
UNIVERSIDADE FEDERAL
DE ALAGOAS

Universidade Federal de Alagoas

# Floyd (WA)rshall

João Ayalla, Filipe Ferreira, Davi Romão

ICPC World Finals 2025

September 4, 2025

# Contest (1)

## template.cpp

25 lines

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

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

template <class T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

#define int long long int
#define pb push_back
#define pi pair<int, int>
#define pii pair<pi, int>
#define fir first
#define sec second
#define MAXN 1000006
#define mod 998244353

signed main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  mt19937_64 rng(chrono::steady_clock::now().time_since_epoch()
      .count());
  return 0;
}
```

## runner.py

44 lines

```
import os
import subprocess

naive = "brute.cpp" # path to naive code
code = "d.cpp" # path to your code
generator = "g.cpp" # path to test generator

def compile_codes():
    os.system('g++ ' + generator + ' -o generator -O2')
    os.system('g++ ' + naive + ' -o naive -O2')
    os.system('g++ ' + code + ' -o code -O2')

def generate_case():
    os.system('./generator > in');

def get_naive_output():
    output = os.popen('./naive <in').read()
    return output

def get_code_output():
    # se tiver um runtime error, vai parar
    xalala = subprocess.run('./code <in', shell=True, text=True
        , capture_output=True, check=True)
    return xalala.stdout

def main():
    compile_codes()

    while True:
        generate_case()
        naive_output = get_naive_output()
        code_output = get_code_output()

        if naive_output == code_output:
            print('ACCEPTED')
        else :
            print('FAILED\n')
            print('ANSWER:')
            print(naive_output)
            print('\nCODE OUTPUT:')
            print(code_output)
            break

if __name__ == '__main__':
    main()
```

## pragma.cpp

5 lines

```
#include <iostream>
using namespace std;
#pragma GCC target("avx2")
#pragma GCC optimize("O3")
#pragma GCC optimize("unroll-loops")
```

# Mathematics (2)

## modint.cpp

78 lines

```
struct modint {
  int val;
  modint(int v = 0) { val = v % mod; }
  int pow(int y) {
    modint x = val;
    modint z = 1;
    while (y) {
      if (y & 1) z *= x;
      x *= x;
      y >>= 1;
    }
    return z.val;
  }
  int inv() { return pow(mod - 2); }
  void operator=(int o) { val = o % mod; }
  void operator=(modint o) { val = o.val % mod; }
  void operator+=(modint o) { *this = *this + o; }
  void operator-=(modint o) { *this = *this - o; }
  void operator*=(modint o) { *this = *this * o; }
  void operator/=(modint o) { *this = *this / o; }
  bool operator==(modint o) { return val == o.val; }
  bool operator!=(modint o) { return val != o.val; }
  int operator*(modint o) { return ((val * o.val) % mod); }
  int operator/(modint o) { return (val * o.inv()) % mod; }
  int operator+(modint o) { return (val + o.val) % mod; }
  int operator-(modint o) { return (val - o.val + mod) % mod; }
};
modint f[MAXN];
modint inv[MAXN];
modint invfat[MAXN];
void calc() {
  f[0] = 1;
  for (int i = 1; i < MAXN; i++) {
    f[i] = f[i - 1] * i;
  }
  inv[1] = 1;
  for (int i = 2; i < MAXN; ++i) {
    int val = mod / i;
    val = (inv[mod % i] * val) % mod;
    val = mod - val;
    inv[i] = val;
  }
  invfat[0] = 1;
  invfat[MAXN - 1] = modint(f[MAXN - 1]).inv();
  for (int i = MAXN - 2; i >= 1; i--) {
    invfat[i] = invfat[i + 1] * (i + 1);
  }
}
modint ncr(int n, int k)  // combinacao
{
  modint ans = f[n] * invfat[k];
  ans *= invfat[n - k];
  return ans;
}
modint arr(int n, int k)  // arranjo
{
  modint ans = f[n] * invfat[n - k];
  return ans;
}
modint ncr(int n, int k) {
  // calcular combinacao para n grande
  // nesse problema n <= 10^12
  // em O(k)
  modint num = 1;
  modint den = 1;
  for (int i = 0; i < k; i++) {
    num = num * modint(n - i);
    den = den * modint(i + 1);
  }
  modint ans = num / den;
  return ans;
}
modint stars_and_bars(int n, int k) {
  // para pares de inteiros n e k
  // enconte a quantidade de k-tuplas com soma == n
  // x1 + x2 + ... + xk = n
  return ncr(n + k - 1, k - 1);
}
```

## divisiontrick.cpp

5 lines

```
for (int l = 1, r; l <= n; l = r + 1) {
  // todos os numeros i no intervalo [l, r] possuem (n / i) ==
      x
  r = n / (n / l);
  int x = (n / l);
}
```

## crivo.cpp

23 lines

```
bitset<MAXN> prime;
vector<int> nxt(MAXN);
vector<int> factors;
void crivo() {
  prime.set();
```

```
  prime[0] = false, prime[1] = false;
  for (int i = 2; i < MAXN; i++) {
    if (prime[i]) {
      nxt[i] = i;
      for (int j = 2; j * i < MAXN; j++) {
        prime[j * i] = false;
        nxt[j * i] = i;
      }
    }
  }
}
void fact(int n) {
  factors.clear();
  while (n > 1) {
    factors.pb(nxt[n]);
    n = n / nxt[n];
  }
}
```

## gaussianelimination.cpp

33 lines

```
#define EPS 1e-9
vector<double> ans;
int gauss(vector<vector<double>> a) {
  int n = a.size(), m = a[0].size() - 1, ret = 1;
  ans.assign(m, 0);
  vector<int> where(m, -1);
  for (int col = 0, row = 0; col < m && row < n; col++, row++)
       {
    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
            ;
      }
    }
  }
  for (int i = 0; i < m; i++) {
    if (where[i] != -1)
      ans[i] = (a[where[i]][m] / a[where[i]][i]);
    else
      ret = 2;
  }
  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) ret = 0;
  }
  return ret;  // 0 = nao existe solucao, 1 = existe uma
       solucao, 2 = existem multiplas solucoes
}
```

## gaussianbitset.cpp

31 lines

```
bitset<MAXN> ans;
int gauss(vector<bitset<MAXN>>& a) {
  ans.reset();
  int n = a.size(), m = a[0].size() - 1, ret = 1;
  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] / a[where[i]][i]);
    else
      ret = 2;
  }
  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) ret = 0;
  }
  return ret;
}
```

## lagrange.cpp

74 lines

```
struct lagrange {
  int n;
  vector<modint> den;
  vector<modint> y;
  vector<modint> fat;
  vector<modint> inv_fat;
  lagrange(vector<modint> &v)  // f(i) = v[i], gera um
       polinomio de grau n - 1
  {
    n = v.size();
    calc(n);
    calc_den(n);
    y = v;
  }
  void calc_den(int n) {
    den.resize(n);
    for (int i = 0; i < n; i++) {
      den[i] = inv_fat[n - i - 1] * inv_fat[i];
      if ((n - i - 1) % 2 == 1) {
        int x = (mod - den[i].val) % mod;
        den[i] = x;
      }
    }
  }
  void calc(int n) {
    fat.resize(n + 1);
    inv_fat.resize(n + 1);
    fat[0] = 1;
    inv_fat[0] = 1;
    for (int i = 1; i <= n; i++) {
      fat[i] = fat[i - 1] * i;
      inv_fat[i] = fat[i].inv();
    }
  }
  modint get_val(int x)  // complexidade: O(n)
  {
    x %= mod;
    vector<modint> l(n);
    vector<modint> r(n);
    l[0] = 1, r[n - 1] = 1;
    for (int i = 1; i < n; i++) {
      modint cof = (x - (i - 1) + mod);
      l[i] = l[i - 1] * cof;
    }
    for (int i = n - 2; i >= 0; i--) {
      modint cof = (x - (i + 1) + mod);
      r[i] = r[i + 1] * cof;
    }
    modint ans = 0;
    for (int i = 0; i < n; i++) {
      modint cof = l[i] * r[i];
      ans += modint(cof * y[i]) * den[i];
    }
    return ans;
  }
  vector<modint> find_coefs()  // encontra os coeficientes do
       polinomio
  {
    int nn = n;
    int d = nn - 1;
    vector<modint> c(nn, 0);
    for (int i = 0; i < y.size(); i++) {
      c[d] += (y[i] * den[i]);
    }
    for (int p = nn - 2; p >= 0; p--) {
      nn--;
      calc_den(nn);
      for (int i = 0; i <= p; i++) {
        y[i] -= (c[p + 1] * modint(i).pow(d));
        c[p] += (y[i] * den[i]);
      }
      d--;
    }
    return c;
  }
};
```

## berlekampmassey.cpp

73 lines

```
// berlekamp massey
// mas precisa que o mod seja primo (para poder achar inverso)
// dado os n primeiros termos de uma recorrencia linear
// a[0], a[1], a[2], ..., a[n - 1]
// ele acha a reccorrencia linear mais curta que da matching
     com os n primeiros valores
vector<modint> berlekamp_massey(vector<modint> x) {
  vector<modint> ls, cur;
  int lf, ld;
  for (int i = 0; i < x.size(); i++) {
    modint t = 0;
    for (int j = 0; j < cur.size(); j++) {
      t += (x[i - j - 1] * cur[j]);
    }
    if (modint(t - x[i]).val == 0) continue;
    if (cur.empty()) {
      cur.resize(i + 1);
      lf = i;
      ld = (t - x[i]) % mod;
      continue;
    }
    modint k = -(x[i] - t);
    k *= modint(ld).inv();
    vector<modint> c(i - lf - 1);
    c.pb(k);
    for (auto const &j : ls) {
      modint curr = modint(j.val * -1) * k;
      c.pb(curr);
    }
    if (c.size() < cur.size()) c.resize(cur.size());
    for (int j = 0; j < cur.size(); j++) {
      c[j] = c[j] + cur[j];
    }
    if (i - lf + ls.size() >= cur.size()) {
```

