
// faster algorithm for computing p(x + a)
//
// fast if n >= 4096
// algo: p(x+a) = p_h(x) (x+a)^m + q_h(x)
// cplx: preproc: O(M(n))
//     div-con: O(M(n) log n)
//
poly shift(poly p, ll a, ll M)
{
 vector<poly> pow(p.size());
 pow[0] = {1};
 pow[1] = {a,1};
 int m = 2;
 for (; m < p.size(); m *= 2)
  pow[m] = mul(pow[m / 2], pow[m / 2], M);
```

```
 function<poly(poly, int)> rec = [&](poly p, int m)
 {
  if (p.size() ≤ 1) return p;
  while (m ≥ p.size()) m /= 2;
  poly q(p.begin() + m, p.end());
  p.resize(m);
  return add(mul(rec(q, m), pow[m], M), rec(p, m), M);
 };
 return rec(p, m);
}
//
// overpeform when n ≥ 134217728 lol
//
ll factmod(ll n, ll M)
{
 if (n ≤ 1)
  return 1;
 ll m = sqrt(n);
 function<poly(int, int)> get = [&](int i, int j)
 {
  if (i == j) return poly();
  if (i+1 == j) return (poly) {i,1};
  return mul(get(i, (i+j)/2), get((i+j)/2, j), M);
 };
 poly p = get(0, m); // = x (x+1) (x+2) ... (x+(m-1))
 vector<ll> x(m);
 for (int i = 0; i < m; ++i)
  x[i] = 1 + i * m;
 vector<ll> y = evaluate(p, x, M);
 ll fac = 1;
 for (int i = 0; i < m; ++i)
  fac = mul(fac, y[i], M);
 for (ll i = m * m + 1; i ≤ n; ++i)
  fac = mul(fac, i, M);
 return fac;
}
ll factmod_n(ll n, ll M)
{
 ll fac = 1;
 for (ll k = 1; k ≤ n; ++k)
  fac = mul(k, fac, M);
 return fac;
}
ll factmod_p(ll n, ll M)
{
 // only works for prime M
 ll fac = 1;
 for (; n > 1; n /= M)
 {
  fac = mul(fac, (n / M) % 2 ? M - 1 : 1, M);
  for (ll i = 2; i ≤ n % M; ++i)
   fac = mul(fac, i, M);
 }
 return fac;
}
```

9.23 SLAE.cpp

SLAE.cpp 62 lines

```
#include <bits/stdc++.h>
using namespace std;
const int N = 101;
/*
  Solve lineal algrebraic ecuations for Z2
*/
int gauss (vector <bitset<N>> a, int n, int m, bitset<N> & ans) {
  vector<int> where (m, -1);
  for (int col=0, row=0; col<m && row<n; ++col) {
    for (int i=row; i<n; ++i)
      if (a[i][col]) {
        swap (a[i], a[row]);
        break;
      }
    if (! a[row][col])
      continue;
    where[col] = row;
    for (int i=0; i<n; ++i)
      if (i ≠ row && a[i][col])
        a[i] ^= a[row];
    ++row;
  }
  for (int i=0; i<m; ++i)
    if (where[i] ≠ -1)
      ans[i] = a[where[i]][m];
  for (int i=0; i<n; ++i) {
    int sum = 0;
    for (int j=0; j<m; ++j)
      sum += ans[j] * a[i][j];
    if (sum - a[i][m])
      return 0;
  }
  for (int i=0; i<m; ++i)
    if (where[i] == -1)
      return 1e9;
  return 1;
}
vector<int> graph[107];
int main(){
  int n,m,u,v;
  cin>>n>>m;
  for(int i = 0;i<m;i++){
    cin>>u>>v;
    u--,v--;
    graph[u].push_back(v);
    graph[v].push_back(u);
  }
  vector<bitset<N>> SLAE(n);
  for(int i = 0;i<n;i++){
    for(auto c:graph[i])
      SLAE[i].set(c);
    if(graph[i].size()&1)
      SLAE[i].set(i);
    else
      SLAE[i].set(n);
  }
  bitset<N>ans(0);
  if(gauss(SLAE,n,n,ans))
    cout<<"Y"<<endl;
  else cout<<"N"<<endl;
  return 0;
}
```

9.24 Simplex.h

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

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

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

Simplex.h aa8530, 62 lines

```
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;
const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j
struct LPSolver {
 int m, n;
 vi N, B;
 vvd D;
 LPSolver(const vvd& A, const vd& b, const vd& c) :
  m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
   rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
   rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
```

```
    rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m+1][n] = 1;
  }
  void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
      T *b = D[i].data(), inv2 = b[s] * inv;
      rep(j,0,n+2) b[j] -= a[j] * inv2;
      b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }
  bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
      int s = -1;
      rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
      if (D[x][s] >= -eps) return true;
      int r = -1;
      rep(i,0,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                     < MP(D[r][n+1] / D[r][s], B[r])) r = i;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  }
  T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
      pivot(r, n);
      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
      rep(i,0,m) if (B[i] == -1) {
        int s = 0;
        rep(j,1,n+1) ltj(D[i]);
        pivot(i, s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
  }
};
```

9.25 SolveLinear.h

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

SolveLinear.h — 44c9ab, 35 lines

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

9.26 SolveLinear2.h

SolveLinear2.h — 8 lines

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

9.27 SolveLinearBinary.h

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

SolveLinearBinary.h — fa2d7a, 32 lines

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

9.28 SubsetSum.cpp

SubsetSum.cpp — 79 lines

```
// number of subsets of an array of n elements having sum equal to k for
    each k from 1 to m
int main() {
```

```cpp
  int n, m; cin >> n >> m;
  vector<int> a(m + 1, 0);
  for (int i = 0; i < n; i++) {
    int k; cin >> k; // k >= 1, handle [k = 0] separately
    if (k <= m) a[k]++;
  }
  poly p(m + 1, 0);
  for (int i = 1; i <= m; i++) {
    for (int j = 1; i * j <= m; j++) {
      if (j & 1) p.a[i * j] += mint(a[i]) / j;
      else p.a[i * j] -= mint(a[i]) / j;
    }
  }
  p = p.exp(m + 1);
  for (int i = 1; i <= m; i++) cout << p[i] << ' '; cout << '\n'; //
      ↪ check for m = 0
  return 0;
}
// Calc bell numbers
vector<mint> bell(int n) { // e^(e^x - 1)
  poly p(n + 1);
  mint f = 1;
  for (int i = 0; i <= n; i++) {
    p.a[i] = mint(1) / f;
    f *= i + 1;
  }
  p.a[0] -= 1;
  p = p.exp(n + 1);
  vector<mint> ans(n + 1);
  f = 1;
  for (int i = 0; i <= n; i++) {
    ans[i] = p[i] * f;
    f *= i + 1;
  }
  return ans;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n; cin >> n;
  auto ans = bell(n);
  cout << ans[n] << '\n';
  return 0;
}
//$O((MAXK^2)/32)$
//MAXK is the maximum possible sum.
bitset<MAXK> dp;
dp[0] = 1;
for (int i = 1; i <= n; i++) {
  for (int x = 0; (1<<x) <= m[i]; x++) {
    dp |= (dp << (a[i]*(1<<x)));
    m[i] -= (1<<x);
  }
  dp |= (dp << (a[i]*m[i]));
}
long long kSum(vector<int>& nums, int k) {
  ll sum=0,n=nums.size();
      vector<ll> ans;
      for(ll i=0;i<nums.size();i++){
          if(nums[i]>0)
              sum+=nums[i];
          nums[i]=abs(nums[i]);
      }
      ans.push_back(sum);
      priority_queue<pair<ll,ll>> pq;
      sort(nums.begin(),nums.end());
      pq.push({sum-nums[0],0});
      while(ans.size()<k){
          auto [sum,ind]=pq.top();
          pq.pop();
          if(ind+1<n){
              pq.push({sum+nums[ind]-nums[ind+1],ind+1});
              pq.push({sum-nums[ind+1],ind+1});
          }
          ans.push_back(sum);
      }
      return ans.back();
  }
```

9.29 Tridiagonal.h

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

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

This is useful for solving problems on the type

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

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

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

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

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

Tridiagonal.h — 8f9fa8, 26 lines

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

9.30 gauss.cpp

gauss.cpp — 48 lines

```cpp
/*
  Tested: SPOJ GS
  Complexity: O(n^3)
*/
const int oo = 0x3f3f3f3f;
const double eps = 1e-9;
int gauss(vector<vector<double>> a, vector<double> &ans)
{
  int n = (int) a.size();
  int m = (int) a[0].size() - 1;
  vector<int> where(m, -1);
  for (int col = 0, row = 0; col < m && row < n; ++col)
  {
    int sel = row;
    for (int i = row; i < n; ++i)
      if (abs(a[i][col]) > abs(a[sel][col]))
        sel = i;
    if (abs(a[sel][col]) < eps)
      continue;
    for (int i = col; i <= m; ++i)
```

```
    swap(a[sel][i], a[row][i]);
   where[col] = row;
   for (int i = 0; i < n; ++i)
    if (i != row)
    {
     double c = a[i][col] / a[row][col];
     for (int j = col; j <= m; ++j)
      a[i][j] -= a[row][j] * c;
    }
   ++row;
  }
  ans.assign(m, 0);
  for (int i = 0; i < m; ++i)
   if (where[i] != -1)
    ans[i] = a[where[i]][m] / a[where[i]][i];
  for (int i = 0; i < n; ++i)
  {
   double sum = 0;
   for (int j = 0; j < m; ++j)
    sum += ans[j] * a[i][j];
   if (abs(sum - a[i][m]) > eps)
    return 0;
  }
  for (int i = 0; i < m; ++i)
   if (where[i] == -1)
    return oo;
  return 1;
}
```

9.31 matrixMul.cpp

matrixMul.cpp 24 lines

```
vector<vector<ld>> mult(vector<vector<ld>> &A,vector<vector<ld>> &B){
    int n = A.size();
    vector<vector<ld>> C(n,vector<ld>(n));
    for(int i = 0;i<n;i++){
        for(int j = 0;j<n;j++){
            for(int k = 0;k<n;k++){
                C[i][j] += (A[i][k] * B[k][j]);
            }
        }
    }
    return C;
}
vector<vector<ld>> pw(vector<vector<ld>> M,int n){
    int N = M.size();
    vector<vector<ld>> X(N,vector<ld> (N));
    for(int i = 0;i<N;i++)
        X[i][i] = 1.0;
    while(n){
        if(n&1)X = mult(M,X);
        M = mult(M,M);
        n >>=1;
    }
    return X;
}
```

9.32 simplex copy.cpp

simplex copy.cpp 205 lines

```
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define int long long
#define __ ios_base::sync_with_stdio(false);cin.tie(NULL);
/*
    [Simplex method]
    [Tested in RPC 2021 02 problem C]
*/
vector<vector<int>> M;
void print(){
    for(auto c:M){
        for(auto d:c)cout<<d<<" ";
        cout<<endl;
    }
}
int getPivotCol(int ncols,vector<int> row0){
    for(int i=0; i<ncols-1; i++)
        if (row0[i] > 0.0)
        return i;
    return -1;
}
int dual_simplex(int nrows, int ncols){
    int pcol, prow;
    while((pcol = getPivotCol(ncols, M[0])) != -1) {
        prow = -1;
        int minval;
        for(int i=1; i<nrows; i++) {
            if (M[i][pcol] <= 0)
                continue;
            if (prow == -1 || M[i][ncols-1]/M[i][pcol] < minval) {
                prow = i;
                minval = M[i][ncols-1]/M[i][pcol];
            }
        }
        assert(prow != -1);
        for(int i=0; i<nrows; i++) {
            if (i == prow) {
                int factor = M[prow][pcol];
                for(int j=0; j<ncols; j++)
                    M[i][j] /= factor;
            }
            else {
                int factor = M[i][pcol]/M[prow][pcol];
                for(int j=0; j<ncols; j++) {
                    M[i][j] -= M[prow][j]*factor;
                }
            }
        }
    }
    print();
    return M[0][ncols-1];
}
/* Get maximum for a linear ecuation a1*x1 + a2*x2 + a3*x3 ... ai*xi
    subject to
            b1*x1 + b2*x2 + b3*x3 + ... + ai*xi<= z1
            c1*x1 + c2*x2 + c3*x3 + ... + ci*xi<= z2
            d1*x1 + d2*x2 + d3*x3 + ... + di*xi<= z3
                   .
                   .
                   .
            y1*x1 + y2*x2 + y3*x3 + ... + yi*xi<= z_i
      for given ai , bi, ci, di, zi
      example
         maximize x1+ 2x2 + 5x2
      subject to
         2x1 + x2 +x3 <=14
         4x1 +2x2 +3x3 <=28
         2x1 +5x2 +5x3 <=30
      x1,x2,x3 >=0
      if you have a restriction like
         4x1 +2x2 +3x3 >= 28
      only multiply by -1 and you have the correct form
         -4x1 -2x2 -3x3 <= -28
   the matrix has the form
   a1 a2 a3 ... ai 0 0 0 ... 0 0
   b1 b2 b3 ... bi 1 0 0 ... 0 z1
   c1 c2 c3 ... ci 0 1 0 ... 0 z2
   d1 d2 d3 ... di 0 0 1 ... 0 z3
   .
   .
   .
   y1 y2 y3 ... yi 0 0 0 ... 1 zi
   note the identity matrix in middle of the ecuation and the restriction and the first row is the linear equation to maximize
*/
```

```cpp
signed main(){__
 int t= 1,n,m;
  while(t--){
    cin>>n>>m;
    vector<vector<int>> C(n,vector<int>(m));
    vector<int> a(n);
    vector<int> b(m);
    for(auto &c:a)cin>>c;
    for(auto &c:b)cin>>c;
    for(auto &c:C)for(auto &d:c)cin>>d;
    M.resize(2+(n*m),vector<int>((n*m)+2+(n*m)));
    for(int i = 0;i<n*m;i++)
      M[0][i] = C[i/m][i%m];
    for(int i = 0;i<n*m;i++)
      M[1][i] = -b[i%3];
    for(int i = 2;i<(n*m)+2;i++)
      M[i][i-2] = 1;
    for(int i = 2;i<(n*m)+2;i++)
      M[i].back() = 1;
    for(int i = 0;i<n;i++)
      M[1][(n*m)+1+(n*m)] -=a[i];
    for(int i = 1;i<(n*m)+2;i++)
      M[i][i+(n*m)-1] = 1;
    // M.resize((n*m)+1,vector<int>(2+(n*m)+(n*m)));
    // for(int i = 0;i<n;i++)
    //   M[0][0] -= a[i];
    // for(int i = 1;i<(n*m)+1;i++)
    //   M[0][i] = 1;
    // for(int i = 1;i<=n*m;i++)
    //   M[i][0] = -b[(i-1)%3];
    // for(int i = 1;i<=n*m;i++){
    //   M[i][i] = 1;
    //   M[i][i+(n*m)] = 1;
    // }
    // for(int i = 1;i<=n*m;i++){
    //   M[i].back() = C[(i-1)/m][(i-1)%m];
    // }
    print();
    // int mn = simplex((n*m)+1,(n*m)+2+(n*m));
    // cout<<mn<<endl;
  }
  return 0;
}
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define int long long
#define __ ios_base::sync_with_stdio(false);cin.tie(NULL);
#define ld long double
const ld eps = 1e-9;
vector<vector<ld>> M;
int getPivotCol(int ncols,vector<ld> row0){
  for(int i=0; i<ncols-1; i++)
    if (row0[i] > 0.0)
    return i;
  return -1;
}
void print(){
  for(auto c:M){
    for(auto d:c)cout<<d<<" ";
    cout<<endl;
  }
}
ld simplex(int nrows, int ncols){
  int pcol, prow;
  while((pcol = getPivotCol(ncols, M[0])) != -1) {
    prow = -1;
    ld minval;
    for(int i=1; i<nrows; i++) {
      if (M[i][pcol] <= 0.0)
        continue;
      if (prow == -1 || M[i][ncols-1]/M[i][pcol] < minval) {
        prow = i;
        minval = M[i][ncols-1]/M[i][pcol];
      }
    }
    assert(prow != -1);
    for(int i=0; i<nrows; i++) {
      if (i == prow) {
        ld factor = M[prow][pcol];
        for(int j=0; j<ncols; j++)
          M[i][j] /= factor;
      }
      else {
        ld factor = M[i][pcol]/M[prow][pcol];
        for(int j=0; j<ncols; j++) {
          M[i][j] -= M[prow][j]*factor;
          if (fabs(M[i][j]) < eps)
            M[i][j] = 0.0;
        }
      }
    }
  }
  return M[0][ncols-1];
}
signed main(){
  int n, m;
  cout << setprecision(2);
  cout << fixed;
  cin >> n >> m;
  M.resize(n+1,vector<ld>(n+m+1));
  for(int i=0; i<n+1; i++)
    for(int j=0; j<n+m+1; j++)
      M[i][j] = 0.0;
  for(int i=1; i<=n; i++)
    cin >> M[i][n+m];
  for(int j=0; j<m; j++) {
    for(int i=1; i<=n; i++) {
      cin >> M[i][j];
     M[i][j] /= 100.0;
    }
    cin >> M[0][j];
  }
  for(int j=m; j<n+m; j++)
    M[j-m+1][j] = 1.0;
  // print();
  ld max = simplex(n+1, n+m+1);
  cout<<-max<<endl;
}
// a_1x1 +a_2x2 + a_
```

9.33 simplex.cpp

simplex.cpp 86 lines