```
      tie(ls, lf, ld) = make_tuple(cur, i, t - x[i]);
    }
    cur = c;
  }
  return cur;
}
modint get_nth(vector<modint> rec, vector<modint> dp, int n) {
  int m = rec.size();
  vector<modint> s(m), t(m);
  s[0] = 1;
  if (m != 1)
    t[1] = 1;
  else
    t[0] = rec[0];
  auto mul = [&rec](vector<modint> v, vector<modint> w) {
    vector<modint> ans(2 * v.size());
    for (int j = 0; j < v.size(); j++) {
      for (int k = 0; k < v.size(); k++) ans[j + k] += v[j] * w
          [k];
    }
    for (int j = 2 * v.size() - 1; j >= v.size(); j--) {
      for (int k = 1; k <= v.size(); k++) ans[j - k] += ans[j]
          * rec[k - 1];
    }
    ans.resize(v.size());
    return ans;
  };
  while (n) {
    if (n & 1) s = mul(s, t);
    t = mul(t, t);
    n >>= 1;
  }
  modint ret = 0;
  for (int i = 0; i < m; i++) ret += s[i] * dp[i];
  return ret;
}
modint guess_nth_term(vector<modint> x, int n) {
  if (n < x.size()) return x[n];
  vector<modint> coef = berlekamp_massey(x);  // coeficientes
      da recorrencia
  if (coef.empty()) return 0;
  return get_nth(coef, x, n);
}
```

## crt.cpp

38 lines

```
namespace crt {
vector<pi> eq;
int gcd(int a, int b, int &x, int &y) {
  if (b == 0) {
    x = 1, y = 0;
    return a;
  }
  int x1, y1, d = gcd(b, a % b, x1, y1);
  x = y1, y = x1 - y1 * (a / b);
  return d;
}
pi crt() {
  int a1 = eq[0].fir, m1 = eq[0].sec;
  a1 %= m1;
  for (int i = 1; i < eq.size(); i++) {
    int a2 = eq[i].fir, m2 = eq[i].sec;
    int g = __gcd(m1, m2);
    if (a1 % g != a2 % g) return {-1, -1};
    int p, q;
    gcd(m1 / g, m2 / g, p, q);
    int mod = m1 / g * m2;
    int x = (a1 * (m2 / g) % mod * q % mod + a2 * (m1 / g) %
        mod * p % mod) % mod;
    a1 = x;
    if (a1 < 0) a1 += mod;
    m1 = mod;
  }
  return {a1, m1};
}
}  // namespace crt
// o menor inteiro a que satisfaz:
// a mod p1 = x1
// a mod p2 = x2
// a funcao crt retorna um pair {a, mod}
// dai a solucao pode ser descrita como
// x = a % mod
// entao os valores possiveis sao:
// a, (a + mod), a + (2 * mod), a + (3 * mod), ...
// cuidado com overflow!
```

## crttrick.cpp

19 lines

```
vector<pi> eq;
map<int, int> by_mod;
// quero checar se existe solucao para o sistema das equacoes
    que ja adicionei
// junto da equacao curr
// geralmente da pra fazer algo como if (check(curr)) { add(
    curr); }
bool check(pi curr) {
  if (by_mod.find(curr.sec) != by_mod.end()) {
    return by_mod[curr.sec] == curr.fir;
  }
  for (auto [x, mod] : eq) {
    if ((curr.fir - x) % __gcd(curr.sec, mod)) return 0;
  }
  return 1;
}
void add(pi curr) {  // [valor, mod]
  eq.pb(curr);
  by_mod[curr.sec] = curr.fir;
}
// quando tem algo de sqrt mods distintos ou algo do tipo
```

## diophantine.cpp

32 lines

```
namespace dio {
vector<pi> sols;
int gcd(int a, int b, int &x, int &y) {
  if (b == 0) {
    x = 1, y = 0;
    return a;
  }
  int x1, y1, d = gcd(b, a % b, x1, y1);
  x = y1, y = x1 - y1 * (a / b);
  return d;
}
void one_sol(int a, int b, int c) {
  int x0, y0, g;
  g = gcd(abs(a), abs(b), x0, y0);
  if (c % g) return;
  x0 *= (c / g);
  y0 *= (c / g);
  if (a < 0) x0 *= -1;
  if (b < 0) y0 *= -1;
  sols.pb({x0, y0});
}
void more_sols(int a, int b, int c) {
  int g = __gcd(a, b);
  int x0 = sols[0].fir, y0 = sols[0].sec;
  for (int k = -200000; k <= 200000; k++) {
    int x = x0 + k * (b / g);
    int y = y0 - k * (a / g);
    sols.pb({x, y});
  }
}
}  // namespace dio
// equacoes do tipo: ax + by = c
```

## extendedeuclidean.cpp

14 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;
}
// achar os numeros x e y tal que:
// a * x + b * y = gcd(a, b)
```

## ntt.cpp

38 lines

```
const int MOD = 998244353;
typedef mod_int<MOD> mint;
void ntt(vector<mint>& a, bool rev) {
  int n = a.size();
  auto b = a;
  assert(!(n & (n - 1)));
  mint g = 1;
  while ((g ^ (MOD / 2)) == 1) g += 1;
  if (rev) g = 1 / g;
  for (int step = n / 2; step; step /= 2) {
    mint w = g ^ (MOD / (n / step)), wn = 1;
    for (int i = 0; i < n / 2; i += step) {
      for (int j = 0; j < step; j++) {
        auto u = a[2 * i + j], v = wn * a[2 * i + j + step];
        b[i + j] = u + v;
        b[i + n / 2 + j] = u - v;
      }
      wn = wn * w;
    }
    swap(a, b);
  }
  if (rev) {
    auto n1 = mint(1) / n;
    for (auto& x : a) x *= n1;
  }
}
vector<mint> convolution(const vector<mint>& a, const vector<
    mint>& b) {
  vector<mint> l(a.begin(), a.end()), r(b.begin(), b.end());
  int N = l.size() + r.size() - 1, n = 1 << __lg(2 * N - 1);
  l.resize(n);
  r.resize(n);
  ntt(l, false);
  ntt(r, false);
  for (int i = 0; i < n; i++) l[i] *= r[i];
  ntt(l, true);
  l.resize(N);
  return l;
}
```

## fft.cpp

43 lines

```
#define PI acos(-1)
```

```
#define cd complex<double>
namespace fft {
int n;
void fft(vector<cd> &a, bool invert) {
  int n = a.size();
  for (int i = 1, j = 0; i < n; i++) {
    int bit = n >> 1;
    for (; j & bit; bit >>= 1) j ^= bit;
    j ^= bit;
    if (i < j) swap(a[i], a[j]);
  }
  for (int len = 2; len <= n; 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 (cd &x : a) x /= n;
}
vector<int> mul(vector<int> a, vector<int> b) {
  vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
  n = 1;
  while (n < a.size() + b.size()) n <<= 1;
  fa.resize(n);
  fb.resize(n);
  fft(fa, false);
  fft(fb, false);
  for (int i = 0; i < n; i++) fa[i] *= fb[i];
  fft(fa, true);
  vector<int> ans(n);
  for (int i = 0; i < n; i++) ans[i] = round(fa[i].real());
  return ans;
}
}  // namespace fft
```

## othermodint.cpp

64 lines

```
template <int p>
struct mod_int {
  ll expo(ll b, ll e) {
    ll ret = 1;
    while (e) {
      if (e % 2) ret = ret * b % p;
      e /= 2, b = b * b % p;
    }
    return ret;
  }
  ll inv(ll b) { return expo(b, p - 2); }
  using m = mod_int;
  int v;
  mod_int() : v(0) {}
  mod_int(ll v_) {
    if (v_ >= p or v_ <= -p) v_ %= p;
    if (v_ < 0) v_ += p;
    v = v_;
  }
  m& operator+=(const m& a) {
    v += a.v;
    if (v >= p) v -= p;
    return *this;
  }
  m& operator-=(const m& a) {
    v -= a.v;
    if (v < 0) v += p;
    return *this;
  }
  m& operator*=(const m& a) {
    v = v * ll(a.v) % p;
    return *this;
  }
  m& operator/=(const m& a) {
    v = v * inv(a.v) % p;
    return *this;
  }
  m operator-() { return m(-v); }
  m& operator^=(ll e) {
    if (e < 0) {
      v = inv(v);
      e = -e;
    }
    v = expo(v, e);
    // possivel otimizacao:
    // cuidado com 0^0
    // v = expo(v, e%(p-1));
    return *this;
  }
  bool operator==(const m& a) { return v == a.v; }
  bool operator!=(const m& a) { return v != a.v; }
  friend istream& operator>>(istream& in, m& a) {
    ll val;
    in >> val;
    a = m(val);
    return in;
  }
  friend ostream& operator<<(ostream& out, m a) { return out <<
       a.v; }
  friend m operator+(m a, m b) { return a += b; }
  friend m operator-(m a, m b) { return a -= b; }
  friend m operator*(m a, m b) { return a *= b; }
  friend m operator/(m a, m b) { return a /= b; }
  friend m operator^(m a, ll e) { return a ^= e; }
};
```

## fraction.cpp

44 lines

```
struct fraction {
  int x, y;  // x / y
  fraction() {}
  fraction(int x, int y) : x(x), y(y) {}
  bool operator==(fraction o) { return (x * o.y == o.x * y); }
  bool operator!=(fraction o) { return (x * o.y != o.x * y); }
  bool operator>(fraction o) { return (x * o.y > o.x * y); }
  bool operator>=(fraction o) { return (x * o.y >= o.x * y); }
  bool operator<(fraction o) { return (x * o.y < o.x * y); }
  bool operator<=(fraction o) { return (x * o.y <= o.x * y); }
  fraction operator+(fraction o) {
    fraction ans;
    ans.y = (y == o.y) ? y : y * o.y;
    ans.x = (x) * (ans.y / y) + (o.x) * (ans.y / o.y);
    // ans.simplify();
    return ans;
  }
  fraction operator*(fraction o) {
    fraction ans;
    ans.x = x * o.x;
    ans.y = y * o.y;
    // ans.simplify();
    return ans;
  }
  fraction inv() {
    fraction ans = fraction(x, y);
    swap(ans.x, ans.y);
    return ans;
  }
  fraction neg() {
    fraction ans = fraction(x, y);
    ans.x *= -1;
    return ans;
  }
  void simplify() {
    if (abs(x) > 1e9 || abs(y) > 1e9)  // slow simplification
    {
      int g = __gcd(y, x);
      x /= g;
      y /= g;
    }
  }
  // substraction and division can be easily done
};
```