```cpp
/*
 Parametric Self-Dual Simplex method
 Description:
 - Solve a canonical LP:
    min. c x
   s.t. A x <= b
    x >= 0
 Complexity: O(n+m) iterations on average
 Tested: http://codeforces.com/contest/375/problem/E
*/
const double eps = 1e-9, oo = numeric_limits<double>::infinity();
typedef vector<double> vec;
typedef vector<vec> mat;
double simplexMethodPD(mat &A, vec &b, vec &c)
{
 int n = c.size(), m = b.size();
 mat T(m + 1, vec(n + m + 1));
 vector<int> base(n + m), row(m);
 for(int j = 0; j < m; ++j)
 {
  for (int i = 0; i < n; ++i)
   T[j][i] = A[j][i];
  T[j][n + j] = 1;
```

```
  base[row[j] = n + j] = 1;
  T[j][n + m] = b[j];
}
for (int i = 0; i < n; ++i)
  T[m][i] = c[i];
while (1)
{
  int p = 0, q = 0;
  for (int i = 0; i < n + m; ++i)
    if (T[m][i] <= T[m][p])
      p = i;
  for (int j = 0; j < m; ++j)
    if (T[j][n + m] <= T[q][n + m])
      q = j;
  double t = min(T[m][p], T[q][n + m]);
  if (t >= -eps)
  {
    vec x(n);
    for (int i = 0; i < m; ++i)
      if (row[i] < n) x[row[i]] = T[i][n + m];
    // x is the solution
    return -T[m][n + m]; // optimal
  }
  if (t < T[q][n + m])
  {
    // tight on c -> primal update
    for (int j = 0; j < m; ++j)
      if (T[j][p] >= eps)
        if (T[j][p] * (T[q][n + m] - t) >=
          T[q][p] * (T[j][n + m] - t))
          q = j;
    if (T[q][p] <= eps)
      return oo; // primal infeasible
  }
  else
  {
    // tight on b -> dual update
    for (int i = 0; i < n + m + 1; ++i)
      T[q][i] = -T[q][i];
    for (int i = 0; i < n + m; ++i)
      if (T[q][i] >= eps)
        if (T[q][i] * (T[m][p] - t) >=
          T[q][p] * (T[m][i] - t))
          p = i;
    if (T[q][p] <= eps)
      return -oo; // dual infeasible
  }
  for (int i = 0; i < m + n + 1; ++i)
    if (i != p) T[q][i] /= T[q][p];
  T[q][p] = 1; // pivot(q, p)
  base[p] = 1;
  base[row[q]] = 0;
  row[q] = p;
  for (int j = 0; j < m + 1; ++j)
    if (j != q)
    {
      double alpha = T[j][p];
      for (int i = 0; i < n + m + 1; ++i)
        T[j][i] -= T[q][i] * alpha;
    }
  }
  return oo;
}
```

9.34 simpson.cpp

simpson.cpp 13 lines

```
template<class F>
double simpson(F f, double a, double b, int n = 2000)
{
  double h = (b - a) / (2 * n), fa = f(a), nfa, res = 0;
  for (int i = 0; i < n; ++i, fa = nfa)
  {
    nfa = f(a + 2 * h);
    res += (fa + 4 * f(a + h) + nfa);
    a += 2 * h;
  }
  res = res * h / 3;
  return res;
}
```

data-structures (10)

10.1 FenwickTree.cpp

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

FenwickTree.cpp 5b7e48, 50 lines

```
//Usefull define to print vectors
#define print(A)for(auto c:A)cout<<c<<" ";cout<<endl;
#define printM(A)for(auto c:A){print(c);}
vector<int> bit;
vector<vector<int>> bit2D;
int n,m;
int sum(int idx) {
    int ret = 0;
    for (++idx; idx > 0; idx -= idx & -idx)ret += bit[idx];
    return ret;
}
int sum(int l, int r) {
    return sum(r) - sum(l - 1);
}
void add(int idx, int delta) {
    for (++idx; idx < n; idx += idx & -idx) bit[idx] += delta;
}
/** This only can accept querys in a point */
void range_add(int l, int r, int val) {
    add(l, val);
    add(r + 1, -val);
}
// Search for first position such \sum_{0}^{pos} a[i] \geq s;
int bit_search(int s){
    int sum = 0;
    int pos = 0;
    for(int i = ceil(log2(n));i>=0;i--){
        if((pos+(1<<i))<n && (sum+bit[pos+(1<<i)])<s){
            sum+=bit[pos+(1<<i)];
            pos+=(1<<i);
        }
    }
    return pos;
}
// Return sum over submatrix with corners (0,0), (x,y)
int sum2D(int x, int y) {
    int ret = 0;
    for (int i = x; i >= 0; i = (i & (i + 1)) - 1)
        for (int j = y; j >= 0; j = (j & (j + 1)) - 1)
            ret += bit2D[i][j];
    return ret;
}
int sum2D(int x0,int y0,int x,int y){
    return sum2D(x,y)-sum2D(x,y0-1)-sum2D(x0-1,y)+sum2D(x0-1,y0-1);
}
void add2D(int x, int y, int delta) {
    for (int i = x; i < n; i = i | (i + 1))
        for (int j = y; j < m; j = j | (j + 1))
            bit2D[i][j] += delta;
}
```

10.2 HashMap.cpp

HashMap.cpp 10 lines

```
#include <bits/stdc++.h>
```

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

10.3 ImplicitTreap.cpp

Description: A powerfull dynamic array that allows operations like: Insert/erase in every position, Range sum/minimum/max, Reverse/Rotate a sub array

Usage: `Root is global and not need modifications`
`Only erase need root -> erase(root,pos)`
`All operations are 0 indexed`

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

ImplicitTreap.cpp df5d5a, 162 lines

```cpp
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<> dis(numeric_limits<int>::min(),
    ↪ numeric_limits<int>::max()) ;
struct Treap{
   Treap *l = NULL,*r = NULL;
   int p,sz = 1,val,sum = 0,mn = 1e9;
   int rev = 0, lazySum = 0,lazyReplace = 0;
   int sumPend = 0,ReplacePend = 0;
   Treap(int v ,int prior = dis(rng)):val(v),p(prior){}
};
Treap *root = NULL;
void update(Treap *T){
   if (!T) return;
   T->sz = 1;
   T->sum = T->val;
   T->mn = T->val;
   if (T->l) {
     T->sz += T->l->sz;
     T->sum += T->l->sum;
     T->mn = min(T->mn,T->l->mn);
   }
   if (T->r) {
     T->sz += T->r->sz;
     T->sum += T->r->sum;
     T->mn = min(T->mn,T->r->mn);
   }
}
void applyRev(Treap *T){
   if(!T)return;
   T->rev^=1;
   swap(T->l,T->r);
}
void applySum(Treap *T,int x){
   if(!T)return;
   T->val+=x;
   T->mn+=x;
   T->sumPend+=x;
   T->lazySum = 1;
   T->sum+=x*T->sz;
}
void applyReplace(Treap *T,int x){
   if(!T)return;
   T->val=x;
   T->mn=x;
   T->ReplacePend=x;
   T->lazyReplace = 1;
   T->sum=x*T->sz;
}
void lazy(Treap *T){
   if(!T)return;
   if(T->rev){
      applyRev(T->l);
      applyRev(T->r);
      T->rev = 0;
   }
   if(T->lazySum){
      applySum(T->l,T->sumPend);
      applySum(T->r,T->sumPend);
      T->lazySum = 0;
      T->sumPend = 0;
   }
   if(T->lazyReplace){
      applyReplace(T->l,T->ReplacePend);
      applyReplace(T->r,T->ReplacePend);
      T->lazyReplace = 0;
      T->ReplacePend = 0;
   }
}
pair<Treap*,Treap*> split(Treap *T,int idx,int cont = 0){
   if(!T)return {NULL,NULL};
   lazy(T);
   Treap *L, *R;
   int idxt = cont + (T->l?T->l->sz:0);
   if(idx<idxt)
      tie(L,T->l) = split(T->l,idx,cont),R = T;
   else
      tie(T->r,R) = split(T->r,idx,idxt+1),L = T;
   update(T);
   return {L,R};
}
void insert(Treap *&T,Treap *v,int x, int cnt) {
   lazy(T);
   int idxt = T ? cnt + (T->l ? T->l->sz : 0) : 0;
   if (!T) T = v;
   else if (v->p > T->p)
      tie(v->l, v->r) = split(T, x, cnt), T = v;
   else if (x < idxt) insert(T->l, v, x, cnt);
   else insert(T->r, v, x, idxt + 1);
   update(T);
}
void insert(int e, int i) {
   insert(root, new Treap(e), i-1, 0);
}
Treap *merge(Treap *a,Treap *b){
   lazy(a),lazy(b);
   Treap *T;
   if(!a || !b)T = a?a:b;
   else if(a->p > b->p)
      a->r = merge(a->r,b),T = a;
   else b->l = merge(a,b->l),T = b;
   update(T);
   return T;
}
void erase(Treap *&T,int x ,int cnt = 0){
   if(!T)return;
   lazy(T);
   int left = cnt+(T->l? T->l->sz:0);
   if(left == x)T = merge(T->l,T->r);
   else if(x<left)erase(T->l,x,cnt);
   else erase(T->r,x,left+1);
   update(T);
}
void print(Treap *t) {
   if (!t) return;
   lazy(t);
   print(t->l);
   print(t->r);
}
void push_back(int e) {
   root = merge(root, new Treap(e));
}
void op(int l,int r, function<void(Treap *T)> f){
   Treap *a,*b,*c;
   tie(a,b) = split(root,l-1);
   tie(b,c) = split(b,r-l);
   f(b);
   root = merge(a, merge(b,c));
}
void reverse(int l,int r){
```

```
    op(l,r,[&](Treap *T){applyRev(T);});
}
void rotate(int l,int r,int k){
    op(l,r,[&](Treap *T){
        Treap *l,*r;
        k%=T->sz;
        tie(l,r) = split(T,T->sz-k-1);
        T = merge(r,l);
    });
}
void add(int l,int r,int x){
    op(l,r,[&](Treap *T){
        applySum(T,x);
    });
}
void replace(int l,int r,int x){
    op(l,r,[&](Treap *T){
        applyReplace(T,x);
    });
}
int get_sum(int l,int r){
    int ans;
    op(l,r,[&](Treap *T){
        ans = T->sum;
    });
    return ans;
}
int get_min(int l,int r){
    int mn;
    op(l,r,[&](Treap *T){
        mn = T->mn;
    });
    return mn;
}
```

10.4 IntervalTree.cpp

Description: Interval tree is a structure that stores segments in a efficient way, allows to get all intervals that intersects with another interval
Usage: `root = build_interval_tree(vector<recta>);`
`query(root,R); R is an instance of recta`
Time: $\mathcal{O}(ans)$ where ans is the number of intervals that intersects.

IntervalTree.cpp 5df955, 105 lines

```
typedef long long int lli;
typedef long double ld;
struct recta {
    ld x1,x2;
    int id;
    friend ostream& operator << (ostream &out, const recta&p ){
        out<<"("<<p.x1<<","<<p.x2<<", "<<p.id<<")";
        return out;
    }
};
struct central{
    ld x;
    vector<recta> x1order;
    vector<recta> x2order;
    friend ostream& operator <<(ostream &out, const central&p){
        out<<"[ ";
        for(int i = 0;i<p.x1order.size();i++){
            out<<p.x1order[i]<<" ";
        }
        out<<"]";
        return out;
    }
};
struct node {
  node *l, *r;
    central C;
  node(node *l, node *r, central C) :
    l(l), r(r),C(C) {}
};
inline bool leaf(node *x){
    return !x->l && !x->r;
}
node* build_interval_tree(vector<recta> &R){
    if(R.size() == 0)return NULL;
    int n = R.size();
    int mid = (n-1)>>1;
  vector<recta> r1,r2;
    central c;
    ld x = (R[mid].x1+R[mid].x2)/2.0;
    c.x = x;
    for(int i = 0;i<n;i++){
        if(islessequal(R[i].x1,x) && islessequal(x,R[i].x2)){
            c.x1order.push_back(R[i]);
            c.x2order.push_back(R[i]);
        }
        else if(R[i].x2<x)
            r1.push_back(R[i]);
        else
            r2.push_back(R[i]);
    }
    sort(c.x1order.begin(),c.x1order.end(),[&](recta a,recta b){
        return islessequal(a.x1,b.x1);
    });
    sort(c.x2order.begin(),c.x2order.end(),[&](recta a,recta b){
        return islessequal(a.x2,b.x2);
    });
  node *left = build_interval_tree(r1);
  node *right = build_interval_tree(r2);
  return new node(left,right,c);
}
set<lli> ids;
void findI(central C,recta R,bool dir){
    if(dir){
        int l = -1, r = C.x2order.size();
        while(l+1<r){
            int m = (l+r)>>1;
            if(isgreaterequal(C.x2order[m].x2,R.x1))
                r = m;
            else
                l = m;
        }
        int n = C.x2order.size();
        for(int i = r;i<n;i++)
            ids.insert(C.x2order[i].id);
    }
    else{
        int l = -1, r = C.x2order.size();
        while(l+1<r){
            int m = (l+r)>>1;
            if(islessequal(C.x1order[m].x1,R.x2))
                l = m;
            else
                r = m;
        }
        for(int i = l;i>=0;i--)
            ids.insert(C.x1order[i].id);
    }
}
void query(node *t, const recta& R){
    if(!t)return;
    if(isgreaterequal(t->C.x,R.x1) && islessequal(t->C.x,R.x2)){
        for(int i = 0;i<t->C.x1order.size();i++)
            ids.insert(t->C.x1order[i].id);
        query(t->l,R);
        query(t->r,R);
    }
    else if(isless(R.x2,t->C.x)){
        findI(t->C,R,0);
        query(t->l,R);
    }
    else{
        findI(t->C,R,1);
        query(t->r,R);
    }
}
```

10.5 LineContainer(CHT).cpp

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

LineContainer(CHT).cpp af1807, 30 lines

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

10.6 OrderStatisticTree.cpp

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

OrderStatisticTree.cpp 6ac8ab, 19 lines

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

10.7 QuadTree.cpp

Description: A divide and conquer structure that divides a plane in fou sections to answer efficiently queries like how many points are inside of a rectangle
Usage: `node* head = new node(range(0,MAXN,0,MAXN));` `count(range(S[i].x,MAXN-S[i].x,S[i].y,MAXN-S[i].y),head); Range is an structure for a rectangle`
Time: $\mathcal{O}(n \log N)$.

QuadTree.cpp e45e77, 87 lines

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

```cpp
    S.push_back({x,y});
    insert(point(x,y),head);
  }
  vector<int> ans;
  cout<<endl;
  for(int i = 0;i<n;i++){
    if(count(range(S[i].x,MAXN-S[i].x,S[i].y,MAXN-S[i].y),head)-count(range(S[i].x,0,S[i].y,0),head)==0)ans.push_back(i+1);
  }
  for(auto c:ans)cout<<c<<" ";
  cout<<endl;
  return 0;
}
```

10.8 SQRTDecomposition.cpp

Description: Find de fist and last ocurrence of x in an array with lazy updates
Time: $\mathcal{O}\left(sqrt(n)\right)$ per query

SQRTDecomposition.cpp 19aeb1, 95 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
#define print(A) for(auto c:A)cout<<c<<" ";cout<<endl;
#define printM(A) for(auto c:A){print(c);}
#define x first
#define y second
#define printP(A)for(auto c:A)cout<<"("<<c.x<<","<<c.y<<") ";cout<<endl;
#define printMP(A)for(auto c:A){printP(c);}
#define endl '\n'
#define MOD(n,k) ( ( ((n) % (k)) + (k) ) % (k))
#define ALL(A) A.begin(),A.end()
#define error(args...) { string _s = #args; replace(_s.begin(), _s.end(), ',', ' ');\
stringstream _ss(_s); istream_iterator<string> _it(_ss); err(_it, args); }
#define rep(i, begin, end) for (__typeof(end) i = (begin) - ((begin) > (end));\
i != (end) - ((begin) > (end)); i += 1 - 2 * ((begin) > (end)))
#define cerr(s) cerr << "\033[48;5;196m\033[38;5;15m" << s << "\033[0m"
void err(istream_iterator<string> it) {}
template<typename T, typename... Args>
void err(istream_iterator<string> it, T a, Args... args) {
 cerr << *it << " = " << a << endl;
 err(++it, args...);
}
typedef long long int lli;
const lli inf = 1000000000;
struct block{
  int l,r;
  int size;
  vector<pair<lli,lli>>A;
  lli plus = 0;
  block(int _l,int _r,int _size): l(_l), r(_r),size(_size){A.resize(size,{-1,-1});}
  void st(){
    sort(A.begin(),A.end());
  }
  void add(int a,int b,lli v){
    if(a>b || b<l ||a>r)return;
    if(a<l)a = l;
    if(b>r)b = r;
    // error(l,r,a,b)
    if(a == l&& b == r){plus+=v;return; }
    for(int i = 0;i<size;i++)
      if(A[i].y>=a && A[i].y<=b)A[i].first+=v;
    st();
  }
  pair<lli,lli> find(lli z){
    z-=plus;
    int index = lower_bound(A.begin(),A.end(),make_pair(z,-inf))-A.begin();
    if(A[index].x!= z)return{inf,-1};
    int index2 = lower_bound(A.begin(),A.end(),make_pair(z+1,-inf))-A.begin();
    index2--;
    // error(index,index2)
    return {A[index].second,A[index2].second};
  }
};
int main(){
  int n,m,t,v,l,r,z;
  cin>>n>>m;
  vector<int> nums(n);
  for(auto &c:nums)cin>>c;
  int raiz = ceil(sqrt(n*1.0));
  vector<block> bloques(raiz,block(0,0,0));
  vector<vector<pair<int,int>>> sq(raiz);
  for(int i = 0;i<raiz;i++){
    bloques[i] = block(i*raiz,((i*raiz)+raiz)-1,raiz);
    for(int j = 0;j<raiz;j++){
      if(i*raiz+j<n)
        bloques[i].A[j] = {nums[(i*raiz)+j],(i*raiz)+j};
    }
    bloques[i].st();
  }
  for(int i = 0;i<m;i++){
    cin>>t;
    if(t ==1){
      cin>>l>>r>>v;
      l--;
      r--;
      for(int i = 0;i<raiz;i++)
        bloques[i].add(l,r,v);
    }
    else{
      cin>>z;
      lli mn = n+2,mx = -1;
      for(int j = 0;j<raiz;j++){
        auto c = bloques[j].find(z);
        mn = min(c.x,mn);
        mx = max(c.y,mx);
        // error(c.x,c.y);
      }
      // error(mx,mn)
      if(mx == -1)cout<<-1<<endl;
      else cout<<mx-mn<<endl;
    }
  }
  // for(int i = 0;i<raiz;i++){printP(bloques[i].A);cout<<endl;}
  return 0;
}
```

10.9 SegmentTree.cpp

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

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

SegmentTree.cpp 454480, 233 lines

```cpp
/** ---- Recursive segment tree with lazy propagation ---- */
vector<int> st;
vector<int> lazy;
void propagate(int v,int l ,int r){
  if(!lazy[v])return ;
 // For asigments replace += to =
  st[v] += ((r-l)+1)*lazy[v];
  if(l!= r){
    lazy[v<<1] += lazy[v];
    lazy[v<<1|1]+= lazy[v];
  }
  lazy[v] = 0;
}
int n; /** n is global for use default values and send less parameters */
void update(int l,int r,int val,int v = 1,int sl = 0,int sr = n-1){
  propagate(v,sl,sr);
  if(r<sl || l>sr || sl>sr)return ;
```

```cpp
    if(sl>= l && sr<=r){
        lazy[v] += val;
        propagate(v,sl,sr);
        return;
    }
    int m = (sl+sr)>>1;
    update(l,r,val,v<<1,sl,m);
    update(l,r,val,v<<1|1,m+1,sr);
    st[v] = st[v<<1]+st[v<<1|1];
}
int query(int l,int r,int v = 1,int sl = 0,int sr = n-1){
    propagate(v,sl,sr);
    if(r<sl || l>sr || sl>sr)return 0;
    if(sl>= l && sr<=r)return st[v];
    int m = (sl+sr)>>1;
    return query(l,r,v<<1,sl,m)+query(l,r,v<<1|1,m+1,sr);
}
/** ---- Iterative segment tree much faster, setted to return min in a range ---- */
int inf = 1e9;
struct STmin{
    int n;
    vector<int> st;
    STmin(int n):n(n){
        st.resize(2*n,inf);
    }
  inline void update(int x, int val) {
    x += n;
    st[x] = val;
    for (; x >>=1 ; st[x] = min(st[x<<1], st[x<<1|1]));
  }
  inline int query(int l, int r) {
    int ans = inf;
        if(r<l)return 0;
    for (l += n, r += n; l <= r; l = (l + 1) / 2, r = (r - 1) / 2) {
      if (l & 1) ans = min(ans, st[l]);
      if (~r & 1) ans = min(ans, st[r]);
    }
    return ans;
  }
};
// Min and max in one same iterative segment tree
struct STMinMax{
    int n;
    vector<pair<int,int>> st;
    STMinMax(int n):n(n){
        st.resize(2*n,make_pair(inf,-inf));
    }
  inline void update(int x, int val) {
    x += n;
    st[x] = {val,val};
        while(x>>=1){
            st[x].first = min(st[x<<1].first, st[x<<1|1].first);
            st[x].second = max(st[x<<1].second, st[x<<1|1].second);
        }
  }
  inline pair<int,int> query(int l, int r) {
    pair<int,int> ans = {inf,-inf};
        if(r<l)return {0,0};
    for (l += n, r += n; l <= r; l = (l + 1) / 2, r = (r - 1) / 2) {
      if (l & 1){
                ans.first = min(ans.first, st[l].first);
                ans.second = max(ans.second, st[l].second);
            }
      if (~r & 1){
                ans.first = min(ans.first, st[r].first);
                ans.second = max(ans.second, st[r].second);
            }
    }
    return ans;
  }
};
#define lp (p << 1)
#define rp ((p << 1)|1)
#define SUM 0
#define MAX 1
template<typename S>
struct info{
    int l, r;
    S sum, max, lazy;
};
template<typename T, typename S>
struct segtree{
    info<S> *tree;
    T *a;
    int n, newed;
    segtree(int n, T* a=nullptr): a(a), n(n), newed(false){
        if(a == nullptr){
            this->a = new T[n+1];
            for(int i = 0; i <= n; ++i) this->a[i] = 0;
            newed = true;
        }
        tree = new info<S>[4*n+1];
        build(1, 1, n);
    }
    void pushup(int p){
        tree[p].sum = tree[lp].sum + tree[rp].sum;
        tree[p].max = max(tree[lp].max, tree[rp].max);
    }
    void pushdown(int p, int ln, int rn){
        if(tree[p].lazy){
            tree[lp].lazy += tree[p].lazy;
            tree[rp].lazy += tree[p].lazy;
            tree[lp].sum += (tree[p].lazy * ln);
            tree[rp].sum += (tree[p].lazy * rn);
            tree[lp].max += tree[p].lazy;
            tree[rp].max += tree[p].lazy;
            tree[p].lazy = 0;
        }
    }
    void build(int p, int l, int r){
        tree[p].l = l;
        tree[p].r = r;
        tree[p].lazy = 0;
        if(l == r){
            tree[p].sum = a[l];
            tree[p].max = a[l];
            return;
        }
        int mid = l + ((r - l) >> 1);
        build(lp, l, mid);
        build(rp, mid+1, r);
        pushup(p);
    }
    void update(int p, int pos, const T& d){
        int L = tree[p].l;
        int R = tree[p].r;
        if(L == pos && R == pos){
            tree[p].sum += d;
            tree[p].max += d;
            return;
        }
        int mid = L + ((R - L) >> 1);
        if(pos <= mid){
            update(lp, pos, d);
        }else{
            update(rp, pos, d);
        }
        pushup(p);
    }
    void update(int pos, const T& d){
        update(1, pos, d);
    }
    void update2(int p, int l, int r, const T& d){
        int L = tree[p].l;//L, □□□□□□□R
        int R = tree[p].r;
        //l, □□□□rquery
```

```cpp
    if(l <= L && r >= R){
        tree[p].lazy += d;
        tree[p].sum += ((S)(R - L + 1))*((S)d);
        tree[p].max += d;
        return;
    }
    int mid = L + ((R - L) >> 1);
    pushdown(p, mid - L + 1, R - mid);
    if(l <= mid){
        update2(lp, l, r, d);
    }
    if(r > mid){
        update2(rp, l, r, d);
    }
    pushup(p);
}
void update2(int l, int r, const T& d){
    update2(1, l, r, d);
}
void query(int p, int l, int r, S& s, S& m){
    int L = tree[p].l;//L, □□□□□□□R
    int R = tree[p].r;
    if(L >= l && R <= r){
            s = tree[p].sum;
            m = tree[p].max;
        return;
    }
    int mid = L + ((R - L) >> 1);
    pushdown(p, mid - L + 1, R - mid);
    s = 0; m = numeric_limits<S>::min();
    if(l <= mid){
        S s2 = 0, m2 = numeric_limits<S>::min();
        query(lp, l, r, s2, m2);
            s += s2;
            m = max(m, m2);
    }
    if(r > mid){
        S s2 = 0, m2 = numeric_limits<S>::min();
        query(rp, l, r, s2, m2);
            s += s2;
            m = max(m, m2);
    }
}
void query(int l, int r, S& s, S& m){
    query(1, l, r, s, m);
}
int querybig(int p, int l, int r, S target){
    int L = tree[p].l;
    int R = tree[p].r;
        if(R < l || L > r || tree[p].max < target) return 0;
        if(L == R){
            return L;
        }
        int mid = L + ((R - L) >> 1);
        pushdown(p, mid - L + 1, R - mid);
        int res = 0;
        if(l <= mid){
            res = querybig(lp, l, r, target);
        }
        if(res) return res;
        return querybig(rp, l, r, target);
    }
    int querybig(int l, int r, S target){
        return querybig(1, l, r, target);
    }
    ~segtree(){
        delete[] tree;
        if(newed) delete[] a;
    }
};
```

10.10 SegmentTreeBeats.cpp

Description: A segment tree with special queries, allows to update For all i in [l,r), change Ai to max/min(Ai, x) Query for the sum of Ai in [l, r]
Usage: `build(); //check that array is global`
`update_max/min(--l,r,x); // update is in a range [l,r) and 0 indexed`
Time: $\mathcal{O}(\log N)$ per query.

SegmentTreeBeats.cpp — 43edb5, 346 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
long long gcd(long long a, long long b) { // always positive
    a = abs(a);
    b = abs(b);
    while (a) {
        b %= a;
        swap(a, b);
    }
    return b;
}
struct STBeats {
    static const int T = (1 << 20);
    static const long long INF = 1e15 + 7;
    struct Node {
        long long min;
        int minCnt;
        long long secondMin;
        long long max;
        int maxCnt;
        long long secondMax;
        long long sum;
        long long diffGcd;
        long long pushSum;
        long long pushEq;
        // If we have pushEq, no other pushes should be made. They're all
            combined together in pushEq.
        // Otherwise we first apply pushSum and then min= and max= pushes
            (in any order btw).
    } tree[T];
    int n;
    void doPushEq(int v, int l, int r, long long val) {
        tree[v].min = tree[v].max = tree[v].pushEq = val;
        tree[v].minCnt = tree[v].maxCnt = r - l;
        tree[v].secondMax = -INF;
        tree[v].secondMin = INF;
        tree[v].sum = val * (r - l);
        tree[v].diffGcd = 0;
        tree[v].pushSum = 0;
    }
    void doPushMinEq(int v, int l, int r, long long val) {
        if (tree[v].min >= val) {
            doPushEq(v, l, r, val);
        } else if (tree[v].max > val) {
            if (tree[v].secondMin == tree[v].max) {
                tree[v].secondMin = val;
            }
            tree[v].sum -= (tree[v].max - val) * tree[v].maxCnt;
            tree[v].max = val;
        }
    }
    void doPushMaxEq(int v, int l, int r, long long val) {
        if (tree[v].max <= val) {
            doPushEq(v, l, r, val);
        } else if (tree[v].min < val) {
            if (tree[v].secondMax == tree[v].min) {
                tree[v].secondMax = val;
            }
            tree[v].sum += (val - tree[v].min) * tree[v].minCnt;
            tree[v].min = val;
        }
    }
    void doPushSum(int v, int l, int r, long long val) {
        if (tree[v].min == tree[v].max) {
            doPushEq(v, l, r, tree[v].min + val);
        } else {
            tree[v].max += val;
            if (tree[v].secondMax != -INF) {
                tree[v].secondMax += val;
```