## totient.cpp

20 lines

```
int phi[MAXN];
void calc() {
  for (int i = 0; i < MAXN; i++) phi[i] = i;
  for (int i = 2; i < MAXN; i++) {
    if (phi[i] == i) {
      for (int j = i; j < MAXN; j += i) phi[j] -= phi[j] / i;
    }
  }
}
int calc_phi(int n) {
  int ans = n;
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      while (n % i == 0) n /= i;
      ans -= ans / i;
    }
  }
  if (n > 1) ans -= ans / n;
  return ans;
}
```

## pollardrho.cpp

62 lines

```
#define int __int128
namespace pollard_rho {
int multiplicate(int x, int y, int m) { return (x * y) % m; }
int modpow(int x, int y, int m) {
  int z = 1;
  while (y) {
    if (y & 1) z = (z * x) % m;
    x = (x * x) % m;
    y >>= 1;
  }
  return z;
}
bool is_composite(int n, int a, int d, int s) {
  int x = modpow(a, d, n);
  if (x == 1 or x == n - 1) return false;
  for (int r = 1; r < s; r++) {
    x = multiplicate(x, x, n);
    if (x == n - 1LL) return false;
  }
  return true;
};
int miller_rabin(int n) {
  if (n < 2) return false;
  int r = 0, d = n - 1LL;
```

```
  while ((d & 1LL) == 0) {
    d >>= 1;
    r++;
  }
  for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
    if (n == a) return true;
    if (is_composite(n, a, d, r)) return false;
  }
  return true;
}
int f(int x, int m) { return multiplicate(x, x, m) + 1; }
int rho(int n) {
  int x0 = 1, t = 0, prd = 2;
  int x = 0, y = 0, q;
  while (t % 40 || __gcd(prd, n) == 1) {
    if (x == y) {
      x0++;
      x = x0;
      y = f(x, n);
    }
    q = multiplicate(prd, max(x, y) - min(x, y), n);
    if (q != 0) prd = q;
    x = f(x, n);
    y = f(y, n);
    y = f(y, n);
    t++;
  }
  return __gcd(prd, n);
}
vector<int> fact(int n) {
  if (n == 1) return {};
  if (miller_rabin(n)) return {n};
  int x = rho(n);
  auto l = fact(x), r = fact(n / x);
  l.insert(l.end(), r.begin(), r.end());
  return l;
}
}  // namespace pollard_rho
```

## primefactors.cpp

14 lines

```
vector<int> facts;
void primefactors(int n) {
  while (n % 2 == 0) {
    facts.pb(2);
    n = n / 2;
  }
  for (int i = 3; i <= sqrt(n); i += 2) {
    while (n % i == 0) {
      facts.pb(i);
      n = n / i;
    }
  }
  if (n > 2) facts.pb(n);
}
```

## xorbasis.cpp

57 lines

```
int modpow(int a, int b) {
  int res = 1;
  while (b > 0) {
    if (b & 1) res = (res * a) % mod;
    a = (a * a) % mod;
    b >>= 1;
  }
  return res;
}
int all, qt;
int dp[33];
void add(int x) {
  all++;
  for (int i = 32; i >= 0; i--) {
    if (x & (1ll << i)) {
      if (dp[i] == 0) {
        dp[i] = x;
        qt++;
        return;
      }
      x ^= dp[i];
    }
  }
}
int get(int x)  // qual o x-esimo menor valor de xor de uma
    subsequencia
{
  int tot = (1ll << qt), ans = 0;
  for (int i = 32; i >= 0; i--) {
    if (dp[i] > 0) {
      int d = tot / 2;
      if (d < x && !(ans & (1ll << i)))
        ans ^= dp[i];
      else if (d >= x && (ans & (1ll << i)))
        ans ^= dp[i];
      if (d < x) x -= d;
      tot /= 2;
    }
  }
  return ans;
}
bool check(int x)  // se existe pelo menos uma subsequencia com
     xor x
{
  for (int i = 32; i >= 0; i--) {
    if (x & (1ll << i)) {
      if (!dp[i]) return 0;
      x ^= dp[i];
    }
  }
  return 1;
}
int count(int x)  // quantas subsequencias tem xor x
{
  if (!check(x)) return 0;
  return modpow(2, all - qt);
}
int x = get(1ll << qt);  // maior xor possivel de uma
    subsequencia
int y = get(1);          // maior xor possivel != 0 (o 0 sempre
     eh possivel - subsequencia vazia)
```

## mobius.cpp

43 lines

```
int lpf[MAXN];
int mobius[MAXN];
int mp[MAXN];
vector<int> d[MAXN];
void calc_lpf() {
  for (int i = 2; i < MAXN; i++) {
    if (!lpf[i]) {
      for (int j = i; j < MAXN; j += i) {
        if (!lpf[j]) lpf[j] = i;
      }
    }
  }
}
void calc() {
  for (int i = 2; i < MAXN; i++)  // divisores
  {
    for (int j = i; j < MAXN; j += i) d[j].pb(i);
  }
  calc_lpf();
  mobius[1] = 1;
  for (int i = 2; i < MAXN; i++) {
    if (lpf[i / lpf[i]] == lpf[i])
      mobius[i] = 0;
    else
      mobius[i] = -1 * mobius[i / lpf[i]];
  }
}
void add(int x, int dd)  // adiciona dd em todos os val[i] que
    gcd(x, i) > 1
{
  for (auto const &i : d[x]) mp[i] += dd;
}
int sum(int x)  // valor de val[x]
{
  int ans = 0;
  for (auto const &i : d[x]) ans += (mobius[i] * -1 * mp[i]);
  return ans;
}
// mobius/inclusao-exclusao com os fatores primos
// a funcao de mobius eh definida como:
// mi(n) = 1, se n eh um square-free com um numero par de
    fatores primos
// mi(n) = -1, se n eh um square-free com um numero impar de
    fatores primos
// mi(n) = 0, caso nenhum dos dois
// square-free = nenhum fator primo aparece duas vezes ou mais
```

## segmentedsieve.cpp

18 lines

```
vector<int> prime;
void segmentedsieve(int l, int r) {
  int lim = sqrt(r);
  vector<bool> mark(lim + 1, false);
  vector<int> primes;
  for (int i = 2; i <= lim; ++i) {
    if (!mark[i]) {
      primes.pb(i);
      for (int j = i * i; j <= lim; j += i) mark[j] = true;
    }
  }
  vector<bool> isprime(r - l + 1, true);
  for (int i : primes)
    for (int j = max(i * i, (l + i - 1) / i * i); j <= r; j +=
        i) isprime[j - l] = false;
  if (l == 1) isprime[0] = false;
  for (int i = 0; i < isprime.size(); i++)
    if (isprime[i]) prime.pb(i + l);
}
```

## xortrie.cpp

51 lines

```
struct node {
  int me, cnt, id;
  int down[2];
  node(int c = 0) : me(c) {
    cnt = 0;
    id = -1;
    fill(begin(down), end(down), -1);
  }
};
struct trie_xor {
  vector<node> t;
  trie_xor() { t.resize(1); }
  void add(int n, int id) {
    int v = 0;
```

```
    t[v].cnt++;
    for (int i = 30; i >= 0; i--) {
      int bit = (n & (1 << i)) ? 1 : 0;
      if (t[v].down[bit] == -1) {
        t[v].down[bit] = t.size();
        t.emplace_back(bit);
      }
      v = t[v].down[bit];
      t[v].cnt++;
    }
    t[v].id = id;
  }
  void rem(int n, int id) {
    int v = 0;
    t[v].cnt--;
    for (int i = 30; i >= 0; i--) {
      int bit = (n & (1 << i)) ? 1 : 0;
      v = t[v].down[bit];
      t[v].cnt--;
    }
  }
  int qry(int n)  // maximum xor with n
  {
    if (t[0].cnt == 0)  // no element
      return -1;
    int v = 0;
    for (int i = 30; i >= 0; i--) {
      int bit = (n & (1 << i)) ? 0 : 1;
      int nxt = t[v].down[bit];
      if (nxt != -1 && t[nxt].cnt > 0)
        v = nxt;
      else
        v = t[v].down[bit ^ 1];
    }
    return t[v].id;
  }
};
```

### matrix.cpp

26 lines

```
namespace matrix {
vector<vector<int>> ans;
int multi(int x, int y) { return (x * y) % mod; }
int sum(int a, int b) { return (a + b >= mod) ? a + b - mod : a
     + b; }
vector<vector<int>> multiply(vector<vector<int>> a, vector<
    vector<int>> b) {
  vector<vector<int>> res(a[0].size(), vector<int>(b[0].size())
      );
  for (int i = 0; i < a.size(); i++) {
    for (int j = 0; j < b[0].size(); j++) {
      res[i][j] = 0;
      for (int k = 0; k < a[0].size(); k++) res[i][j] = sum(res
          [i][j], multi(a[i][k], b[k][j]));
    }
  }
  return res;
}
vector<vector<int>> expo(vector<vector<int>> mat, int m) {
  ans = vector<vector<int>>(mat.size(), vector<int>(mat[0].size
      ()));
  for (int i = 0; i < mat.size(); i++)
    for (int j = 0; j < mat[0].size(); j++) ans[i][j] = (i == j
        );
  while (m > 0) {
    if (m & 1) ans = multiply(ans, mat);
    m = m / 2;
    mat = multiply(mat, mat);
  }
  return ans;
}
}  // namespace matrix
```

### fwht.cpp

25 lines

```
vector<modint> fwht(char op, vector<modint> f, bool inv = 0) {
  int n = f.size();
  for (int k = 0; (n - 1) >> k; k++) {
    for (int i = 0; i < n; i++) {
      if (i >> k & 1) {
        int j = i ^ (1 << k);
        if (op == '^') f[j] += f[i], f[i] = f[j] - modint(2) *
            f[i];
        if (op == '|') f[i] += modint(inv ? -1 : 1) * f[j];
        if (op == '&') f[j] += modint(inv ? -1 : 1) * f[i];
      }
    }
  }
  if (op == '^' and inv) {
    for (auto &i : f) i /= n;
  }
  return f;
}
vector<modint> conv(char op, vector<modint> a, vector<modint> b
    ) {
  a = fwht(op, a, 0);
  b = fwht(op, b, 0);
  for (int i = 0; i < a.size(); i++) {
    a[i] *= b[i];
  }
  return fwht(op, a, 1);
}
```