```
        }
        tree[v].min += val;
        if (tree[v].secondMin != INF) {
            tree[v].secondMin += val;
        }
        tree[v].sum += (r - l) * val;
        tree[v].pushSum += val;
    }
}
void pushToChildren(int v, int l, int r) {
    if (l + 1 == r) {
        return;
    }
    int mid = (r + l) / 2;
    if (tree[v].pushEq != -1) {
        doPushEq(2 * v, l, mid, tree[v].pushEq);
        doPushEq(2 * v + 1, mid, r, tree[v].pushEq);
        tree[v].pushEq = -1;
    } else {
        doPushSum(2 * v, l, mid, tree[v].pushSum);
        doPushSum(2 * v + 1, mid, r, tree[v].pushSum);
        tree[v].pushSum = 0;
        doPushMinEq(2 * v, l, mid, tree[v].max);
        doPushMinEq(2 * v + 1, mid, r, tree[v].max);
        doPushMaxEq(2 * v, l, mid, tree[v].min);
        doPushMaxEq(2 * v + 1, mid, r, tree[v].min);
    }
}
void updateFromChildren(int v) {
    tree[v].sum = tree[2 * v].sum + tree[2 * v + 1].sum;
    tree[v].max = max(tree[2 * v].max, tree[2 * v + 1].max);
    tree[v].secondMax = max(tree[2 * v].secondMax, tree[2 * v + 1].
        ↪ secondMax);
    tree[v].maxCnt = 0;
    if (tree[2 * v].max == tree[v].max) {
        tree[v].maxCnt += tree[2 * v].maxCnt;
    } else {
        tree[v].secondMax = max(tree[v].secondMax, tree[2 * v].max);
    }
    if (tree[2 * v + 1].max == tree[v].max) {
        tree[v].maxCnt += tree[2 * v + 1].maxCnt;
    } else {
        tree[v].secondMax = max(tree[v].secondMax, tree[2 * v + 1].max)
            ↪ ;
    }
    tree[v].min = min(tree[2 * v].min, tree[2 * v + 1].min);
    tree[v].secondMin = min(tree[2 * v].secondMin, tree[2 * v + 1].
        ↪ secondMin);
    tree[v].minCnt = 0;
    if (tree[2 * v].min == tree[v].min) {
        tree[v].minCnt += tree[2 * v].minCnt;
    } else {
        tree[v].secondMin = min(tree[v].secondMin, tree[2 * v].min);
    }
    if (tree[2 * v + 1].min == tree[v].min) {
        tree[v].minCnt += tree[2 * v + 1].minCnt;
    } else {
        tree[v].secondMin = min(tree[v].secondMin, tree[2 * v + 1].min)
            ↪ ;
    }
    tree[v].diffGcd = gcd(tree[2 * v].diffGcd, tree[2 * v + 1].diffGcd
        ↪ );
    long long anyLeft = tree[2 * v].secondMax; // any value that's not
        ↪ equal to left child max and min
    long long anyRight = tree[2 * v + 1].secondMax; // any value that'
        ↪ s not equal to right child max and min
    if (anyLeft != -INF && anyLeft != tree[2 * v].min && anyRight != -
        ↪ INF && anyRight != tree[2 * v + 1].min) {
        tree[v].diffGcd = gcd(tree[v].diffGcd, anyLeft - anyRight); //
            ↪ connect two spanning trees
    }
    long long any = -1; // any value that's not equal to current
        ↪ vertex max and min
    if (anyLeft != -INF && anyLeft != tree[2 * v].min) {
        any = anyLeft;
    } else if (anyRight != -INF && anyRight != tree[2 * v + 1].min) {
        any = anyRight;
    }
    // adding all values that are not max and min anymore to diffGcd
    for (long long val : {tree[2 * v].min, tree[2 * v].max, tree[2 * v
        ↪ + 1].min, tree[2 * v + 1].max}) {
        if (val != tree[v].min && val != tree[v].max) {
            if (any != -1) {
                tree[v].diffGcd = gcd(tree[v].diffGcd, val - any);
            } else {
                any = val;
            }
        }
    }
}
void build(int v, int l, int r, const vector<int>& inputArray) {
    tree[v].pushSum = 0;
    tree[v].pushEq = -1;
    if (l + 1 == r) {
        tree[v].max = inputArray[l];
        tree[v].secondMax = -INF;
        tree[v].maxCnt = 1;
        tree[v].min = inputArray[l];
        tree[v].secondMin = INF;
        tree[v].minCnt = 1;
        tree[v].diffGcd = 0;
        tree[v].sum = inputArray[l];
    } else {
        int mid = (r + l) / 2;
        build(2 * v, l, mid, inputArray);
        build(2 * v + 1, mid, r, inputArray);
        updateFromChildren(v);
    }
}
void build(const vector<int>& inputArray) {
    n = inputArray.size();
    build(1, 0, n, inputArray);
}
void updateMinEq(int v, int l, int r, int ql, int qr, int val) {
    if (qr <= l || r <= ql || tree[v].max <= val) {
        return;
    }
    if (ql <= l && r <= qr && tree[v].secondMax < val) {
        doPushMinEq(v, l, r, val);
        return;
    }
    pushToChildren(v, l, r);
    int mid = (r + l) / 2;
    updateMinEq(2 * v, l, mid, ql, qr, val);
    updateMinEq(2 * v + 1, mid, r, ql, qr, val);
    updateFromChildren(v);
}
void updateMinEq(int ql, int qr, int val) {
    updateMinEq(1, 0, n, ql, qr, val);
}
void updateMaxEq(int v, int l, int r, int ql, int qr, int val) {
    if (qr <= l || r <= ql || tree[v].min >= val) {
        return;
    }
    if (ql <= l && r <= qr && tree[v].secondMin > val) {
        doPushMaxEq(v, l, r, val);
        return;
    }
    pushToChildren(v, l, r);
    int mid = (r + l) / 2;
    updateMaxEq(2 * v, l, mid, ql, qr, val);
    updateMaxEq(2 * v + 1, mid, r, ql, qr, val);
    updateFromChildren(v);
}
void updateMaxEq(int ql, int qr, int val) {
    updateMaxEq(1, 0, n, ql, qr, val);
}
```

```
void updateEq(int v, int l, int r, int ql, int qr, int val) {
   if (qr <= l || r <= ql) {
      return;
   }
   if (ql <= l && r <= qr) {
      doPushEq(v, l, r, val);
      return;
   }
   pushToChildren(v, l, r);
   int mid = (r + l) / 2;
   updateEq(2 * v, l, mid, ql, qr, val);
   updateEq(2 * v + 1, mid, r, ql, qr, val);
   updateFromChildren(v);
}
void updateEq(int ql, int qr, int val) {
   updateEq(1, 0, n, ql, qr, val);
}
void updatePlusEq(int v, int l, int r, int ql, int qr, int val) {
   if (qr <= l || r <= ql) {
      return;
   }
   if (ql <= l && r <= qr) {
      doPushSum(v, l, r, val);
      return;
   }
   pushToChildren(v, l, r);
   int mid = (r + l) / 2;
   updatePlusEq(2 * v, l, mid, ql, qr, val);
   updatePlusEq(2 * v + 1, mid, r, ql, qr, val);
   updateFromChildren(v);
}
void updatePlusEq(int ql, int qr, int val) {
   updatePlusEq(1, 0, n, ql, qr, val);
}
long long findSum(int v, int l, int r, int ql, int qr) {
   if (qr <= l || r <= ql) {
      return 0;
   }
   if (ql <= l && r <= qr) {
      return tree[v].sum;
   }
   pushToChildren(v, l, r);
   int mid = (r + l) / 2;
   return findSum(2 * v, l, mid, ql, qr) + findSum(2 * v + 1, mid, r,
      ↪ ql, qr);
}
long long findSum(int ql, int qr) {
   return findSum(1, 0, n, ql, qr);
}
long long findMin(int v, int l, int r, int ql, int qr) {
   if (qr <= l || r <= ql) {
      return INF;
   }
   if (ql <= l && r <= qr) {
      return tree[v].min;
   }
   pushToChildren(v, l, r);
   int mid = (r + l) / 2;
   return min(findMin(2 * v, l, mid, ql, qr), findMin(2 * v + 1, mid,
      ↪ r, ql, qr));
}
long long findMin(int ql, int qr) {
   return findMin(1, 0, n, ql, qr);
}
long long findMax(int v, int l, int r, int ql, int qr) {
   if (qr <= l || r <= ql) {
      return -INF;
   }
   if (ql <= l && r <= qr) {
      return tree[v].max;
   }
   pushToChildren(v, l, r);
   int mid = (r + l) / 2;
   return max(findMax(2 * v, l, mid, ql, qr), findMax(2 * v + 1, mid,
      ↪ r, ql, qr));
}
long long findMax(int ql, int qr) {
   return findMax(1, 0, n, ql, qr);
}
long long findGcd(int v, int l, int r, int ql, int qr) {
   if (qr <= l || r <= ql) {
      return 0;
   }
   if (ql <= l && r <= qr) {
      long long ans = tree[v].diffGcd;
      if (tree[v].secondMax != -INF) {
         ans = gcd(ans, tree[v].secondMax - tree[v].max);
      }
      if (tree[v].secondMin != INF) {
         ans = gcd(ans, tree[v].secondMin - tree[v].min);
      }
      ans = gcd(ans, tree[v].max);
      return ans;
   }
   pushToChildren(v, l, r);
   int mid = (r + l) / 2;
   return gcd(findGcd(2 * v, l, mid, ql, qr), findGcd(2 * v + 1, mid,
      ↪ r, ql, qr));
}
long long findGcd(int ql, int qr) {
   return findGcd(1, 0, n, ql, qr);
}
} segTree;
int main() {
   ios::sync_with_stdio(0);
   cin.tie(0);
   int n;
   cin >> n;
   vector<int> inputArray(n);
   for (int &val : inputArray) {
      cin >> val;
   }
   segTree.build(inputArray);
   int q;
   cin >> q;
   for (int i = 0; i < q; i++) {
      int type, ql, qr;
      cin >> type >> ql >> qr;
      ql--;
      if (type == 1) {
         long long k;
         cin >> k;
         segTree.updateMinEq(ql, qr, k);
      } else if (type == 2) {
         long long k;
         cin >> k;
         segTree.updateMaxEq(ql, qr, k);
      } else if (type == 3) {
         long long k;
         cin >> k;
         segTree.updateEq(ql, qr, k);
      } else if (type == 4) {
         long long k;
         cin >> k;
         segTree.updatePlusEq(ql, qr, k);
      } else if (type == 5) {
         cout << segTree.findSum(ql, qr) << '\n';
      } else if (type == 6) {
         cout << segTree.findMin(ql, qr) << '\n';
      } else if (type == 7) {
         cout << segTree.findMax(ql, qr) << '\n';
      } else {
         cout << segTree.findGcd(ql, qr) << '\n';
      }
   }
   return 0;
}
```

10.11 SegmentTreeDynamic.cpp

Description: A Segment Tree that stores data only if is needed or asked , that alows to manage bigger "arrays" more than 10^7
Usage: `Node st(0,maximum_size);`
Time: $\mathcal{O}(\log N)$ per query.

SegmentTreeDynamic.cpp 372f41, 49 lines

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

10.12 SegmentTreeDynamic2D.cpp

Description: A 2D segment tree with updates in a point
Usage: `ST t(0, n - 1);`
`t.upd(x, y, v); // update in a point(x,y) a value v (asigment)`
`t.query(x1, x2, y1, y2); // Returns an assosiative function in a submatrix`
Time: $\mathcal{O}(\log N)$ per query.

SegmentTreeDynamic2D.cpp ab213f, 143 lines

```
mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
const int N = 3e5 + 9;
struct node {
 node *l, *r;
 int pos, key, mn, mx;
 long long val, g;
 node(int position, long long value) {
  l = r = nullptr;
  mn = mx = pos = position;
  key = rnd();
  val = g = value;
 }
 void pull() {
  g = val;
  if(l) g = __gcd(g, l->g);
  if(r) g = __gcd(g, r->g);
  mn = (l ? l->mn : pos);
  mx = (r ? r->mx : pos);
 }
};
//memory O(n)
struct treap {
 node *root;
 treap() {
  root = nullptr;
 }
 void split(node *t, int pos, node *&l, node *&r) {
  if (t == nullptr) {
   l = r = nullptr;
   return;
  }
  if (t->pos < pos) {
   split(t->r, pos, l, r);
   t->r = l;
   l = t;
  } else {
   split(t->l, pos, l, r);
   t->l = r;
   r = t;
  }
  t->pull();
 }
 node* merge(node *l, node *r) {
  if (!l || !r) return l ? l : r;
  if (l->key < r->key) {
   l->r = merge(l->r, r);
   l->pull();
   return l;
  }
  r->l = merge(l, r->l);
  r->pull();
  return r;
 }
 bool find(int pos) {
  node *t = root;
  while (t) {
   if (t->pos == pos) return true;
   if (t->pos > pos) t = t->l;
   else t = t->r;
  }
  return false;
 }
 void upd(node *t, int pos, long long val) {
  if (t->pos == pos) {
   t->val = val;
   t->pull();
   return;
  }
  if (t->pos > pos) upd(t->l, pos, val);
  else upd(t->r, pos, val);
  t->pull();
 }
 void insert(int pos, long long val) { //set a_pos = val
  if (find(pos)) upd(root, pos, val);
  else {
   node *l, *r;
   split(root, pos, l, r);
   root = merge(merge(l, new node(pos, val)), r);
  }
 }
 long long query(node *t, int st, int en) {
  if (t->mx < st || en < t->mn) return 0;
  if (st ≤ t->mn && t->mx ≤ en) return t->g;
  long long ans = (st ≤ t->pos && t->pos ≤ en ? t->val : 0);
```

```cpp
  if (t->l) ans = __gcd(ans, query(t->l, st, en));
  if (t->r) ans = __gcd(ans, query(t->r, st, en));
  return ans;
 }
 long long query(int l, int r) { //gcd of a_i such that l <= i <= r
  if (!root) return 0;
  return query(root, l, r);
 }
 void print(node *t) {
  if (!t) return;
  print(t->l);
  cout << t->val << " ";
  print(t->r);
 }
};
//total memory along with treap = nlogn
struct ST {
 ST *l, *r;
 treap t;
 int b, e;
 ST() {
  l = r = nullptr;
 }
 ST(int st, int en) {
  l = r = nullptr;
  b = st, e = en;
 }
 void fix(int pos) {
  long long val = 0;
  if (l) val = __gcd(val, l->t.query(pos, pos));
  if (r) val = __gcd(val, r->t.query(pos, pos));
  t.insert(pos, val);
 }
 void upd(int x, int y, long long val) { //set a[x][y] = val
  if (e < x || x < b) return;
  if (b == e) {
   t.insert(y, val);
   return;
  }
  if (b != e) {
   if (x <= (b + e) / 2) {
    if (!l) l = new ST(b, (b + e) / 2);
    l->upd(x, y, val);
   } else {
    if (!r) r = new ST((b + e) / 2 + 1, e);
    r->upd(x, y, val);
   }
  }
  fix(y);
 }
 long long query(int i, int j, int st, int en) { //gcd of a[x][y] such that i <= x <= j && st <= y <= en
  if (e < i || j < b) return 0;
  if (i <= b && e <= j) return t.query(st, en);
  long long ans = 0;
  if (l) ans = __gcd(ans, l->query(i, j, st, en));
  if (r) ans = __gcd(ans, r->query(i, j, st, en));
  return ans;
 }
};
```

10.13 SegmentTreeDynamicOpt.cpp

Description: An optimized in memory dynamic segment tree
Time: $\mathcal{O}(\log N)$ per query.

SegmentTreeDynamicOpt.cpp dbbd38, 63 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
struct node{
   int sum,idl,idr;
   node(int l,int r,int _idl = -1,int _idr = -1):sum((r-l)+1),idl(_idl), idr(_idr){}
};
struct SegmentTree{
   const int l,r;
   vector<node> tree;
   SegmentTree(int low,int h):l(low),r(h){
      tree.reserve((1u << 21) + (1u << 20));
      tree.push_back(node(l,r));
   }
   void extend(int id,int a,int b){
      if(tree[id].idl == -1){
         int m = (a+b)/2;
         tree.push_back(node(a,m));
         tree[id].idl = tree.size()-1;
         tree.push_back(node(m+1,b));
         tree[id].idr = tree.size()-1;
      }
   }
   void update(int id ,int pos,int val,int a ,int b ){
      if(pos<a || pos >b)return ;
      if(a == b){
         tree[id].sum += val;
         return;
      }
      extend(id,a,b);
      int m = (a+b)/2;
      const int idl = tree[id].idl; assert(idl != -1);
      const int idr = tree[id].idr; assert(idr != -1);
      update(idl,pos,val,a,m);
      update(idr,pos,val,m+1,b);
      tree[id].sum = tree[idl].sum+tree[idr].sum;
   }
   int query(int id,int k,int a ,int b ){
      if(a == b)return a;
      extend(id,a,b);
      const int idl = tree[id].idl; assert(idl != -1);
      const int idr = tree[id].idr; assert(idr != -1);
      int m = (a+b)>>1;
      if(tree[idl].sum>= k)
         return query(idl,k,a,m);
      else
         return query(idr,k-tree[idl].sum,m+1,b);
   }
};
int main() {
   int n, m,q;
   char t;
   cin>>n>>m;
   SegmentTree st(1,n);
   for(int i = 0;i<m;i++){
      cin>>t>>q;
      int x = st.query(0,q,1,n);
      if (t == 'L')
         cout<<x<<endl;
      else
         st.update(0,x, -1,1,n);
   }
   return 0;
}
```

10.14 SegmentTreePersistent.cpp

Description: A persistent data structure is a data structure that remembers it previous state for each modification. This allows to access any version of this data structure that interest us and execute a query on it.
Time: $\mathcal{O}(\log N)$ per query.

SegmentTreePersistent.cpp 68598c, 190 lines

```cpp
const int maxn = 800007;
int L[maxn],R[maxn],st[maxn],lazy[maxn],N;
int n; /** Must be global for default values in functions */
int newLeaf(int val){
   int p = ++N;
   L[p] = R[p] = 0;
   st[p] = val;
   return p;
}
int newParent(int l,int r){
```

```cpp
    int p = ++N;
    L[p] = l;
    R[p] = r;
    st[p] = st[l]+st[r];
    return p;
}
int newLazy(int v,int val,int l,int r){
    int p = ++N;
    L[p] = L[v];
    R[p] = R[v];
    lazy[p] += val;
    st[p] = st[v]+((r-l)+1)*val;
    return p;
}
int build(vector<int> &A,int l = 0,int r = n-1){
    if(l== r)return newLeaf(A[l]);
    int mid = (l+r)>>1;
    return newParent(build(A,l,mid),build(A,mid+1,r));
}
void propagate(int p,int l,int r){
    if(lazy[p]==0)return;
    if(l!= r){
        int mid = (l+r)>>1;
        L[p] = newLazy(L[p],lazy[p],l,mid);
        R[p] = newLazy(R[p],lazy[p],mid+1,r);
    }
    lazy[p] = 0;
}
int update(int l,int r,int val,int p,int sl = 0 ,int sr = n-1){
    if(sr<l || sl>r)return p;
    if(sl>=l && sr<=r)return newLazy(p,val,sl,sr);
    propagate(p,sl,sr);
    int mid = (sl+sr)>>1;
    return newParent(update(l,r,val,L[p],sl,mid),update(l,r,val,R[p],mid+1,sr));
}
int query(int l,int r,int p,int sl = 0,int sr = n-1){
    if(sr<l || sl> r)return 0;
    if(sl>=l && sr<=r)return st[p];
    int mid = (sl+sr)>>1;
    propagate(p,sl,sr);
    return query(l,r,L[p],sl,mid)+query(l,r,R[p],mid+1,r);
}
// -------------------
struct Vertex {
    Vertex *l, *r;
    ll sum;
    Vertex(int val) : l(nullptr), r(nullptr), sum(val) {}
    Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
        if (l) sum += l -> sum;
        if (r) sum += r -> sum;
    }
};
struct persistentSegmentTree {
    int n;
    Vertex* build(vll &nums, int tl, int tr) {
        if (tl == tr) return new Vertex(nums[tl]);
        int mid = (tl + tr) / 2;
        return new Vertex(build(nums,tl,mid),build(nums,mid+1,tr));
    }
    Vertex *build(vll &nums){
        n = nums.size();
        return build(nums,0,n-1);
    }
    int query(Vertex *v, int l, int r) {
        return query(v,0,n-1,l,r);
    }
    ll query(Vertex *v, int tl, int tr, int l, int r) {
        if (tl > r || tr < l) return 0;
        if (tl >= l && tr <= r) return v -> sum;
        int mid = (tl + tr) / 2;
        return query(v -> l, tl,mid,l,r) + query(v -> r,mid+1,tr,l,r);
    }
    Vertex* update(Vertex *v, int tl, int tr, int pos, int val) {
        if (tl == tr) return new Vertex(val);
        int mid = (tl + tr) / 2;
        if (pos <= mid) return new Vertex(update(v -> l,tl,mid,pos,val), v -> r);
        return new Vertex(v -> l, update(v -> r, mid + 1, tr, pos ,val));
    }
    Vertex* update(Vertex *v, int pos, ll val) {
        return update(v,0,n-1,pos,val);
    }
    int find_kth(Vertex* vl, Vertex *vr, int tl, int tr, int k) {
        if (tl == tr)
            return tl;
        int tm = (tl + tr) / 2, left_count = vr->l->sum - vl->l->sum;
        if (left_count >= k)
            return find_kth(vl->l, vr->l, tl, tm, k);
        return find_kth(vl->r, vr->r, tm+1, tr, k-left_count);
    }
};
// kth sum in range
struct Node {
  int sum;
  int cnt;
  int L, R;
};
vector<int> sum;
vector<int> cnt;
vector<int> LCH;
vector<int> RCH;
int newNode(int val,int c,int l = 0,int r = 0){
  sum.push_back(val);
  cnt.push_back(c);
  LCH.push_back(l);
  RCH.push_back(r);
  return sum.size()-1;
}
int mn, mx;
vector<int> roots;
int update(int v, int tl, int tr, int idx) {
  if (tl == tr) {
    return newNode(sum[v]+tl,cnt[v]+1);
  }
  int tm = tl + (tr - tl) / 2;
  int L = LCH[v];
  int R = RCH[v];
  if (idx <= tm)
    L = update(L, tl, tm, idx);
  else
    R = update(R, tm + 1, tr, idx);
  return newNode(sum[L]+sum[R],cnt[L]+cnt[R],L,R);
}
void init(const vector<int>& arr){
  roots.clear();
  sum.clear();
  cnt.clear();
  LCH.clear();
  RCH.clear();
  mn = 0;
  mx = 0;
  roots.resize(arr.size()+1);
  newNode(0,0);
  for (int val : arr) mn = min(mn, val), mx = max(mx, val);
  for (int i = 0; i < (int)arr.size(); i++)
    roots[i + 1] = update(roots[i], mn, mx, arr[i]);
}
/* find kth smallest/greatest number among arr[l], arr[l+1], ... , arr[r]
 * k is 1-based, so find_kth(l,r,1) returns the min
 */
int query(int vl, int vr, int tl, int tr, int k,bool mx){
  if (tl == tr)
    return tl;
  int tm = tl + (tr - tl) / 2;
  int lL = LCH[vl], lR = RCH[vl], rL = LCH[vr],rR = RCH[vr];
    //^ first l is for range, L is Left child , same for r,
```

```
  if(mx){
    swap(lL,lR),swap(rL,rR);
  }
  int left_count = cnt[rL] - cnt[lL];
  if (left_count >= k) return query(lL, rL,mx?tm+1: tl, mx?tr:tm, k ,mx);
  return query(lR, rR, mx?tl:tm + 1, mx?tm:tr, k - left_count,mx);
}
// Call this helper function
int query(int l, int r, int k,bool max_num = false){
  assert(1 <= k && k <= r - l + 1);
  assert(0 <= l && r + 1 < (int)roots.size());
  return query(roots[l], roots[r + 1], mn, mx, k,max_num);
}
/* find **sum** of k smallest/greatest numbers among arr[l], arr[l+1],
     ↪ ... , arr[r]
 * k is 1-based, so find_kth(l,r,1) returns the min
 */
int query_sum(int vl, int vr, int tl, int tr, int k,bool mx){
  if (tl == tr)
    return tl * k;
  int tm = (tl+tr)>>1;
  int lL = LCH[vl], lR = RCH[vl], rL = LCH[vr],rR = RCH[vr];
  if(mx){
    swap(lL,lR),swap(rL,rR);
  }
  int left_count = cnt[rL] - cnt[lL];
  int left_sum = sum[rL] - sum[lL];
  if (left_count >= k) return query_sum(lL, rL,mx?tm+1:tl, mx?tr:tm, k,mx
     ↪ );
  return left_sum + query_sum(lR, rR, mx?tl:tm + 1, mx?tm:tr, k -
     ↪ left_count,mx);
}
// Call this helper function
int query_sum(int l, int r, int k,bool max_sum = false){
  assert(1 <= k && k <= r - l + 1); //note this condition implies L <= R
  assert(0 <= l && r + 1 < (int)roots.size());
  return query_sum(roots[l], roots[r + 1], mn, mx, k,max_sum);
}
```

10.15 SparseTable.cpp

Description: Sparse table is similar to segment tree but don´t allows updates

Time: $\mathcal{O}(1)$ per query, $\mathcal{O}(N \log N)$ build

SparseTable.cpp 92dab9, 38 lines

```
#define MAXN 1000000
#define MAXPOWN 1048576  // 2^(ceil(log_2(MAXN)))
#define MAXLEV 21   // ceil(log_2(MAXN)) + 1
int n, P, Q;
int A[MAXPOWN];
int table[MAXLEV][MAXPOWN];
int maxlev, siz
size = n;
maxlev = __builtin_clz(n) ^31;  // floor(log_2(n))
if( (1<<maxlev) != n)
  size = 1<<++maxlev;
void build(int level=0,int l=0, int r=size){
 int m = (l+r)/2;
 table[level][m] = A[m]%P;
 for(int i=m-1;i>=l;i--)
   table[level][i] = (long long)table[level][i+1] * A[i] % P;
 if(m+1 < r) {
   table[level][m+1] = A[m+1]%P;
   for(int i=m+2;i<r;i++)
     table[level][i] = (long long)table[level][i-1] * A[i] % P;
 }
 if(l + 1 != r)  // r - l > 1
 {
   build(level+1, l, m);
   build(level+1, m, r);
 }
}
int query(int x, int y)
{
 if(x == y)
   return A[x]%P;
 int k2 = __builtin_clz(x^y) ^31;
 int lev = maxlev - 1 - k2;
 int ans = table[lev][x];
 if(y & ((1<<k2) - 1)) // y % (1<<k2)
   ans = (long long)ans * table[lev][y] % P;
 return ans;
}
```

10.16 SubMatrix.cpp

Description: Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).

Usage: prefixSums(matrix);
sum2D(0, 0, 2, 2); // top left 4 elements

Time: $\mathcal{O}\left(N^2 + Q\right)$

SubMatrix.cpp 304c8e, 32 lines

```
vector<vector<long long>> sum;
void prefixSums(vector<vector<long long>> M){
    int n = M.size();
    int m = M[0].size();
    sum.assign(n,vector<int> (m,0));
    sum[0][0] = M[0][0];
    for(int i = 1;i<n;i++)
        sum[i][0] = sum[i-1][0]+ M[i][0];
    for(int i = 1;i<m;i++)
        sum[0][i] = sum[0][i-1] + M[0][i];
    for(int i = 1;i<n;i++){
        for(int j = 1;j<m;j++){
            sum[i][j] = sum[i-1][j]+sum[i][j-1]-sum[i-1][j-1]+ M[i][j];
        }
    }
    for(int i = 0;i<n;i++){
        for(int j = 0;j<m;j++){
            cout<<sum[i][j]<<" ";
        }
        cout<<endl;
    }
}
// Upper left corner and bottom right corner inclusive
int sum2D(int x0,int y0,int x1,int y1){
    int u = 0;
    int d = 0;
    int l = 0;
    if(x0 && y0)d = sum[x0-1][y0-1];
    if(y0) l = sum[x1][y0-1];
    if(x0) u = sum[x0-1][y1];
    return sum[x1][y1]-l-u+d;
}
```

10.17 moHilbert.cpp

Description: Mo's algorithm with Hilbert order and without Hilbert order, mo's algorithm is a technique to solve range queries in offline mode.

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

moHilbert.cpp 1cd133, 106 lines

```
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define int long long
#define __ ios_base::sync_with_stdio(false);cin.tie(NULL);
inline int gilbertOrder(int x, int y, int pow, int rotate) {
  if (pow == 0){
    return 0;
  }
  int hpow = 1 << (pow-1);
  int seg = (x < hpow) ? (
    (y < hpow) ? 0 : 3
  ) : (
    (y < hpow) ? 1 : 2
  );
  seg = (seg + rotate) & 3;
  const int rotateDelta[4] = {3, 0, 0, 1};
  int nx = x & (x ^hpow), ny = y & (y ^hpow);
  int nrot = (rotate + rotateDelta[seg]) & 3;
```

```cpp
  int subSquareSize = 1ll << (2*pow - 2);
  int ans = seg * subSquareSize;
  int add = gilbertOrder(nx, ny, pow-1, nrot);
  ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
  return ans;
}
struct query{
  int l,r,id;int ord;
  inline void calcOrder() {
    ord = gilbertOrder(l, r, 21, 0);
  }
};
inline bool operator<(const query &a, const query &b) {
  return a.ord < b.ord;
}
// Without hilbert order.
// Around 4 times slower , depend of number of queries
// int block = 316;
// struct query {
//  int l, r, id;
//  inline pair<int, int> toPair() const {
//   return make_pair(l / block, ((l / block) & 1) ? -r : +r);
//  }
// };
// inline bool operator<(const query &a, const query &b) {
//  return a.toPair() < b.toPair();
// }
int k = 0;
int total = 0;
int mp[1<<20];
void add(int x){
    total+=mp[x^k];
    mp[x]++;
}
void rem(int x){
    mp[x]--;
    total-=mp[x^k];
}
signed main(){__
    int T = 1,n,m,l,r;
    while(T--){
        cin>>n>>m>>k;
        vector<int> nums(n);
        for(auto &c:nums)cin>>c;
        vector<int> pref(n+1);
        for(int i = 1;i<=n;i++){
            pref[i] = pref[i-1]^nums[i-1];
        }
        vector<query> Q;
        for(int id = 0;id<m;id++){
            cin>>l>>r;
            l--;
            // Consider manage a range (l,r] if the problem work with prefix , this to be able to delete the contribution of prefix_l
            Q.push_back({l,r,id});
            // if you use hilbert order must call calcOrder() for each query
            Q.back().calcOrder();
        }
        sort(Q.begin(),Q.end());
        int L = 0,R = -1;
        vector<int> ans(m);
        for(int i = 0;i<m;i++){
            int l = Q[i].l;
            int r = Q[i].r;
            while(L>l){
                L--;
                add(pref[L]);
            }
            while(R<r){
                R++;
                add(pref[R]);
            }
            while(L<l){
                rem(pref[L]);
                L++;
            }
            while(R>r){
                rem(pref[R]);
                R--;
            }
            ans[Q[i].id] = total;
        }
        for(int i = 0;i<m;i++)
            cout<<ans[i]<<endl;
        cout<<endl;
    }
    return 0;
}
```