### teoremas.md

39 lines

```
# Teorema de Lucas
sejam m e n numeros inteiros nao negativos e p um numero primo
desenvolver n e m na base p
ou seja:
m = m[k]*p^(k) + m[k - 1]*p^(k - 1) + ... + m[0]*p^(0)
n = n[k]*p^(k) + n[k - 1]*p^(k - 1) + ... + n[0]*p^(0)
entao:
ncr(m, n) mod p = produtorio de (ncr(m[i], n[i]) mod p)
dai pra generalizar pro mod 2 eh deboas, pq se tiver um bit
    setado em n[i] que nao ta setado em m[i], entao miou, vai
    dar zero
# Manhattan and Chebyshev distances equivalences
It is well known that given points (x, y) and you need to
    calculate the Manhattan distances between them, instead of
     using:
|x1-x2|+|y1-y2|
you can first convert all points (x, y) into (x+y, x-y) (rotate
     45 degrees) and the distances will become max(|x1-x2|, |
    y1-y2|) (also known as Chebyshev distance).
# Chicken McNugget Theorem
For any two coprime numbers (n > 0, m > 0), the greatest
    integer that cannot be written in the form:
an + bm, (a >= 0, b >= 0)
is (n \* m) - n - m
## Consequence of the theorem
That there are exactly ((n - 1) \* (m - 1)) / 2 positive
    integers which cannot be expressed in the form an + bm, (a
     >= 0, b >= 0)
## Generalization
If n and m are not coprime, so all numbers that are not
    multiples of gcd(n, m) cannot be expressed in the form an
    + bm, (a >= 0, b >= 0)
in addition, you can consider n = (n / gcd(n, m)) and m = (m /
    gcd(n, m)), to find how many multiples of gcd(n, m) cannot
     be expressed, or to find the greatest multiple of gcd(n,
    m) that cannot be expressed
## Considering a > 0, b > 0
Considering (n > 0, m > 0), n and m are coprime:
let y = ((n \* m) + min(n, m)) - 1
The number of positive integers which cannot be expressed
    increases by (y / n)
The number of positive integers which cannot be expressed
    increases by (y / m)
you must not count the multiples of (n \* m) more than once,
    just decrease number of positive integers which cannot be
    expressed by (y / (n \* m))
# Binomial Theorem
### Theorem
$$
(x + y)^n = \sum_{k = 0}^{n} {n \choose k} x^{n - k}y^{k}
$$
in addition, we have:
$$
(x - y)^n = \sum_{k = 0}^{n} (-1)^k {n \choose k} x^{n - k}y^{k
    }
$$
$$
(1 + x)^n = \sum_{k = 0}^{n} {n \choose k} x^{k}
```

# Data structures (3)

### segtree.cpp

25 lines

```
struct segtree {
  int n;
  vector<int> seg;
  int neutral() { return 0; }
  int merge(int a, int b) { return a + b; }
  void build(vector<int> &v) {
    n = 1;
    while (n < v.size()) n <<= 1;
    seg.assign(n << 1, neutral());
    for (int i = 0; i < v.size(); i++) seg[i + n] = v[i];
    for (int i = n - 1; i; i--) seg[i] = merge(seg[i << 1], seg
        [(i << 1) | 1]);
  }
  void upd(int i, int value) {
    seg[i += n] += value;
    for (i >>= 1; i; i >>= 1) seg[i] = merge(seg[i << 1], seg[(
        i << 1) | 1]);
  }
  int qry(int l, int r) {
    int ansl = neutral(), ansr = neutral();
    for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
      if (l & 1) ansl = merge(ansl, seg[l++]);
      if (r & 1) ansr = merge(seg[--r], ansr);
    }
    return merge(ansl, ansr);
  }
};
```

### segtreelazy.cpp

53 lines

```
struct segtree {
  int n;
  vector<int> v;
  vector<int> seg;
  vector<int> lazy;
  segtree(int sz) {
    n = sz;
```

```
    seg.assign(4 * n, 0);
    lazy.assign(4 * n, 0);
  }
  int single(int x) { return x; }
  int neutral() { return 0; }
  int merge(int a, int b) { return a + b; }
  void add(int i, int l, int r, int diff) {
    seg[i] += (r - l + 1) * diff;
    if (l != r) {
      lazy[i << 1] += diff;
      lazy[(i << 1) | 1] += diff;
    }
    lazy[i] = 0;
  }
  void update(int i, int l, int r, int ql, int qr, int diff) {
    if (lazy[i]) add(i, l, r, lazy[i]);
    if (l > r || l > qr || r < ql) return;
    if (l >= ql && r <= qr) {
      add(i, l, r, diff);
      return;
    }
    int mid = (l + r) >> 1;
    update(i << 1, l, mid, ql, qr, diff);
    update((i << 1) | 1, mid + 1, r, ql, qr, diff);
    seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
  }
  int query(int l, int r, int ql, int qr, int i) {
    if (lazy[i]) add(i, l, r, lazy[i]);
    if (l > r || l > qr || r < ql) return neutral();
    if (l >= ql && r <= qr) return seg[i];
    int mid = (l + r) >> 1;
    return merge(query(l, mid, ql, qr, i << 1), query(mid + 1,
        r, ql, qr, (i << 1) | 1));
  }
  void build(int l, int r, int i) {
    if (l == r) {
      seg[i] = single(v[l]);
      return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, i << 1);
    build(mid + 1, r, (i << 1) | 1);
    seg[i] = merge(seg[i << 1], seg[(i << 1) | 1]);
  }
  int qry(int l, int r) { return query(0, n - 1, l, r, 1); }
  void upd(int l, int r, int x) { update(1, 0, n - 1, l, r, x);
       }
};
```

## fenwick.cpp

33 lines

```
struct fenw {
  int n;
  vector<int> bit;
  fenw() {}
  fenw(int sz) {
    n = sz;
    bit.assign(sz + 1, 0);
  }
  int qry(int r)  // query de prefixo a[0] + a[1] + ...  a[r]
  {
    int ret = 0;
    for (int i = r + 1; i > 0; i -= i & -i) ret += bit[i];
    return ret;
  }
  void upd(int r, int x)  // a[r] += x
  {
    for (int i = r + 1; i <= n; i += i & -i) bit[i] += x;
  }
  int bs(int x)  // retorna o maior indice i (i < n) tal que:
       qry(i) < x
  {
    int i = 0, k = 0;
    while (1 << (k + 1) <= n) k++;
    while (k >= 0) {
      int nxt_i = i + (1 << k);
      if (nxt_i <= n && bit[nxt_i] < x) {
        i = nxt_i;
        x -= bit[i];
      }
      k--;
    }
    return i - 1;
  }
};
```

## treap.cpp

93 lines

```
struct treap {
  int data, priority;
  int sz, lazy2;
  bool lazy;
  treap *l, *r, *parent;
};
int size(treap *node) { return (!node) ? 0 : node->sz; }
void recalc(treap *node) {
  if (!node) return;
  node->sz = 1;
  node->parent = 0;
  if (node->l) node->sz += node->l->sz, node->l->parent = node;
  if (node->r) node->sz += node->r->sz, node->r->parent = node;
}
void lazy_propagation(treap *node) {
  if (node == NULL) return;
  if (node->lazy2) {
    if (node->l) node->l->lazy2 += node->lazy2;
    if (node->r) node->r->lazy2 += node->lazy2;
    node->data += node->lazy2;
    node->lazy2 = 0;
  }
  if (node->lazy) {
    swap(node->l, node->r);
    if (node->l) node->l->lazy = !node->l->lazy;
    if (node->r) node->r->lazy = !node->r->lazy;
    node->lazy = 0;
  }
}
void split(treap *t, treap *&l, treap *&r, int n) {
  if (!t) return void(l = r = 0);
  lazy_propagation(t);
  if (size(t->l) >= n)
    split(t->l, l, t->l, n), r = t;
  else
    split(t->r, t->r, r, n - size(t->l) - 1), l = t;
  recalc(t);
}
void merge(treap *&t, treap *l, treap *r) {
  lazy_propagation(l);
  lazy_propagation(r);
  if (!l)
    t = r;
  else if (!r)
    t = l;
  else if (l->priority > r->priority)
    merge(l->r, l->r, r), t = l;
  else
    merge(r->l, l, r->l), t = r;
  recalc(t);
}
void troca(treap *&t, int l, int r, int ll, int rr)  // sap de
     ranges
{
  treap *a0, *a1, *b0, *b1, *c0, *c1, *d0, *d1;
  split(t, a0, a1, l);
  split(a1, b0, b1, r - l + 1);
  ll -= (r + 1);
  rr -= (r + 1);
  split(b1, c0, c1, ll);
  split(c1, d0, d1, rr - ll + 1);
  merge(t, a0, d0);
  merge(t, t, c0);
  merge(t, t, b0);
  merge(t, t, d1);
}
void add(treap *&t, int l, int r) {
  treap *a0, *a1, *b0, *b1;
  split(t, a0, a1, l);
  split(a1, b0, b1, r - l + 1);
  b0->lazy ^= 1;
  b0->lazy2 += 1;
  merge(t, a0, b0);
  merge(t, t, b1);
}
void dfs(treap *t) {
  if (!t) return;
  lazy_propagation(t);
  dfs(t->l);
  solve(t->data);
  dfs(t->r);
}
treap *create_node(int data, int priority) {
  treap *ret = new treap;
  ret->data = data;
  ret->priority = priority;
  ret->l = 0;
  ret->r = 0;
  ret->sz = 1;
  ret->lazy = 0;
  ret->lazy2 = 0;
  ret->parent = 0;
  return ret;
}
```