10.18 randomizedKdTree.cpp

randomizedKdTree.cpp 192 lines

```cpp
typedef complex<double> point;
struct randomized_kd_tree
{
  struct node
  {
    point p;
    int d, s;
    node *l, *r;
    bool is_left_of(node *x)
    {
      if (x->d)
        return real(p) < real(x->p);
      else
        return imag(p) < imag(x->p);
    }
  }*root;
  randomized_kd_tree() : root(0) {}
  int size(node *t)
  {
    return t ? t->s : 0;
  }
  node *update(node *t)
  {
    t->s = 1 + size(t->l) + size(t->r);
    return t;
  }
  pair<node*, node*> split(node *t, node *x)
  {
    if (!t)
      return {0, 0};
    if (t->d == x->d)
    {
      if (t->is_left_of(x))
      {
        auto p = split(t->r, x);
        t->r = p.first;
        return {update(t), p.second};
      }
      else
      {
        auto p = split(t->l, x);
        t->l = p.second;
        return {p.first, update(t)};
      }
    }
    else
    {
      auto l = split(t->l, x);
      auto r = split(t->r, x);
      if (t->is_left_of(x))
      {
        t->l = l.first;
        t->r = r.first;
```

```
        return {update(t), join(l.second, r.second, t->d)};
      }
      else
      {
        t->l = l.second;
        t->r = r.second;
        return {join(l.first, r.first, t->d), update(t)};
      }
    }
  }
  node *join(node *l, node *r, int d)
  {
    if (!l)
      return r;
    if (!r)
      return l;
    if (rand() % (size(l) + size(r)) < size(l))
    {
      if (l->d == d)
      {
        l->r = join(l->r, r, d);
        return update(l);
      }
      else
      {
        auto p = split(r, l);
        l->l = join(l->l, p.first, d);
        l->r = join(l->r, p.second, d);
        return update(l);
      }
    }
    else
    {
      if (r->d == d)
      {
        r->l = join(l, r->l, d);
        return update(r);
      }
      else
      {
        auto p = split(l, r);
        r->l = join(p.first, r->l, d);
        r->r = join(p.second, r->r, d);
        return update(r);
      }
    }
  }
  node *insert(node *t, node *x)
  {
    if (rand() % (size(t) + 1) == 0)
    {
      auto p = split(t, x);
      x->l = p.first;
      x->r = p.second;
      return update(x);
    }
    else
    {
      if (x->is_left_of(t))
        t->l = insert(t->l, x);
      else
        t->r = insert(t->r, x);
      return update(t);
    }
  }
  void insert(point p)
  {
    root = insert(root, new node({ p, rand() % 2 }));
  }
  node *remove(node *t, node *x)
  {
    if (!t)
      return t;
    if (t->p == x->p)
      return join(t->l, t->r, t->d);
    if (x->is_left_of(t))
      t->l = remove(t->l, x);
    else
      t->r = remove(t->r, x);
    return update(t);
  }
  void remove(point p)
  {
    node n = { p };
    root = remove(root, &n);
  }
  void closest(node *t, point p, pair<double, node*> &ub)
  {
    if (!t)
      return;
    double r = norm(t->p - p);
    if (r < ub.first)
      ub = {r, t};
    node *first = t->r, *second = t->l;
    double w = t->d ? real(p - t->p) : imag(p - t->p);
    if (w < 0)
      swap(first, second);
    closest(first, p, ub);
    if (ub.first > w * w)
      closest(second, p, ub);
  }
  point closest(point p)
  {
    pair<double, node*> ub(1.0 / 0.0, 0);
    closest(root, p, ub);
    return ub.second->p;
  }
  // verification
  int height(node *n)
  {
    return n ? 1 + max(height(n->l), height(n->r)) : 0;
  }
  int height()
  {
    return height(root);
  }
  int size_rec(node *n)
  {
    return n ? 1 + size_rec(n->l) + size_rec(n->r) : 0;
  }
  int size_rec()
  {
    return size_rec(root);
  }
  void display(node *n, int tab = 0)
  {
    if (!n)
      return;
    display(n->l, tab + 2);
    for (int i = 0; i < tab; ++i)
      cout << " ";
    cout << n->p << " (" << n->d << ")" << endl;
    display(n->r, tab + 2);
  }
  void display()
  {
    display(root);
  }
};
```

10.19 splayTree.cpp

Description: A self balanced tree that allows mantain a full dinamyc array and get information in ranges
Time: $\mathcal{O}(logn)$ per query, $\mathcal{O}(N \log N)$ build

splayTree.cpp b49533, 218 lines

```
#include <bits/stdc++.h>
namespace allocator {
```

```cpp
// Array allocator.
template <class T, int MAXSIZE>
struct array {
  T v[MAXSIZE], *top;
  array() : top(v) {}
  T *alloc(const T &val = T()) { return &(*top++ = val); }
  void dealloc(T *p) {}
};
// Stack-based array allocator.
template <class T, int MAXSIZE>
struct stack {
  T v[MAXSIZE];
  T *spot[MAXSIZE], **top;
  stack() {
    for (int i = 0; i < MAXSIZE; ++i) spot[i] = v + i;
    top = spot + MAXSIZE;
  }
  T *alloc(const T &val = T()) { return &(**--top = val); }
  void dealloc(T *p) { *top++ = p; }
};
}  // namespace allocator
namespace splay {
// Abstract node struct.
template <class T>
struct node {
  T *f, *c[2];
  int size;
  node() {
    f = c[0] = c[1] = nullptr;
    size = 1;
  }
  void push_down() {}
  void update() {
    size = 1;
    for (int t = 0; t < 2; ++t)
      if (c[t]) size += c[t]->size;
  }
};
// Abstract reversible node struct.
template <class T>
struct reversible_node : node<T> {
  int r;
  reversible_node() : node<T>() { r = 0; }
  void push_down() {
    node<T>::push_down();
    if (r) {
      for (int t = 0; t < 2; ++t)
        if (node<T>::c[t]) node<T>::c[t]->reverse();
      r = 0;
    }
  }
  void update() { node<T>::update(); }
  // Reverse the range of this node.
  void reverse() {
    std::swap(node<T>::c[0], node<T>::c[1]);
    r = r ^ 1;
  }
};
template <class T, int MAXSIZE = 500000,
          class alloc = allocator::array<T, MAXSIZE + 2>>
struct tree {
  alloc pool;
  T *root;
  // Get a new node from the pool.
  T *new_node(const T &val = T()) { return pool.alloc(val); }
  tree() {
    root = new_node(), root->c[1] = new_node(), root->size = 2;
    root->c[1]->f = root;
  }
  // Helper function to rotate node.
  void rotate(T *n) {
    int v = n->f->c[0] == n;
    T *p = n->f, *m = n->c[v];
    if (p->f) p->f->c[p->f->c[1] == p] = n;
    n->f = p->f, n->c[v] = p;
    p->f = n, p->c[v ^ 1] = m;
    if (m) m->f = p;
    p->update(), n->update();
  }
  // Splay n so that it is under s (or to root if s is null).
  void splay(T *n, T *s = nullptr) {
    while (n->f != s) {
      T *m = n->f, *l = m->f;
      if (l == s)
        rotate(n);
      else if ((l->c[0] == m) == (m->c[0] == n))
        rotate(m), rotate(n);
      else
        rotate(n), rotate(n);
    }
    if (!s) root = n;
  }
  // Get the size of the tree.
  int size() { return root->size - 2; }
  // Helper function to walk down the tree.
  int walk(T *n, int &v, int &pos) {
    n->push_down();
    int s = n->c[0] ? n->c[0]->size : 0;
    (v = s < pos) && (pos -= s + 1);
    return s;
  }
  // Insert node n to position pos.
  void insert(T *n, int pos) {
    T *c = root;
    int v;
    ++pos;
    while (walk(c, v, pos), c->c[v] && (c = c->c[v]))
      ;
    c->c[v] = n, n->f = c, splay(n);
  }
  // Find the node at position pos. If sp is true, splay it.
  T *find(int pos, int sp = true) {
    T *c = root;
    int v;
    ++pos;
    while ((pos < walk(c, v, pos) || v) && (c = c->c[v]))
      ;
    if (sp) splay(c);
    return c;
  }
  // Find the range [posl, posr) on the splay tree.
  T *find_range(int posl, int posr) {
    T *r = find(posr), *l = find(posl - 1, false);
    splay(l, r);
    if (l->c[1]) l->c[1]->push_down();
    return l->c[1];
  }
  // Insert nn of size nn_size to position pos.
  void insert_range(T **nn, int nn_size, int pos) {
    T *r = find(pos), *l = find(pos - 1, false), *c = l;
    splay(l, r);
    for (int i = 0; i < nn_size; ++i) c->c[1] = nn[i], nn[i]->f = c, c = nn[i];
    for (int i = nn_size - 1; i >= 0; --i) nn[i]->update();
    l->update(), r->update(), splay(nn[nn_size - 1]);
  }
  // Helper function to dealloc a subtree.
  void dealloc(T *n) {
    if (!n) return;
    dealloc(n->c[0]);
    dealloc(n->c[1]);
    pool.dealloc(n);
  }
  // Remove from position [posl, posr).
  void erase_range(int posl, int posr) {
    T *n = find_range(posl, posr);
```

```
    n->f->c[1] = nullptr, n->f->update(), n->f->f->update(), n->f = nullptr;
    dealloc(n);
  }
};
} // namespace splay
const int MAXSIZE = 200000;
struct node: splay::reversible_node<node> {
  long long val, val_min, label_add;
  node(long long v = 0) : splay::reversible_node<node>(), val(v) { val_min = label_add = 0; }
  // Add v to the subtree.
  void add(long long v) {
    val += v;
    val_min += v;
    label_add += v;
  }
  void push_down() {
    splay::reversible_node<node>::push_down();
    for (int t = 0; t < 2; ++t) if (c[t]) c[t]->add(label_add);
    label_add = 0;
  }
  void update() {
    splay::reversible_node<node>::update();
    val_min = val;
    for (int t = 0; t < 2; ++t) if (c[t]) val_min = std::min(val_min, c[t]->val_min);
  }
};
splay::tree<node, MAXSIZE, allocator::stack<node, MAXSIZE + 2>> t;
int main() {
  int N, M;
  scanf("%d", &N);
  while (N--) {
    long long u; scanf("%lld", &u);
    t.insert(t.new_node(node(u)), t.size());
  }
  scanf("%d", &M);
  while (M--) {
    char c[10];
    scanf("%s", c);
    if (strcmp(c, "ADD") == 0) {
      int x, y; long long D;
      scanf("%d%d%lld", &x, &y, &D);
      t.find_range(x - 1, y)->add(D);
    } else if (strcmp(c, "REVERSE") == 0) {
      int x, y;
      scanf("%d%d", &x, &y);
      t.find_range(x - 1, y)->reverse();
    } else if (strcmp(c, "REVOLVE") == 0) {
      int x, y; long long T;
      scanf("%d%d%lld", &x, &y, &T);
      T %= (y - x + 1);
      if (T > 0) {
        // swap [x - 1, y - T) and [y - T, y)
        node *right = t.find_range(y - T, y);
        right->f->c[1] = nullptr, right->f->update(), right->f->f->update(), right->f = nullptr;
        t.insert(right, x - 1);
      }
    } else if (strcmp(c, "INSERT") == 0) {
      int x; long long P;
      scanf("%d%lld", &x, &P);
      t.insert(t.new_node(node(P)), x);
    } else if (strcmp(c, "DELETE") == 0) {
      int x;
      scanf("%d", &x);
      t.erase_range(x - 1, x);
    } else if (strcmp(c, "MIN") == 0) {
      int x, y;
      scanf("%d%d", &x, &y);
      printf("%lld\n", t.find_range(x - 1, y)->val_min);
    }
  }
}
```

10.20 treap.cpp

Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data. this version is a full binary search tree different from cp algorithms, this allows to get other usefull data like order of key kth sum etc.
Time: $\mathcal{O}(\log N)$

treap.cpp 6471dd, 156 lines

```
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
// #define int long long
#define ll long long
#define __ ios_base::sync_with_stdio(false);cin.tie(NULL);
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
int rand(int l, int r){
  uniform_int_distribution<int> ludo(l, r); return ludo(rng);
}
#define PT Treap *
struct Treap {
  int pr,key,sz;
   ll sum;
  Treap *l = NULL,*r = NULL;
   Treap(int key):key(key),pr(rand(0,1e9)),sz(1),l(0),r(0),sum(key) {}
};
PT root = NULL;
int sz(PT T){return T?T->sz:0;}
ll sum(PT T){return T?T->sum:0ll;}
void update(PT T){
    if(!T)return ;
    T->sz=sz(T->l)+sz(T->r)+1;
    T->sum=sum(T->l)+sum(T->r)+T->key;
}
void split(PT T, int key, PT& l, PT& r){ // l: <= key, r: > key
  if(!T)l=r=NULL;
  else if(T->key<=key)split(T->r,key,T->r,r),l=T;
  else split(T->l,key,l,T->l),r=T;
  update(T);
}
void insert(PT& T,PT v){
  if(!T)T=v;
    else if(v->pr>T->pr)split(T,v->key,v->l,v->r),T=v;
    else insert(v->key<=T->key?T->l:T->r,v);
  update(T);
}
void insert(int key){
    insert(root,new Treap(key));
}
void merge(PT& T, PT l, PT r){
  if(!l||!r)T=l?l:r;
  else if(l->pr>r->pr)merge(l->r,l->r,r),T=l;
  else merge(r->l,l,r->l),T=r;
  update(T);
}
void erase(PT& T, int key){
  if(T->key==key)merge(T,T->l,T->r);
  else erase(key<T->key?T->l:T->r,key);
  update(T);
}
void print(PT T){
    if(!T)return;
    print(T->l);
    cout<<"("<<T->key<<","<<T->sz<<","<<T->sum<<") ";
    print(T->r);
    update(T);
}
// Number of elements less or equal than x
int order_of_key(PT T,int x,int cont = 0){
    if(!T)return cont;
    if(T->key == x)return cont+sz(T->l)+1;
    if(x<T->key)return order_of_key(T->l,x,cont);
    return order_of_key(T->r,x,sz(T->l)+cont+1);
```

```
}
// Get the sum of all y<=x
// /- Seems that not work -/
int sum_to_key(PT T,int x,int s = 0){
   if(!T)return s;
   if(T->key == x)return sum(T->l)+s+x;
   if(x<T->key)return sum_to_key(T->l,x,s);
   return sum_to_key(T->r,x,sum(T->l)+s+T->key);
}
// Get the kth element, one indexed, the order is non decreseal
int getKth(PT T,int idx){
   if(!T)return -1;
   if(idx == 0 || sz(T->l)+sz(T->r)==0)return T->key;
   if(sz(T->l)+1 == idx)return T->key;
   if(sz(T->l)>=idx)return getKth(T->l,idx);
   return getKth(T->r,idx-(sz(T->l)+1));
}
// get the sum of the fist kth elements
int getKthSum(PT T,int idx,int s = 0){
   if(!T)return s;
   if(idx == 0 || sz(T->l)+sz(T->r)==0)return s+T->key;
   if(sz(T->l)+1 == idx)return T->key +s+sum(T->l);
   if(sz(T->l)>=idx)return getKthSum(T->l,idx,s);
   return getKthSum(T->r,idx-(sz(T->l)+1),s+sum(T->l)+T->key);
}
// Return the maximum k such sum of kth smallest elements is les or equal than x
int getCountUntilSum(Treap *T ,int x){
  if(!T)return 0;
   if(T->l && T->l->sum >x)return getCountUntilSum(T->l,x);
   int ans = sz(T->l);
   if(sum(T->l)+T->key <=x)ans +=1+getCountUntilSum(T->r,x-(sum(T->l)+T->key));
   return ans;
}
void clean(PT T){
   if(!T)return;
   clean(T->l);
   clean(T->r);
   delete T;
}
void clean(){
   clean(root);
   root = NULL;
}
// Ignore from here
signed main(){__
   int T = 1,n = 5,c;
   clean();
   vector<int> nums = {1,2,3,4,5,1,2,3,4,5,9,8,10};
   // for(auto &c:nums)cin>>c;
   for(int i = 0;i<nums.size();i++)
      insert(nums[i]);
   print(root);
   cout<<endl;
   // cout<<order_of_key(root,1)<<endl;;
   // cout<<order_of_key(root,2)<<endl;;
   // cout<<order_of_key(root,3)<<endl;;
   cout<<sum_to_key(root,1)<<endl;;
   cout<<sum_to_key(root,2)<<endl;;
   cout<<sum_to_key(root,3)<<endl;;
   cout<<sum_to_key(root,4)<<endl;;
   cout<<sum_to_key(root,5)<<endl;;
   cout<<sum_to_key(root,6)<<endl;;
   cout<<sum_to_key(root,7)<<endl;;
   cout<<sum_to_key(root,8)<<endl;;
   cout<<sum_to_key(root,9)<<endl;;
   cout<<sum_to_key(root,10)<<endl;;
   // cout<<getKth(root,1)<<endl;;
   // cout<<getKth(root,2)<<endl;;
   // cout<<getKth(root,3)<<endl;;
   // cout<<getKth(root,4)<<endl;;
   // cout<<getKth(root,5)<<endl;;
   // cout<<getKth(root,6)<<endl;;
   // cout<<getKthSum(root,1)<<endl;;
   // cout<<getKthSum(root,2)<<endl;;
   // cout<<getKthSum(root,3)<<endl;;
   // cout<<getKthSum(root,4)<<endl;;
   // cout<<getKthSum(root,5)<<endl;;
   // cout<<getKthSum(root,6)<<endl;;
   // cout<<getCountUntilSum(root,1)<<endl;
   // cout<<getCountUntilSum(root,2)<<endl;
   // cout<<getCountUntilSum(root,3)<<endl;
   // cout<<getCountUntilSum(root,4)<<endl;
   // cout<<getCountUntilSum(root,5)<<endl;
   // cout<<getCountUntilSum(root,6)<<endl;
   // cout<<getCountUntilSum(root,7)<<endl;
   // cout<<getCountUntilSum(root,8)<<endl;
   // cout<<getCountUntilSum(root,9)<<endl;
   // cout<<getCountUntilSum(root,10)<<endl;
   // cout<<getCountUntilSum(root,11)<<endl;
   // cout<<getCountUntilSum(root,12)<<endl;
   return 0;
}
```

10.21 vantagePointTree.cpp

vantagePointTree.cpp 90 lines

```
/*
 Vantage Point Tree (vp tree)
 Description:
 Vantage point tree is a metric tree.
 Each tree node has a point, radius, and two childs.
 The points of left descendants are contained in the ball B(p,r)
 and the points of right descendants are excluded from the ball.
 We can find k-nearest neighbors of a given point p efficiently
 by pruning search.
 Complexity:
 Construction: O(n log n)
 Search: O(log n)
*/
typedef complex<double> point;
namespace std
{
 bool operator <(point p, point q)
 {
  if (real(p) != real(q))
   return real(p) < real(q);
  return imag(p) < imag(q);
 }
}
struct vantage_point_tree
{
 struct node
 {
  point p;
  double th;
  node *l, *r;
 }*root;
 vector<pair<double, point>> aux;
 vantage_point_tree(vector<point> ps)
 {
  for (int i = 0; i < ps.size(); ++i)
   aux.push_back({ 0, ps[i] });
  root = build(0, ps.size());
 }
 node *build(int l, int r)
 {
  if (l == r)
   return 0;
  swap(aux[l], aux[l + rand() % (r - l)]);
  point p = aux[l++].second;
  if (l == r)
   return new node({ p });
  for (int i = l; i < r; ++i)
   aux[i].first = norm(p - aux[i].second);
  int m = (l + r) / 2;
```

```cpp
  nth_element(aux.begin() + l, aux.begin() + m, aux.begin() + r);
  return new node({ p, sqrt(aux[m].first), build(l, m), build(m, r) });
 }
 priority_queue<pair<double, node*>> que;
 void k_nn(node *t, point p, int k)
 {
  if (!t)
   return;
  double d = abs(p - t->p);
  if (que.size() < k)
   que.push({ d, t });
  else if (que.top().first > d)
  {
   que.pop();
   que.push({ d, t });
  }
  if (!t->l && !t->r)
   return;
  if (d < t->th)
  {
   k_nn(t->l, p, k);
   if (t->th - d <= que.top().first)
    k_nn(t->r, p, k);
  }
  else
  {
   k_nn(t->r, p, k);
   if (d - t->th <= que.top().first)
    k_nn(t->l, p, k);
  }
 }
 vector<point> k_nn(point p, int k)
 {
  k_nn(root, p, k);
  vector<point> ans;
  for (; !que.empty(); que.pop())
   ans.push_back(que.top().second->p);
  reverse(ans.begin(), ans.end());
  return ans;
 }
};
```

10.22 `waveletTree.cpp`

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

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

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

`waveletTree.cpp` 502a47, 135 lines

```cpp
typedef vector<int>::iterator it;
struct wavelet{
  vector<vector<int>> mapLeft;
  int mx;
  wavelet(vector<int> &A,int mx):mapLeft(mx*2),mx(mx){
    build(A.begin(),A.end(),0,mx-1,1);
  }
  void build(it s,it e,int l,int r,int v){
    if(l== r)return;
    int m = (l+r)>>1;
    mapLeft[v].reserve(e-s+1);
    mapLeft[v].push_back(0);
    auto f = [m](int x){
      return x<=m;
    };
    for(it iter = s; iter!= e;iter++)
      mapLeft[v].push_back(mapLeft[v].back() + (*iter<=m));
    it p = stable_partition(s,e,f);
    build(s,p,l,m,v<<1);
    build(p,e,m+1,r,v<<1|1);
  }
  //counts the number of elements equal to c in range [1,i]
  //IF you want in the range [i,j] only calls rank(j)- rank(i-1)
  int rank(int c,int i){
    i++;
    int l = 0,r = mx-1,u = 1,m,left;
    while(l!= r){
      m = (l+r)>>1;
      left = mapLeft[u][i];
      u<<=1;
      if(c<=m)
        i = left,r = m;
      else
        i-=left,l = m+1,u|=1;
    }
    return i;
  }
  // return the kth smallest element in a range [i,j]
  // k=1 is the smallest
  // 0 indexed this is indexes are in [0,n-1]
  int kth(int i,int j,int k){
    j++;
    int l = 0,r = mx-1,u = 1,li,lj;
    while(l!=r){
      int m = (l+r)>>1;
      li = mapLeft[u][i],lj = mapLeft[u][j];
      u<<=1;
      if(k<= lj-li)
        i = li,j = lj, r = m;
      else
        i-=li,j-=lj,l = m+1,u|=1,k-=(lj-li);
    }
    return r;
  }
  int kthSum(int i,int j,int k){
    j++;
    int l = 0,r = mx,li,lj;
    int si,sj;
    int ans = 0;
    int u = 1;
    while(l!=r){
      int m = (l+r)>>1;
      li = mapLeft[u][i],lj = mapLeft[u][j];
      si = sumLeft[u][i],sj = sumLeft[u][j];
      u<<=1;
      if(k<= lj-li){
        i = li,j = lj, r = m;
      }
      else {
        ans+=(sj-si);
        u|=1;
        i-=li,j-=lj,l = m+1,k-=(lj-li);
      }
    }
    ans+=rev[r]*k;
    return ans;
  }
  int l,r;
  // count the ocurrences of numbers in the range [a,b]
  // and only in the secuende [i,j]
  // can be seen as how many points are in a specified rectangle with
      corns i,a and j,b
  int range(int i ,int j ,int a,int b){
    if( b<a || j<i)return 0;
    l = a,r = b;
    return range(i,j+1,0,mx-1,1);
  }
  int range(int i, int j,int a,int b,int v){
    if(b<l || a>r)return 0;
    if(a>=l && b<=r)return j-i;
    int m = (a+b)>>1;
    int li = mapLeft[v][i],lj = mapLeft[v][j];
    return range(li,lj,a,m,v<<1)+range(i-li,j-lj,m+1,b,v<<1|1);
  }
  /*
```

```
    Return the minimun number that their frecuence in the range [i,j] is at least k
    complexity depends of k, if k is small and j-i is large maybe go up to o(n)
    the problem tested has a k up to (j-i)/5 and the complexity has O(5 log n)
  */
  int minimun_of_ocurrences(int i,int j,int k){
    return minimun_of_ocurrences(i,j+1,k,1,0,mx-1);
  }
  int minimun_of_ocurrences(int i,int j,int k,int v ,int l,int r ){
    if(l == r)return j-i>= k?l:mx+2;
    if(j-i<k)return mx+2;
    int m = (l+r)>>1;
    int li = mapLeft[v][i],lj = mapLeft[v][j];
    int c = lj-li;
    int ans= mx+2;
    if(c >=k)
      ans = min(ans,minimun_of_ocurrences(li,lj,k,v<<1,l,m));
    if((j-i)-c>=k)
      ans = min(ans,minimun_of_ocurrences(i-li,j-lj,k,v<<1|1,m+1,r));
    if(c <k && (j-i)-c<k)return mx+2;
    return ans;
  }
  /* swap element arr[i] and arr[i+1] */
  /*- No tested */
  void swapadjacent(int i){
    swapadjacent(i,0,mx-1,1);
  }
  void swapadjacent(int i,int l,int r,int v){
    if(l == r)
      return ;
    mapLeft[v][i]= mapLeft[v][i-1] + mapLeft[v][i+1] - mapLeft[v][i];
    // c[i] = c[i-1] + c[i+1] - c[i];
    if(mapLeft[v][i+1]-mapLeft[v][i] == mapLeft[v][i] - mapLeft[v][i-1]){
      if(mapLeft[v][i]-mapLeft[v][i-1])
        return swapadjacent(mapleft[v][i],l,mid,v<<1);
      else
        return swapadjacent(i-mapLeft[v][i],mid+1,r,v<<1|1);
    }
    else
      return ;
  }
};
```

test (11)

11.1 listToN.sh

listToN.sh — 23 lines

```bash
#!/bin/bash

# Read the value of n from the user
read -p "Enter the value of n: " n

# Initialize an empty array
numbers=()

# Loop through the numbers from 1 to n
for (( i=1; i<=n; i++ ))
do
  numbers+=($i) # Append the current number to the array
done

# Join the array elements with commas
numbers_string=$(IFS=,; echo "${numbers[*]}")

# Copy the array with numbers to the clipboard
echo "$numbers_string" | xclip -selection clipboard

echo "Array copied to the clipboard."
```

11.2 s.sh

s.sh — 63 lines

```bash
green=$(tput setaf 71);
red=$(tput setaf 12);
blue=$(tput setaf 32);
orange=$(tput setaf 178);
bold=$(tput bold);
reset=$(tput sgr0);

g++ -std=c++17 gen.cpp -o generator || { echo ${bold}${orange}Compilation Error in ${reset} gen.cpp; exit 1; }
g++ -std=c++17 $1.cpp -o original || { echo ${bold}${orange}Compilation Error${reset} in $1.cpp; exit 1; }
g++ -std=c++17 $2.cpp -o brute || { echo ${bold}${orange}Compilation Error${reset} in $2.cpp; exit 1; }

if [ $# -eq 2 ]
  then
    max_tests=10
  else
    max_tests=$3
fi
diff_found=0
i=1
while [ $i -le $max_tests ]
do
  ./generator > input1
  ./original < input1 > original_output || { echo "${orange}test_case #$i $1: ${bold}${red}RE${reset} "; exit 1; }

  ./brute < input1 > brute_outpu || { echo "${orange}test_case #$i $2: ${bold}${red}RE${reset} "; exit 1; }

  if diff --tabsize=1 -F --label --side-by-side --ignore-space-change original_output brute_output > dont_show_on_terminal; then
    echo "${orange}test_case #$i: ${bold}${green}AC${reset}"
  else
    echo "${orange}test_case #$i: ${bold}${red}WA${reset}"
    diff_found=1
    break
  fi
  i=$((i+1))
done

if [ $diff_found -eq 1 ]
then
  echo "${blue}Input: ${reset}"
  cat input1
  echo ""

  echo "${blue}Output: ${reset}"
  cat original_output
  echo ""

  echo "${blue}Expected: ${reset}"
  cat brute_output
  echo ""
  notify-send "Wrong Answer"
else
  notify-send "Accepted"
  rm input1
fi

rm generator
rm original
rm brute
rm original_output
rm brute_output
rm dont_show_on_terminal
```

11.3 gen.cpp

gen.cpp 101 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define accuracy chrono::steady_clock::now().time_since_epoch().count()
#define rep(i,a,n) for (int i = a; i <= n; ++i)
const int N = 1e6 + 4;
int32_t permutation[N];
mt19937 rng(accuracy);
int rand(int l, int r){
 uniform_int_distribution<int> ludo(l, r); return ludo(rng);
}
const int inf = 1LL << 31;
using pii = pair<int,int>;
namespace generator {
 string gen_string(int len = 0, bool upperCase = false, int l = 1, int r = 26) {
  assert(len >= 0 && len <= 5e6);
  string str(len, (upperCase ? 'A' : 'a'));
  for (char &ch: str) {
   ch += rand(l, r) - 1;
  }
  return str;
 }
 vector<int> gen_array(int len = 0, int minRange = 0, int maxRange = inf ){
  assert(len >= 0 and len <= 5e6);
  vector<int> a(len);
  for (int &x: a) x = rand(minRange, maxRange);
  return a;
 }
 vector<pair<int, int>> gen_tree(int n = 0){
  assert(n >= 0);
  vector<pii> res(n ? n - 1 : 0);
  // if you like to have bamboo like tree or star like tree uncomment below 8 lines
  /*if (rng() % 5 == 0) { // bamboo like tree
   for (int i = 1; i < n; ++i) res[i-1] = {i, i + 1};
   return res;
  }
  if (rng() % 7 == 0) { // star tree
   for (int i = 2; i <= n; ++i) res[i-2] = {1, i};
   return res;
  }*/
  iota(permutation, permutation + 1 + n, 0);
  shuffle(permutation + 1, permutation + 1 + n, rng);
  for(int i = 2; i <= n; ++i){
   int u = i, v = rand(1 , i-1);
   u = permutation[u], v = permutation[v];
   res[i-2] = minmax(u, v); // u < v, just for convenience while debugging
  }
  shuffle(res.begin() , res.end() , rng);
  return res;
 }
 vector<pair<int, int>> simple_graph(int n = 0, int m = 0) {
  assert(n > 0 && m >= n);
  int max_edges = n * (n - 1) / 2;
  assert(m <= max_edges);
  vector<pii> res = gen_tree(n);
  set<pii> edge(res.begin(), res.end());
  for (int i = n; i <= m; ++i) {
   while (true) {
    int u = rand(1, n), v = rand(1, n);
    if (u == v) continue;
    auto it = edge.insert(minmax(u, v));
    if (it.second) break;
   }
  }
  res.assign(edge.begin(), edge.end());
  return res;
 }
}
using namespace generator;
template<typename T = int>
ostream& operator<< (ostream &other, const vector<T> &v) {
  for (const T &x: v) other << x << ' ';
  other << '\n';
  return other;
}
ostream& operator<< (ostream &other, const vector<pair<int,int>> &v) {
  for (const auto &x: v) other << x.first << ' ' << x.second << '\n';
  return other;
}
vector<string> D= {"Mon","Tue","Wed","Thu","Fri","Sat","Sun"};
// comment the just below line if test cases required
#define SINGLE_TEST
const int max_tests = 10;
// complete this function according to the requirements
void generate_test() {
 int n = rand(0, 6);
 cout<<D[n];
 n = rand(1,1000);
 cout << n << '\n';
 cout << gen_array(n, 0, 10000);
}
signed main() {
 srand(accuracy);
 int t = 1;
 #ifndef SINGLE_TEST
  t = rand(1, max_tests), cout << t << '\n';
 #endif
 while (t--) {
  generate_test();
 }
}
```

11.4 genTree2.cpp

genTree2.cpp 135 lines

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define accuracy chrono::steady_clock::now().time_since_epoch().count()
#define rep(i,a,n) for (int i = a; i <= n; ++i)
const int N = 1e6 + 4;
int32_t permutation[N];
mt19937 rng(accuracy);
int rand(int l, int r){
 uniform_int_distribution<int> ludo(l, r); return ludo(rng);
}
const int inf = 1LL << 31;
using pii = pair<int,int>;
namespace generator {
 string gen_string(int len = 0, bool upperCase = false, int l = 1, int r = 26) {
  assert(len >= 0 && len <= 5e6);
  string str(len, (upperCase ? 'A' : 'a'));
  for (char &ch: str) {
   ch += rand(l, r) - 1;
  }
  return str;
 }
 vector<int> gen_array(int len = 0, int minRange = 0, int maxRange = inf ){
  assert(len >= 0 and len <= 5e6);
  vector<int> a(len);
  for (int &x: a) x = rand(minRange, maxRange);
  return a;
 }
 vector<pair<int, int>> gen_tree(int n = 0){
  assert(n >= 0);
  vector<pii> res(n ? n - 1 : 0);
  // if you like to have bamboo like tree or star like tree uncomment below 8 lines
  /*if (rng() % 5 == 0) { // bamboo like tree
   for (int i = 1; i < n; ++i) res[i-1] = {i, i + 1};
```

```cpp
    return res;
  }
  if (rng() % 7 == 0) { // star tree
    for (int i = 2; i <= n; ++i) res[i-2] = {1, i};
    return res;
  }*/
  iota(permutation, permutation + 1 + n, 0);
  shuffle(permutation + 1, permutation + 1 + n, rng);
  for(int i = 2; i <= n; ++i){
    int u = i, v = rand(1 , i-1);
    u = permutation[u], v = permutation[v];
    res[i-2] = minmax(u, v); // u < v, just for convenience while debugging
  }
  shuffle(res.begin() , res.end() , rng);
  return res;
 }
 vector<pair<int, int>> simple_graph(int n = 0, int m = 0) {
  assert(n > 0 && m >= n);
  int max_edges = n * (n - 1) / 2;
  assert(m <= max_edges);
  vector<pii> res = gen_tree(n);
  set<pii> edge(res.begin(), res.end());
  for (int i = n; i <= m; ++i) {
    while (true) {
      int u = rand(1, n), v = rand(1, n);
      if (u == v) continue;
      auto it = edge.insert(minmax(u, v));
      if (it.second) break;
    }
  }
  res.assign(edge.begin(), edge.end());
  return res;
 }
}
using namespace generator;
template<typename T = int>
ostream& operator<< (ostream &other, const vector<T> &v) {
  for (const T &x: v) other << x << ' ';
  other << '\n';
  return other;
}
ostream& operator<< (ostream &other, const vector<pair<int,int>> &v) {
  for (const auto &x: v) other << x.first << ' ' << x.second << '\n';
  return other;
}
// comment the just below line if test cases required
#define SINGLE_TEST
const int max_tests = 10;
// complete this function according to the requirements
void generate_test() {
  int n = rand(1, 40);
  cout << n << '\n';
  cout << gen_array(n, 1, 20);
}
signed main() {
  srand(accuracy);
  int t = 1;
  #ifndef SINGLE_TEST
    t = rand(1, max_tests), cout << t << '\n';
  #endif
  while (t--) {
    generate_test();
  }
}// generating a tree in a not-so-stupid way
#include <bits/stdc++.h>
using namespace std;
int rand(int a, int b) {
    return a + rand() % (b - a + 1);
}
int main(int argc, char* argv[]) {
    srand(atoi(argv[1]));
    int n = rand(5, 15);
    printf("%d\n", n);
    vector<pair<int,int>> edges;
    for(int i = 2; i <= n; ++i) {
        edges.emplace_back(rand(1, i - 1), i);
    }
    vector<int> perm(n + 1); // re-naming vertices
    for(int i = 1; i <= n; ++i) {
        perm[i] = i;
    }
    random_shuffle(perm.begin() + 1, perm.end());
    random_shuffle(edges.begin(), edges.end()); // random order of edges
    for(pair<int, int> edge : edges) {
        int a = edge.first, b = edge.second;
        if(rand() % 2) {
            swap(a, b); // random order of two vertices
        }
        printf("%d %d\n", perm[a], perm[b]);
    }
    int q = rand(1,10);
    cout<<q<<endl;
    for(int i = 0;i<q;i++){
        int u = rand(1,n),v;
        while(1){
            v = rand(1,n);
            if(v !=u)break;
        }
        cout<<u<<" "<<v<<endl;
    }
}
```