## colorupdate.cpp

74 lines

```
const int inf = 1e15;
struct color_upd {
#define left fir
#define right sec.fir
#define color sec.sec
  set<pii> ranges;
  vector<pii> erased;
  color_upd(int n)  // inicialmente, todo mundo pintado com a
       cor inf
  {
    // nao usar cores negativas!!!!!!!!!
    ranges.insert({0, {n - 1, inf}});
  }
  int get(int i) {
    auto it = ranges.upper_bound({i, {1e18, 1e18}});
    if (it == ranges.begin()) return -1;
    it--;
    return (*(it)).color;
  }
  void del(int l, int r)  // apaga o intervalo [l, r]
  {
    erased.clear();
```

```
    auto it = ranges.upper_bound({l, {0, 0}});
    if (it != ranges.begin()) {
      it--;
    }
    while (it != ranges.end()) {
      if ((*(it)).left > r)
        break;
      else if ((*(it)).right >= l)
        erased.push_back(*it);
      it++;
    }
    if (erased.size() > 0) {
      int sz = erased.size();
      auto it = ranges.lower_bound({erased[0].left, {0, 0}});
      auto it2 = ranges.lower_bound({erased[sz - 1].left, {0, 0
          }});
      pii ini = *it, fim = *it2;
      it2++;
      ranges.erase(it, it2);
      pii upd1 = {ini.left, {l - 1, ini.color}};
      pii upd2 = {r + 1, {fim.right, fim.color}};
      erased[0].left = max(erased[0].left, l);
      erased[sz - 1].right = min(erased[sz - 1].right, r);
      if (upd1.left <= upd1.right) ranges.insert(upd1);
      if (upd2.left <= upd2.right) ranges.insert(upd2);
    }
  }
  void add(int a, int b, int c) {
    auto it = ranges.lower_bound({a, {b, 0}});
    pii aa = {-1, {-1, -1}};
    pii bb = {-1, {-1, -1}};
    if (it != ranges.end()) {
      if ((*it).color == c && (*it).left == b + 1) {
        aa = *it;
        b = (*it).right;
      }
    }
    if (it != ranges.begin()) {
      it--;
      if ((*it).color == c && (*it).right == a - 1) {
        bb = *it;
        a = (*it).left;
      }
    }
    ranges.erase(aa);
    ranges.erase(bb);
    ranges.insert({a, {b, c}});
  }
  void upd(int a, int b, int c)  // pinta o intervalo [a, b]
       com a cor c
  {
    del(a, b);
    add(a, b, c);
  }
};
```

## bit2d.cpp

51 lines

```
struct bit2d {
  vector<int> ord;
  vector<vector<int>> t;
  vector<vector<int>> coord;
  bit2d(vector<pi> &pts)  // recebe todos os pontos que vao ser
       inseridos pra construir, mas nao insere eles
  {
    sort(pts.begin(), pts.end());
    for (auto const &a : pts) {
      if (ord.empty() || a.fir != ord.back()) ord.pb(a.fir);
    }
    t.resize(ord.size() + 1);
    coord.resize(t.size());
    for (auto &a : pts) {
      swap(a.fir, a.sec);
    }
    sort(pts.begin(), pts.end());
    for (auto &a : pts) {
      swap(a.fir, a.sec);
      for (int on = upper_bound(ord.begin(), ord.end(), a.fir)
           - ord.begin(); on < t.size(); on += on & -on) {
        if (coord[on].empty() || coord[on].back() != a.sec)
          coord[on].push_back(a.sec);
      }
    }
    for (int i = 0; i < t.size(); i++) t[i].assign(coord[i].
        size() + 1, 0);
  }
  void add(int x, int y, int v)  // v[a][b] += v
  {
    for (int xx = upper_bound(ord.begin(), ord.end(), x) - ord.
        begin(); xx < t.size(); xx += xx & -xx) {
      for (int yy = upper_bound(coord[xx].begin(), coord[xx].
          end(), y) - coord[xx].begin(); yy < t[xx].size();
          yy += yy & -yy)
        t[xx][yy] += v;
    }
  }
  int qry(int x, int y)  // soma de todos os v[a][b] com (a <=
       x && b <= y)
  {
    int ans = 0;
    for (int xx = upper_bound(ord.begin(), ord.end(), x) - ord.
        begin(); xx > 0; xx -= xx & -xx) {
      for (int yy = upper_bound(coord[xx].begin(), coord[xx].
          end(), y) - coord[xx].begin(); yy > 0; yy -= yy & -
          yy)
        ans += t[xx][yy];
    }
    return ans;
  }
  int qry2(int x1, int y1, int x2, int y2) {
    return qry(x2, y2) - qry(x2, y1 - 1) - qry(x1 - 1, y2) +
        qry(x1 - 1, y1 - 1);
  }
  void add2(int x1, int y1, int x2, int y2, int v) {
    add(x1, y1, v);
    add(x1, y2 + 1, -v);
    add(x2 + 1, y1, -v);
    add(x2 + 1, y2 + 1, v);
  }
};
```

## mo.cpp

45 lines

```
namespace mo {
struct query {
  int idx, l, r;
};
int block;
vector<query> queries;
vector<int> ans;
// bool cmp(query &x, query &y){  essa funcao de ordenacao pode
     funcionar em caso de TLE
//   int ablock = x.l / MAGIC, bblock = y.l / MAGIC;
//   if (ablock != bblock) return ablock < bblock;
//   if (ablock & 1) return x.r < y.r;
//   return x.r > y.r;
// }
bool cmp(query &x, query &y) {
  if (x.l / block != y.l / block) return x.l / block < y.l /
      block;
  return x.r < y.r;
}
void run() {
  block = (int)sqrt(n);
  sort(queries.begin(), queries.end(), cmp);
  ans.resize(queries.size());
  int cl = 0, cr = -1, sum = 0;
  auto add = [&](int x) { sum += x; };
  auto rem = [&](int x) { sum -= x; };
  for (int i = 0; i < queries.size(); i++) {
    while (cl > queries[i].l) {
      cl--;
      add(v[cl]);
    }
    while (cr < queries[i].r) {
      cr++;
      add(v[cr]);
    }
    while (cl < queries[i].l) {
      rem(v[cl]);
      cl++;
    }
    while (cr > queries[i].r) {
      rem(v[cr]);
      cr--;
    }
    ans[queries[i].idx] = sum;
  }
}
}  // namespace mo
```

## segtree2d.cpp

33 lines

```
struct segtree2d {
  int n, m;
  vector<vector<int>> seg;
  int neutral() { return 0; }
  int merge(int a, int b) { return a + b; }
  segtree2d(int nn, int mm) {
    n = nn, m = mm;
    seg = vector<vector<int>>(2 * n, vector<int>(2 * m, neutral
        ()));
  }
  int qry(int x1, int y1, int x2, int y2) {
    int ret = neutral();
    int y3 = y1 + m, y4 = y2 + m;
    for (x1 += n, x2 += n; x1 <= x2; ++x1 /= 2, --x2 /= 2) {
      for (y1 = y3, y2 = y4; y1 <= y2; ++y1 /= 2, --y2 /= 2) {
        if (x1 % 2 == 1 and y1 % 2 == 1) ret = merge(ret, seg[
            x1][y1]);
        if (x1 % 2 == 1 and y2 % 2 == 0) ret = merge(ret, seg[
            x1][y2]);
        if (x2 % 2 == 0 and y1 % 2 == 1) ret = merge(ret, seg[
            x2][y1]);
        if (x2 % 2 == 0 and y2 % 2 == 0) ret = merge(ret, seg[
            x2][y2]);
      }
    }
    return ret;
  }
  void upd(int x, int y, int val) {
    int y2 = y += m;
    for (x += n; x; x /= 2, y = y2) {
      if (x >= n)
        seg[x][y] = val;
      else
        seg[x][y] = merge(seg[2 * x][y], seg[2 * x + 1][y]);
```

```
      while (y /= 2) seg[x][y] = merge(seg[x][2 * y], seg[x][2
          * y + 1]);
    }
  }
};
```

### persistentseg.cpp

37 lines

```
struct node {
  int item, l, r;
  node() {}
  node(int l, int r, int item) : l(l), r(r), item(item) {}
};
int n, q;
vector<node> seg;
vector<int> roots;
void init() { seg.resize(1); }
int newleaf(int vv) {
  int p = seg.size();
  seg.pb(node(0, 0, vv));
  return p;
}
int newpar(int l, int r) {
  int p = seg.size();
  seg.pb(node(l, r, seg[l].item + seg[r].item));
  return p;
}
int upd(int i, int l, int r, int pos) {
  if (l == r) return newleaf(seg[i].item + 1);
  int mid = (l + r) >> 1;
  if (pos <= mid) return newpar(upd(seg[i].l, l, mid, pos), seg
      [i].r);
  return newpar(seg[i].l, upd(seg[i].r, mid + 1, r, pos));
}
int build(int l, int r) {
  if (l == r) return newleaf(0);
  int mid = (l + r) >> 1;
  return newpar(build(l, mid), build(mid + 1, r));
}
int qry(int vl, int vr, int l, int r, int k) {
  if (l == r) return l;
  int mid = (l + r) >> 1;
  int c = seg[seg[vr].l].item - seg[seg[vl].l].item;
  if (c >= k) return qry(seg[vl].l, seg[vr].l, l, mid, k);
  return qry(seg[vl].r, seg[vr].r, mid + 1, r, k - c);
}
```

### rmq.cpp

21 lines

```
struct rmq {
  int n;
  vector<vector<pi>> m;
  vector<int> log;
  rmq() {}
  rmq(vector<pi> &v) {
    n = v.size();
    log.resize(n + 1);
    log[1] = 0;
    for (int i = 2; i <= n; i++) log[i] = log[i / 2] + 1;
    int sz = log[n] + 2;
    m = vector<vector<pi>>(sz, vector<pi>(n + 1));
    for (int i = 0; i < n; i++) {
      m[0][i] = v[i];
    }
    for (int j = 1; j < sz; j++) {
      for (int i = 0; i + (1 << j) <= n; i++) m[j][i] = min(m[j
          - 1][i], m[j - 1][i + (1 << (j - 1))]);
    }
  }
  int qry(int a, int b) { return min(m[log[b - a + 1]][a], m[
      log[b - a + 1]][b - (1 << log[b - a + 1]) + 1]).second;
      }
};
```

### binarylifting.cpp

23 lines

```
item st[MAXN][21];
for (int i = 0; i < n; i++) {
  st[i][0].nxt = min(i + 1, n - 1);
  st[i][0].sum = v[st[i][0].nxt];
}
for (int i = 1; i < 21; i++) {
  for (int v = 0; v < n; v++) {
    st[v][i].nxt = st[st[v][i - 1].nxt][i - 1].nxt;
    st[v][i].sum = st[v][i - 1].sum + st[st[v][i - 1].nxt][i -
        1].sum;
  }
}
while (q--) {
  int l, r;
  cin >> l >> r;
  int ans = v[l], len = r - l;
  for (int i = 20; i >= 0; i--) {
    if (len & (1 << i)) {
      ans += st[l][i].sum;
      l = st[l][i].nxt;
    }
  }
  cout << ans << endl;
}
```