Techniques (A)

techniques.txt 218 lines

```
#Techniques / Hints / things to think

#Do stress test with a brute solution
#Recursion
#Divide and conquer
 - Finding interesting points in N log N
#Techniques/ things to thing
 * Meet in the middle
 * Two pointers
 * Sweep line
 * Prefix function
 * Try different complexity for different sizes SQRT for sizes ≤ SQRT
     ↪ and other solution
 * How many different values are?
 * Greedy
 * See in reverse way
 * Duplicate array
 * How fast grow
 * Does have a binary behavior
 * Divide and conquer
 * xor hashing
 * Mo's algorithm , SQRT decomposition etc.
#Algorithm analysis
 - Master theorem
 - Amortized time complexity
#Greedy algorithm
 - Scheduling
 - Max contiguous subvector sum
 - Invariants
 - Huffman encoding
#Graph theory
 - Dynamic graphs (extra book-keeping)
 - Breadth first search
 - Depth first search
 - * Normal trees / DFS trees
 - Dijkstra's algorithm
 - MST: Prim's algorithm
 - Bellman-Ford
 - Konig's theorem and vertex cover
 - Min-cost max flow
 - Lovasz toggle
 - Matrix tree theorem / Number Spaning trees in a graph
 - Maximal matching, general graphs
 - Hopcroft-Karp
 - Hall's marriage theorem {\displaystyle |W|\leq |N_{G}(W)|.}
 - Graphical sequences
 - Floyd-Warshall
 - Euler cycles
 - Flow networks
 - * Augmenting paths
 - * Edmonds-Karp
 - * Min cost max flow
 - * Min cut
 - Bipartite matching
 - Min. path cover
 - Topological sorting
 - Strongly connected components
 - 2-SAT
 - Cut vertices, cut-edges and biconnected components
 - Edge coloring
 - * Trees
 - Vertex coloring
 - * Bipartite graphs (=> trees)
 - * 3^n (special case of set cover)
 - Diameter and centroid
 - DSU on tree.
 - Small to large.
 - K'th shortest path
 - Shortest cycle
 - Euler tour, can manage path to root and subtree queries
 - Euler tour tree
 - link cut tree
 - Add Dummy nodes

#Dynamic programming
 - Knapsack
 - Coin change
 - Longest common subsequence
 - Longest increasing subsequence
 - Number of paths in a dag
 - Shortest path in a dag
 - Dynprog over intervals
 - Dynprog over subsets
 - Dynprog over probabilities
 - Dynprog over trees
 - 3^n set cover
 - SOS DP (n*2^n)
 - Divide and conquer
 - Knuth optimization
 - Convex hull optimizations
 - Alien trick
 - 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
 - Stirling
 - Bell numbers
 - Pick's theorem
Number theory
 - Integer parts
 - Divisibility
 - Euclidean algorithm
 - Modular arithmetic
 - Linear Congruence Equation -> a . x == b (mod n) -> x = b . a^-1 b
 - Linear diophantine Equation -> ax +by = c
 - Discrete log -> find x such a^x == b (mod n)
 - Discrete root -> find x such x^k == a (mod n)
 - * Modular multiplication
 - * Modular inverses
 - * Modular exponentiation by squaring
 - Chinese remainder theorem
 - Fermat's little theorem
 - Euler's theorem
 - Phi function
 - Frobenius number
 - Quadratic reciprocity
 - Pollard-Rho
 - Miller-Rabin
 - Hensel lifting
 - Vieta root jumping
 - Subset sum (DP, NTT)
Game theory
 - Combinatorial games
 - Game trees
 - Mini-max
 - Nim
 - Games on graphs
 - Games on graphs with loops
 - Grundy numbers
 - Bipartite games without repetition
 - General games without repetition
 - Alpha-beta pruning
Probability theory
Optimization
 - Binary search
 - Ternary search
```

```
  - Unimodality and convex functions
  - Binary search on derivative
Numerical methods
  - Numeric integration
  - Newton's method
  - Root-finding with binary/ternary search
  - Golden section search
Matrices
  - Gaussian elimination
  - Exponentiation by squaring
Sorting
  - Radix sort
Geometry
  - Coordinates and vectors
  - * Cross product
  - * Scalar product
  - Convex hull
  - Polygon cut
  - Closest pair
  - Coordinate-compression
  - Quadtrees
  - KD-trees
  - All segment-segment intersection
Sweeping
  - Discretization (convert to events and sweep)
  - Angle sweeping
  - Line sweeping
  - Discrete second derivatives
Strings
  - Longest common substring
  - Palindrome subsequences
  - Knuth-Morris-Pratt
  - Tries
  - Rolling polynomial hashes
  - Suffix array
  - Suffix tree
  - Aho-Corasick
  - Manacher's algorithm
  - Letter position lists
Combinatorial search
  - Meet in the middle
  - Brute-force with pruning
  - Best-first (A*)
  - Bidirectional search
  - Iterative deepening DFS / A*
Data structures
  - LCA (2^k-jumps in trees in general)
  - Pull/push-technique on trees
  - Heavy-light decomposition
  - Centroid decomposition
  - Lazy propagation
  - Self-balancing trees
  - Convex hull trick
  - Monotone queues / monotone stacks / sliding queues
  - Sliding queue using 2 stacks
  - Persistent segment tree
  - Treap (Can be used as order statics sets with extra operations)
  - Implicit treap (Full dynamic array with operations in range)
  - O(1) queries with disjoint sparse table
General
  - If problem is check for all multiples of a number in [1,n] and this
      ↪ multiples don´t exed maxn complexity is (maxn log(maxn))
  - Sum of n/1+ n/2 + n/3 + n/4 +  ... is nlogn
  - Merge many sets can be do it in n logn if we insert elements of the
      ↪ minor set to the mayor set
  - Strings? Do you made a suffix array or suffix tree
  - Graphs? shortes Path with two variables or many types of edges? try
      ↪ to clone graph
  - try to decompose the formula
  - TLE? and modulos , try to do less % operations if you have long long
      ↪ do modulo only when a≥(mod*8) where modulo is something like
      ↪ 1e9+7
  - Best of all posibilities whit small n like 30-40 try meet in the
      ↪ middle
  - need a subset whit some features and at least n/2 elements? try
      ↪ randomize
  - Boolean assigments? 2-sat? basisxor? SLAE?
  - Queries about paths with some specific value like sum xor? try to
      ↪ decompose with centroid decomposition and solve for each tree
      ↪ root in each centroid
  - Check parity
  - querys on path of tree? if it´s only to root is enough to do an euler
      ↪  traversal and flat the tree in other case use HLD
  - Work with ceil function≤
 xn□□≤
xn
Also note that since □A/□B and C are both integers, □A/□B<C is
     ↪ equivalent to □A/□≤-BC1.
Combining these equivalences means that the original condition is
     ↪ equivalent to A/≤-BC1.
```

A.1 CheatSheet

Ejemplo

Cheat Sheet under construction

Hi	Hello world
Hi	Hello world

Ejemplo

Exemplo básico de plotagem de gráfico:

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('Números de Exemplo')
plt.show()
```

Neste exemplo, foi gerado um valor para Y baseado no valor de X informado.

A.2 Some numbers

A.2.1 Stirling numbers of the first kind

stirlingInk represents the number of permutations of n elements in exactly k disjoint cycles.

$$\begin{bmatrix}0\\0\end{bmatrix} = 1$$

$$\begin{bmatrix}0\\n\end{bmatrix} = \begin{bmatrix}n\\0\end{bmatrix} = 0 \qquad , \quad n > 0$$

$$\begin{bmatrix}n\\k\end{bmatrix} = (n-1)\begin{bmatrix}n-1\\k\end{bmatrix} + \begin{bmatrix}n-1\\k-1\end{bmatrix} \qquad , \quad k > 0$$

$$\sum_{k=0}^{n}\begin{bmatrix}n\\k\end{bmatrix} = n!$$

$$\sum_{k=0}^{\infty}\begin{bmatrix}n\\k\end{bmatrix}x^k = \prod_{k=0}^{n-1}(x+k)$$

A.2.2 Stirling numbers of the second type

stirlingIInk represents the number of ways to partition a set of n distinguishable objects into k nonempty subsets.

$$\begin{Bmatrix}0\\0\end{Bmatrix} = 1$$

$$\begin{Bmatrix}0\\n\end{Bmatrix} = \begin{Bmatrix}n\\0\end{Bmatrix} = 0 \qquad , \quad n > 0$$

$$\begin{Bmatrix}n\\k\end{Bmatrix} = k\begin{Bmatrix}n-1\\k\end{Bmatrix} + \begin{Bmatrix}n-1\\k-1\end{Bmatrix} \qquad , \quad k > 0$$

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

A.2.3 Euler numbers

$\left\langle {n \atop k} \right\rangle$ represents the number of permutations from 1 to n where exactly k numbers are greater than the previous number.

$$\left\langle {1 \atop 0} \right\rangle = 1$$

$$\left\langle {n \atop k} \right\rangle = (n-k)\left\langle {n-1 \atop k-1} \right\rangle + (k+1)\left\langle {n-1 \atop k} \right\rangle \qquad , \quad n \geq 2$$

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

$$\sum_{k=0}^{n-1}\left\langle {n \atop k} \right\rangle = n!$$

A.2.4 Catalan numbers

$$C_0 = 1$$

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \sum_{j=0}^{n-1}C_jC_{n-1-j}$$

$$\sum_{n=0}^{\infty}C_nx^n = \frac{1-\sqrt{1-4x}}{2x}$$

- Parentheses Expressions: Catalan numbers count the number of different ways to arrange parentheses in a valid expression. For example, for n = 3, there are five valid expressions: ((())), ()(()), (())(), ()()(), and (()()).
- Binary Trees: They count the number of structurally different binary search trees with n nodes. Binary search trees are used in computer science for data storage and searching.
- Polygon Triangulations: In geometry, Catalan numbers count the ways to triangulate a convex polygon with n+2 sides, meaning the number of non-overlapping triangles formed by connecting n+2 vertices.

A.2.5 Bell Numbers

B_n represents the number of ways to partition a set of n elements.

$$B_n = \sum_{k=0}^{n}\begin{Bmatrix}n\\k\end{Bmatrix} = \sum_{k=0}^{n-1}\binom{n-1}{k}B_k$$

$$\sum_{n=0}^{\infty}\frac{B_n}{n!}x^n = e^{e^x-1}$$

A.2.6 Bernoulli numbers

$$
\begin{aligned}
{B_0}^+ &= 1 \\
{B_n}^+ &= 1 - \sum_{k=0}^{n-1} \binom{n}{k} \frac{{B_k}^+}{n-k+1}
\end{aligned}
$$

$$\sum_{n=0}^{\infty} \frac{{B_n}^+ x^n}{n!} = \frac{x}{1-e^{-x}} = \frac{1}{\frac{1}{1!} - \frac{x}{2!} + \frac{x^2}{3!} - \frac{x^3}{4!} + \cdots}$$

A.3 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.

A.4 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 $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

A.5 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 $n < 1e19$.

A.6 Mobius Function

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

Mobius Inversion:

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

Other useful formulas/forms:

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

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

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

A.7 Complexity

- If problem is check for all multiples of a number in [1,n] and this multiples don´t exed maxn complexity is (maxn log(maxn)) - Sum of n/1+ n/2 + n/3 + n/4 + ... is nlogn - Merge many sets can be do it in n logn if we insert elements of the minor set to the mayor set - Strings? Do you made a suffix array or suffix tree - Graphs? shortes Path with two variables or many types of edges? try to clone graph - try to decompose the formula - TLE? and modulos , try to do less - Best of all posibilities whit small n like 30-40 try meet in the middle - need a subset whit some features and at least n/2 elements? try randomize - Boolean assigments? 2-sat? basisxor? SLAE? - Queries about paths with some specific value like sum xor? try to decompose with centroid decomposition and solve for each tree root in each centroid - Querys on path of tree? if it´s only to root is enough to do an euler traversal and flat the tree in other case use HLD

If g a, then g ab.

If gx ax then g a.

If g a and g b then for every (x,y) g ax+by.

Furthermore, you should assume all variables defined in this blog from here on are integers unless mentioned otherwise.

For all numbers k,n, there exists unique q,r such that n=kq+r where 0 r<|k|. This is the elementary theorem of division.

amodk creates an equivalence relation, where a bmodk if for some (q1,q2,r), a=q1k+r and b=q2k+r. Notice that this is equivalent to k (a—b)

A.8 tricks

- ((x - $a + b = a^b + 2 \times a\&b$ - $a + b = (a||b) + (a\&b)$. - a1=(a01+a12—a02)/2 - lcm(a, gcd(b, c)) = gcd(lcm(a, b), lcm(a, c)) – (1) - gcd(a, lcm(b, c)) = lcm(gcd(a, b), gcd(a, c)) – (2) - gcd(a,b) = gcd(a,b-a); - Desplazar coeficientes de multiplicación pasar de esto w *3 + x*2 + y*1 -> w*4 + x*3 +y*2 +z*1 usar doble acumulado, La diferencia ente uno y otros es w+x+y+z - El numero de maneras de conectar un grafo con k componentes conexas con el minimo número de aristas donde s_i es el tamaño del componente $(s_1 \times s_2 \ldots s_k) \times\ n^{k-2}$ - Sum of nrc(i*2,n) 0<= i <= n/2 is $2^(n-1)$ - Sum of nrc(i,n) 0<= i <= n is $2^(n)$ - (ip) = p (i). -A number in base 10 has non-repeating decimals if its reduced fraction s/t has a denominator in the form t = $2^v \times 5^w$