### minqueue.cpp

19 lines

```
namespace min_queue {
deque<pi> q;
int l, r;

void init() {
  l = r = 1;
  q.clear();
}
void push(int v) {
  while (!q.empty() && v < q.back().fir) q.pop_back();
  q.pb({v, r});
  r++;
}
void pop() {
  if (!q.empty() && q.front().sec == l) q.pop_front();
  l++;
}
int getmin() { return q.front().fir; }
}  // namespace min_queue
```

# Graph (4)

### floydwarshall.cpp

21 lines

```
int dist[MAXN][MAXN];
void floyd_warshall() {
  for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
      }
    }
  }
}
```

```
void initialize() {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (i == j) {
        dist[i][j] = 0;
      } else {
        dist[i][j] = INF;
      }
    }
  }
}
```

### centroiddecomp.cpp

68 lines

```
int n, k, resp;
vector<int> adj[MAXN];
vector<int> cnt;
namespace cd {
int sz;
vector<int> subtree_size;
vector<bool> visited;
void dfs(int s, int f) {
  sz++;
  subtree_size[s] = 1;
  for (auto const &v : adj[s]) {
    if (v != f && !visited[v]) {
      dfs(v, s);
      subtree_size[s] += subtree_size[v];
    }
  }
}
int get_centroid(int s, int f) {
  bool is_centroid = true;
  int heaviest_child = -1;
  for (auto const &v : adj[s]) {
    if (v != f && !visited[v]) {
      if (subtree_size[v] > sz / 2) is_centroid = false;
      if (heaviest_child == -1 || subtree_size[v] >
          subtree_size[heaviest_child]) heaviest_child = v;
    }
  }
  return (is_centroid && sz - subtree_size[s] <= sz / 2) ? s :
      get_centroid(heaviest_child, s);
}
void dfs2(int s, int f, int d) {
  while (d >= cnt.size()) cnt.pb(0);
  cnt[d]++;
  for (auto const &v : adj[s]) {
    if (v != f && !visited[v]) dfs2(v, s, d + 1);
  }
}
void solve(int s) {
  vector<int> tot;
  for (auto const &v : adj[s]) {
    if (visited[v]) continue;
    cnt.clear();
    dfs2(v, s, 1);
    for (int i = 1; i < cnt.size(); i++) {
      if (k - i < tot.size() && k - i >= 1) resp += (cnt[i] *
          tot[k - i]);
    }
    for (int i = 1; i < cnt.size(); i++) {
      while (i >= tot.size()) tot.pb(0);
      tot[i] += cnt[i];
    }
  }
  if (k < tot.size()) resp += tot[k];
}
int decompose_tree(int s) {
```

```
  sz = 0;
  dfs(s, s);
  int cend_tree = get_centroid(s, s);
  visited[cend_tree] = true;
  solve(cend_tree);
  for (auto const &v : adj[cend_tree]) {
    if (!visited[v]) decompose_tree(v);
  }
  return cend_tree;
}
void init() {
  subtree_size.resize(n);
  visited.resize(n);
  decompose_tree(0);
}
}  // namespace cd
```

## dsu.cpp

24 lines

```
struct dsu {
  int tot;
  vector<int> parent;
  vector<int> sz;
  dsu(int n) {
    parent.resize(n);
    sz.resize(n);
    tot = n;
    for (int i = 0; i < n; i++) {
      parent[i] = i;
      sz[i] = 1;
    }
  }
  int find_set(int i) { return parent[i] = (parent[i] == i) ? i
       : find_set(parent[i]); }
  void make_set(int x, int y) {
    x = find_set(x), y = find_set(y);
    if (x != y) {
      if (sz[x] > sz[y]) swap(x, y);
      parent[x] = y;
      sz[y] += sz[x];
      tot--;
    }
  }
};
```

## dsubipartido.cpp

41 lines

```
struct dsu {
  vector<pi> parent;
  vector<int> rank;
  vector<int> bipartite;
  dsu(int n) {
    parent.resize(n);
    rank.resize(n);
    bipartite.resize(n);
    for (int v = 0; v < n; v++) {
      parent[v] = {v, 0};
      rank[v] = 0;
      bipartite[v] = 1;
    }
  }
  dsu() {}
  pi find_set(int v) {
    if (v != parent[v].fir) {
      int parity = parent[v].sec;
      parent[v] = find_set(parent[v].fir);
      parent[v].sec ^= parity;
    }
    return parent[v];
  }
  void add_edge(int a, int b) {
    pi pa = find_set(a);
    a = pa.fir;
    int x = pa.sec;
    pi pb = find_set(b);
    b = pb.fir;
    int y = pb.sec;
    if (a == b) {
      if (x == y) bipartite[a] = 0;
    } else {
      if (rank[a] < rank[b]) swap(a, b);
      parent[b] = {a, x ^ y ^ 1};
      bipartite[a] &= bipartite[b];
      if (rank[a] == rank[b]) rank[a]++;
    }
  }
  bool is_bipartite(int v) { return bipartite[find_set(v).fir];
       }
};
```

## cycledetection.cpp

26 lines

```
int n, m, idx;
vector<int> cycles[MAXN];
vector<int> adj[MAXN];
int color[MAXN];
int parent[MAXN];
int ans[MAXN];
void dfs(int u, int p) {  // chama dfs a partir de qm tem cor 0
  if (color[u] == 2) return;
  if (color[u] == 1) {
    idx++;
    int curr = p;
    ans[curr] = idx;
    cycles[idx].pb(curr);
    while (curr != u) {
      curr = parent[curr];
      cycles[idx].pb(curr);
      ans[curr] = idx;
    }
    return;
  }
  parent[u] = p;
  color[u] = 1;
  for (auto const &v : adj[u])
    if (v != parent[u]) dfs(v, u);
  color[u] = 2;
}
```

## blockcuttree.cpp

74 lines

```
struct block_cut_tree {
  // Se art[i] >= 1, i eh ponto de articulacao
  // tree - eh a propia block-cut tree
  // pos[i] responde a qual vertice da arvore vertice i
       pertence
  vector<vector<int>> g, blocks, tree;
  vector<vector<pi>> edgblocks;  // sao as arestas do bloco i
  stack<int> s;
  stack<pi> s2;
  vector<int> id, art, pos;

  block_cut_tree(vector<vector<int>> g_) : g(g_) {
    int n = g.size();
    id.resize(n, -1), art.resize(n), pos.resize(n);
    build();
  }
  int dfs(int i, int &t, int p = -1) {
    int lo = id[i] = t++;
    s.push(i);
    if (p != -1) {
      s2.emplace(i, p);
    }
    for (int j : g[i]) {
      if (j != p and id[j] != -1) s2.emplace(i, j);
    }
    for (int j : g[i]) {
      if (j != p) {
        if (id[j] == -1) {
          int val = dfs(j, t, i);
          lo = min(lo, val);
          if (val >= id[i]) {
            art[i]++;
            blocks.emplace_back(1, i);
            while (blocks.back().back() != j) {
              blocks.back().pb(s.top());
              s.pop();
            }
            edgblocks.emplace_back(1, s2.top());
            s2.pop();
            pi aux = {j, i};
            while (edgblocks.back().back() != aux) {
              edgblocks.back().pb(s2.top());
              s2.pop();
            }
          }
          // if (val > id[i]) aresta i-j eh ponte
        } else {
          lo = min(lo, id[j]);
        }
      }
    }
    if (p == -1 and art[i]) {
      art[i]--;
    }
    return lo;
  }
  void build() {
    int t = 0;
    for (int i = 0; i < g.size(); i++) {
      if (id[i] == -1) dfs(i, t, -1);
    }
    tree.resize(blocks.size());
    for (int i = 0; i < g.size(); i++) {
      if (art[i]) pos[i] = tree.size(), tree.emplace_back();
    }
    for (int i = 0; i < blocks.size(); i++) {
      for (int j : blocks[i]) {
        if (!art[j])
          pos[j] = i;
        else
          tree[i].pb(pos[j]), tree[pos[j]].pb(i);
      }
    }
  }
};
```

## bridges.cpp

21 lines

```
nt n, m, timer;
vector<pi> edges;
vector<bool> is_bridge;
vector<pi> adj[MAXN];
int tin[MAXN];
int low[MAXN];            // memset -1
bool vis[MAXN];           // memset -1
void dfs(int v, int p) {  // chama de quem nao foi vis ainda
```

```
  vis[v] = true;
  tin[v] = timer, low[v] = timer++;
  for (auto const &u : adj[v]) {
    if (u.fir == p) continue;
    if (vis[u.fir]) {
      low[v] = min(low[v], tin[u.fir]);
      continue;
    }
    dfs(u.fir, v);
    low[v] = min(low[v], low[u.fir]);
    if (low[u.fir] > tin[v]) is_bridge[u.sec] = 1;
  }
}
```

## dinic.cpp

83 lines

```
#define INF 1e9
struct edge {
  int to, from, flow, capacity, id;
};
struct dinic {
  int n, src, sink;
  vector<vector<edge>> adj;
  vector<int> level;
  vector<int> ptr;

  dinic(int sz) {
    n = sz;
    adj.resize(n);
    level.resize(n);
    ptr.resize(n);
  }
  void add_edge(int a, int b, int c, int id) {
    adj[a].pb({b, (int)adj[b].size(), c, c, id});
    adj[b].pb({a, (int)adj[a].size() - 1, 0, 0, id});
  }
  bool bfs() {
    level.assign(n, -1);
    level[src] = 0;
    queue<int> q;
    q.push(src);
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (auto at : adj[u]) {
        if (at.flow && level[at.to] == -1) {
          q.push(at.to);
          level[at.to] = level[u] + 1;
        }
      }
    }
    return level[sink] != -1;
  }
  int dfs(int u, int flow) {
    if (u == sink || flow == 0) return flow;
    for (int &p = ptr[u]; p < adj[u].size(); p++) {
      edge &at = adj[u][p];
      if (at.flow && level[u] == level[at.to] - 1) {
        int kappa = dfs(at.to, min(flow, at.flow));
        at.flow -= kappa;
        adj[at.to][at.from].flow += kappa;
        if (kappa != 0) return kappa;
      }
    }
    return 0;
  }
  int run() {
    int max_flow = 0;
    while (bfs()) {
      ptr.assign(n, 0);
      while (1) {
        int flow = dfs(src, INF);
        if (flow == 0) break;
        max_flow += flow;
      }
    }
    return max_flow;
  }
  vector<pii> cut_edges()  // arestas do corte minimo
  {
    bfs();
    vector<pii> ans;
    for (int i = 0; i < n; i++) {
      for (auto const &j : adj[i]) {
        if (level[i] != -1 && level[j.to] == -1 && j.capacity >
            0) ans.pb({j.capacity, {i, j.to}});
      }
    }
    return ans;
  }
  vector<int> flow_edges(int n, int m)  // fluxo em cada aresta
      , na ordem da entrada
  {
    vector<int> ans(m);
    for (int i = 0; i < n; i++) {
      for (auto const &j : adj[i])
        if (!j.capacity) ans[j.id] = j.flow;
    }
    return ans;
  }
};
```

## hopcroftkarp.cpp

137 lines

```
#define INF 1e9
struct hopcroft_karp {
  vector<int> match;
  vector<int> dist;
  vector<vector<int>> adj;
  int n, m, t;
  hopcroft_karp(int a, int b) {
    n = a, m = b;
    t = n + m + 1;
    match.assign(t, n + m);
    dist.assign(t, 0);
    adj.assign(t, vector<int>{});
  }
  void add_edge(int u, int v) {
    adj[u].pb(v);
    adj[v].pb(u);
  }
  bool bfs() {
    queue<int> q;
    for (int u = 0; u < n; u++) {
      if (match[u] == n + m)
        dist[u] = 0, q.push(u);
      else
        dist[u] = INF;
    }
    dist[n + m] = INF;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      if (dist[u] < dist[n + m]) {
        for (auto const &v : adj[u]) {
          if (dist[match[v]] == INF) {
            dist[match[v]] = dist[u] + 1;
            q.push(match[v]);
          }
        }
      }
    }
    return dist[n + m] < INF;
  }
  bool dfs(int u) {
    if (u < n + m) {
      for (auto const &v : adj[u]) {
        if (dist[match[v]] == dist[u] + 1 && dfs(match[v])) {
          match[v] = u;
          match[u] = v;
          return true;
        }
      }
      dist[u] = INF;
      return false;
    }
    return true;
  }
  vector<pi> run() {
    int cnt = 0;
    while (bfs())
      for (int u = 0; u < n; u++)
        if (match[u] == n + m && dfs(u)) cnt++;
    vector<pi> ans;
    for (int v = n; v < n + m; v++)
      if (match[v] < n + m) ans.pb({match[v], v});
    return ans;
  }
  vector<int> mvc()  // minimum vertex cover
  {
    vector<pi> ans = run();
    vector<bool> vis(n + m, 0);
    for (int i = 0; i < n; i++) {
      if (match[i] == n + m) {
        queue<int> q;
        q.push(i);
        while (!q.empty()) {
          int x = q.front();
          q.pop();
          vis[x] = 1;
          for (auto const &y : adj[x]) {
            if (!vis[y]) {
              vis[y] = 1;
              q.push(match[y]);
            }
          }
        }
      }
    }
    vector<int> vc;
    for (int i = 0; i < n; i++) {
      if (!vis[i]) vc.pb(i);
    }
    for (int i = n; i < n + m; i++) {
      if (vis[i]) vc.pb(i);
    }
    return vc;
  }
  vector<pi> mec()  // minimum edge cover
  {
    vector<pi> ans = run();
    for (int i = 0; i < n + m; i++) {
      if (match[i] == n + m && adj[i].size() > 0) {
        if (i < n)
          ans.pb({i, adj[i][0]});
        else
          ans.pb({adj[i][0], i});
```

```
      }
    }
    return ans;
  }
};
// minimum path cover on dag
// minimum set of paths such that each of the vertices belongs
    to exactly one path
vector<vector<int>> mpc(int n, vector<pi> &e) {
  hopcroft_karp h(n, n);
  for (auto const &i : e) h.add_edge(i.fir, n + i.sec);
  vector<pi> mat = h.run();
  vector<int> prv(n, -1);
  vector<int> nxt(n, -1);
  for (int i = 0; i < mat.size(); i++) {
    nxt[mat[i].fir] = mat[i].sec - n;
    prv[mat[i].sec - n] = mat[i].fir;
  }
  vector<vector<int>> ans;
  for (int i = 0; i < n; i++) {
    if (prv[i] == -1 && nxt[i] == -1) {
      ans.pb({i});
    } else if (prv[i] == -1) {
      vector<int> curr;
      int x = i;
      while (1) {
        curr.pb(x);
        if (nxt[x] == -1) break;
        x = nxt[x];
      }
      ans.pb(curr);
    }
  }
  return ans;
}
```

## pushrelabel.cpp

93 lines

```
#define INF 1e9
struct edge {
  int dest, back, f, c, id;
};
struct push_relabel {
  int n;
  vector<vector<edge>> g;
  vector<int> ec;
  vector<edge *> cur;
  vector<vector<int>> hs;
  vector<int> H;
  push_relabel(int sz) : g(sz), ec(sz), cur(sz), hs(2 * sz), H(
      sz) { n = sz; }
  void add_edge(int s, int t, int cap, int rcap, int id) {
    if (s == t) return;
    g[s].pb({t, (int)g[t].size(), 0, cap, id});
    g[t].pb({s, (int)g[s].size() - 1, 0, rcap, -1});
  }
  void add_flow(edge &e, int 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;
  }
  int calc(int s, int t) {
    int v = g.size();
    H[s] = v;
    ec[t] = 1;
    vector<int> co(2 * v);
    co[0] = v - 1;
    for (int i = 0; i < v; i++) cur[i] = g[i].data();
    for (edge &e : g[s]) add_flow(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) {
        if (cur[u] == g[u].data() + g[u].size()) {
          H[u] = INF;
          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)
            for (int i = 0; i < v; i++)
              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)
          add_flow(*cur[u], min(ec[u], cur[u]->c));
        else
          ++cur[u];
      }
    }
  }
  vector<int> flow_edges(int m)  // fluxo em cada aresta
  {
    vector<int> ans(m);
    for (int i = 0; i < n; i++) {
      for (auto const &j : g[i]) {
        if (j.id != -1) ans[j.id] = j.f;
      }
    }
    return ans;
  }
};
struct flow_with_demands {
  push_relabel pr;
  vector<int> in, out;
  int n;

  flow_with_demands(int sz) : n(sz), pr(sz + 2), in(sz), out(sz
      ) {}
  void add_edge(int u, int v, int cap, int dem, int id) {
    pr.add_edge(u, v, cap - dem, 0, id);
    out[u] += dem, in[v] += dem;
  }
  int run(int s, int t) {
    pr.add_edge(t, s, INF, 0, -1);
    for (int i = 0; i < n; i++) {
      pr.add_edge(n, i, in[i], 0, -1);
      pr.add_edge(i, n + 1, out[i], 0, -1);
    }
    return pr.calc(n, n + 1);
  }
  bool check()  // todas as constraints foram satisfeitas?
  {
    for (auto const &i : pr.g[n]) {
      if (i.c > 0) return 0;
    }
    return 1;
  }
};
```

## hld.cpp

65 lines

```
struct hld {
  int n, cur_pos;
  segtree seg;
  vector<vector<int>> adj;
  vector<int> parent, depth, heavy, head, pos, sz;
  int dfs(int s) {
    int size = 1, max_c_size = 0;
    for (auto const &c : adj[s]) {
      if (c != parent[s]) {
        parent[c] = s;
        depth[c] = depth[s] + 1;
        int c_size = dfs(c);
        size += c_size;
        if (c_size > max_c_size) max_c_size = c_size, heavy[s]
            = c;
      }
    }
    return sz[s] = size;
  }
  void decompose(int s, int h) {
    head[s] = h;
    pos[s] = cur_pos++;
    if (heavy[s] != -1) decompose(heavy[s], h);
    for (int c : adj[s]) {
      if (c != parent[s] && c != heavy[s]) decompose(c, c);
    }
  }
  hld(vector<vector<int>> &g) {
    n = g.size();
    adj = g;
    seg = segtree(n);
    parent.assign(n, -1);
    depth.assign(n, -1);
    heavy.assign(n, -1);
    head.assign(n, -1);
    pos.assign(n, -1);
    sz.assign(n, 1);
    cur_pos = 0;
    dfs(0);
    decompose(0, 0);
  }
  int query_path(int a, int b) {
    int res = 0;
    for (; head[a] != head[b]; b = parent[head[b]]) {
      if (depth[head[a]] > depth[head[b]]) swap(a, b);
      res += seg.query(0, n - 1, pos[head[b]], pos[b], 1);
    }
    if (depth[a] > depth[b]) swap(a, b);
    res += seg.query(0, n - 1, pos[a], pos[b], 1);
    return res;
  }
  void update_path(int a, int b, int x) {
    for (; head[a] != head[b]; b = parent[head[b]]) {
      if (depth[head[a]] > depth[head[b]]) swap(a, b);
      seg.update(1, 0, n - 1, pos[head[b]], pos[b], x);
    }
    if (depth[a] > depth[b]) swap(a, b);
    seg.update(1, 0, n - 1, pos[a], pos[b], x);
  }
  void update_subtree(int a, int x) { seg.update(1, 0, n - 1,
      pos[a], pos[a] + sz[a] - 1, x); }
  int query_subtree(int a) { return seg.query(0, n - 1, pos[a],
      pos[a] + sz[a] - 1, 1); }
  int lca(int a, int b) {
    if (pos[a] < pos[b]) swap(a, b);
    return (head[a] == head[b]) ? b : lca(parent[head[a]], b);
  }
```

```
};
```

## hldedge.cpp

67 lines

```
namespace hld {
int cur_pos;
vector<int> parent, depth, heavy, head, pos, sz, up;
int dfs(int s) {
  int size = 1, max_c_size = 0;
  for (auto const &c : adj[s]) {
    if (c.fir != parent[s]) {
      parent[c.fir] = s;
      depth[c.fir] = depth[s] + 1;
      int c_size = dfs(c.fir);
      size += c_size;
      if (c_size > max_c_size) max_c_size = c_size, heavy[s] =
          c.fir;
    }
  }
  return sz[s] = size;
}
void decompose(int s, int h) {
  head[s] = h;
  pos[s] = cur_pos++;
  seg::v[pos[s]] = up[s];
  for (auto const &c : adj[s]) {
    if (c.fir != parent[s] && c.fir == heavy[s]) {
      up[c.fir] = c.sec;
      decompose(heavy[s], h);
    }
  }
  for (auto const &c : adj[s]) {
    if (c.fir != parent[s] && c.fir != heavy[s]) {
      up[c.fir] = c.sec;
      decompose(c.fir, c.fir);
    }
  }
}
void init() {
  parent.assign(MAXN, -1);
  depth.assign(MAXN, -1);
  heavy.assign(MAXN, -1);
  head.assign(MAXN, -1);
  pos.assign(MAXN, -1);
  sz.assign(MAXN, 1);
  up.assign(MAXN, 0);
  cur_pos = 0;
  dfs(0);
  decompose(0, 0);
  seg::build(0, n - 1, 1);
}
int query_path(int a, int b) {
  int res = -1;
  for (; head[a] != head[b]; b = parent[head[b]]) {
    if (depth[head[a]] > depth[head[b]]) swap(a, b);
    res = max(res, seg::query(0, n - 1, pos[head[b]], pos[b],
        1));
  }
  if (depth[a] > depth[b]) swap(a, b);
  res = max(res, seg::query(0, n - 1, pos[a] + 1, pos[b], 1));
  return res;
}
void update_path(int a, int b, int x) {
  for (; head[a] != head[b]; b = parent[head[b]]) {
    if (depth[head[a]] > depth[head[b]]) swap(a, b);
    seg::update(1, 0, n - 1, pos[head[b]], pos[b], x);
  }
  if (depth[a] > depth[b]) swap(a, b);
  seg::update(1, 0, n - 1, pos[a] + 1, pos[b], x);
}
void update_subtree(int a, int x) { seg::update(1, 0, n - 1,
    pos[a] + 1, pos[a] + sz[a] - 1, x); }
int query_subtree(int a, int x) { return seg::query(0, n - 1,
    pos[a] + 1, pos[a] + sz[a] - 1, 1); }
}  // namespace hld
```

## strongorientation.cpp

25 lines

```
// encontrar uma orientacao para as arestas tal que o numero
// minimo de scc eh o menor possivel
// numero minimo de scc = numero de componentes conexas +
    numero de pontes
int n, m, timer, comps, bridges;
vector<pi> edges;
vector<pi> adj[MAXN];
int tin[MAXN];  // memset -1
int low[MAXN];  // memset -1
bool vis[MAXN];
char orient[MAXN];
void find_bridges(int v) {  // chama se tem tin == -1
  low[v] = timer, tin[v] = timer++;
  for (auto const &p : adj[v]) {
    if (vis[p.sec]) continue;
    vis[p.sec] = true;
    orient[p.sec] = (v == edges[p.sec].first) ? '>' : '<';
    if (tin[p.fir] == -1) {
      find_bridges(p.fir);
      low[v] = min(low[v], low[p.fir]);
      if (low[p.fir] > tin[v]) bridges++;
    } else {
      low[v] = min(low[v], low[p.fir]);
    }
  }
}
```

## twosat.cpp

74 lines

```
struct two_sat {
  int n;
  vector<vector<int>> g, gr;   // gr is the reversed graph
  vector<int> comp, ord, ans;  // comp[v]: ID of the SCC
      containing node v
  vector<bool> vis;
  two_sat() {}
  two_sat(int sz) {
    n = sz;
    g.assign(2 * n, vector<int>());
    gr.assign(2 * n, vector<int>());
    comp.resize(2 * n);
    vis.resize(2 * n);
    ans.resize(2 * n);
  }
  void add_edge(int u, int v) {
    g[u].push_back(v);
    gr[v].push_back(u);
  }
  // int x, bool val: if 'val' is true, we take the variable to
      be x. Otherwise we take it to be x's complement (not x)
      .
  void implies(int i, bool f, int j, bool g)  // a -> b
  {
    add_edge(i + (f ? 0 : n), j + (g ? 0 : n));
    add_edge(j + (g ? n : 0), i + (f ? n : 0));
  }
  void add_clause_or(int i, bool f, int j, bool g)  // At least
      one of them is true
  {
    add_edge(i + (f ? n : 0), j + (g ? 0 : n));
    add_edge(j + (g ? n : 0), i + (f ? 0 : n));
  }
  void add_clause_xor(int i, bool f, int j, bool g)  // only
      one of them is true
  {
    add_clause_or(i, f, j, g);
    add_clause_or(i, !f, j, !g);
  }
  void add_clause_and(int i, bool f, int j, bool g)  // both of
      them have the same value
  {
    add_clause_xor(i, !f, j, g);
  }
  void set(int i, bool f)  // Set a variable
  {
    add_clause_or(i, f, i, f);
  }
  void top_sort(int u) {
    vis[u] = 1;
    for (auto const &v : g[u]) {
      if (!vis[v]) top_sort(v);
    }
    ord.push_back(u);
  }
  void scc(int u, int id) {
    vis[u] = 1;
    comp[u] = id;
    for (auto const &v : gr[u]) {
      if (!vis[v]) scc(v, id);
    }
  }
  bool solve() {
    fill(vis.begin(), vis.end(), 0);
    for (int i = 0; i < 2 * n; i++) {
      if (!vis[i]) top_sort(i);
    }
    fill(vis.begin(), vis.end(), 0);
    reverse(ord.begin(), ord.end());
    int id = 0;
    for (const auto &v : ord) {
      if (!vis[v]) scc(v, id++);
    }
    for (int i = 0; i < n; i++) {
      if (comp[i] == comp[i + n]) return 0;
      ans[i] = (comp[i] > comp[i + n]) ? 1 : 0;
    }
    return 1;
  }
};
```

## sack.cpp

47 lines

```
vector<int> adj[MAXN];
vector<int> v[MAXN];
int c[MAXN];
int cnt[MAXN];
int sz[MAXN];
void dfs_sz(int x, int p) {
  sz[x] = 1;
  for (auto const &i : adj[x]) {
    if (i != p) {
      dfs_sz(i, x);
      sz[x] += sz[i];
    }
  }
}
void modify(int c, int val) { cnt[c] += val; }
void dfs(int x, int p, bool keep) {
  int best = -1, big_child = -1;
```

```
  for (auto const &i : adj[x]) {
    if (i != p && sz[i] > best) {
      best = sz[i];
      big_child = i;
    }
  }
  for (auto const &i : adj[x]) {
    if (i != p && i != big_child) dfs(i, x, 0);
  }
  if (big_child != -1) {
    dfs(big_child, x, 1);
    swap(v[x], v[big_child]);  // O(1)
  }
  v[x].pb(x);
  modify(c[x], 1);  // adiciona
  for (auto const &i : adj[x]) {
    if (i != p && i != big_child) {
      for (auto const &j : v[i]) {
        v[x].pb(j);
        modify(c[j], 1);  // adiciona
      }
    }
  }
  // a cor c aparece cnt[c] vezes na subtree de x
  // dai vc pode fazer algo tendo essa informacao
  // seja responser queries ou algo do tipo aqui
  if (!keep) {
    for (auto const &i : v[x]) modify(c[i], -1);  // remove
  }
}
```

## reroot.cpp

23 lines

```
int n;
vector<int> adj[MAXN];
int sz[MAXN];
int dp[MAXN];
int dfs(int u, int v) {
  sz[u] = 1;
  for (auto const &i : adj[u])
    if (i != v) sz[u] += dfs(i, u);
  return sz[u];
}
void reroot(int u, int v) {
  for (auto const &i : adj[u]) {
    if (i != v) {
      int a = sz[u], b = sz[i];
      dp[i] = dp[u];
      dp[i] -= sz[u], dp[i] -= sz[i];
      sz[u] -= sz[i], sz[i] = n;
      dp[i] += sz[u], dp[i] += sz[i];
      reroot(i, u);
      sz[u] = a, sz[i] = b;
    }
  }
}
```

## hungarian.cpp

45 lines

```
struct hungarian {  // declarar algo como hungarian h(n), n de
     cada lado
  int n, inf;
  vector<vector<int>> a;
  vector<int> u, v, p, way;
  hungarian(int n_) : n(n_), u(n + 1), v(n + 1), p(n + 1), way(
       n + 1) {
    a = vector<vector<int>>(n, vector<int>(n));
    inf = numeric_limits<int>::max();
  }
  void add_edge(int x, int y, int c) { a[x][y] = c; }
  pair<int, vector<int>> run() {
    for (int i = 1; i <= n; i++) {
      p[0] = i;
      int j0 = 0;
      vector<int> minv(n + 1, inf);
      vector<int> used(n + 1, 0);
      do {
        used[j0] = true;
        int i0 = p[j0], j1 = -1;
        int delta = inf;
        for (int j = 1; j <= n; j++) {
          if (!used[j]) {
            int cur = a[i0 - 1][j - 1] - u[i0] - v[j];
            if (cur < minv[j]) minv[j] = cur, way[j] = j0;
            if (minv[j] < delta) delta = minv[j], j1 = j;
          }
        }
        for (int j = 0; j <= n; j++) {
          if (used[j])
            u[p[j]] += delta, v[j] -= delta;
          else
            minv[j] -= delta;
        }
        j0 = j1;
      } while (p[j0] != 0);
      do {
        int j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
      } while (j0);
    }
    vector<int> ans(n);
    for (int j = 1; j <= n; j++) ans[p[j] - 1] = j - 1;
    return make_pair(-v[0], ans);
  }
};
```

## mincostmaxflow.cpp

64 lines

```
namespace mcf {
struct edge {
  int to, capacity, cost, res;
};
int source, destiny;
vector<edge> adj[MAXN];
vector<int> dist;
vector<int> parent;
vector<int> edge_index;
vector<bool> in_queue;
void add_edge(int a, int b, int c, int d) {
  adj[a].pb({b, c, d, (int)adj[b].size()});
  adj[b].pb({a, 0, -d, (int)adj[a].size() - 1});
}
bool dijkstra(int s) {
  dist.assign(MAXN, INF);
  parent.assign(MAXN, -1);
  edge_index.assign(MAXN, -1);
  in_queue.assign(MAXN, false);
  dist[s] = 0;
  queue<int> q;
  q.push(s);
  while (!q.empty()) {
    int u = q.front(), idx = 0;
    q.pop();
    in_queue[u] = false;
    for (auto const &v : adj[u]) {
      if (v.capacity && dist[v.to] > dist[u] + v.cost) {
        dist[v.to] = dist[u] + v.cost;
        parent[v.to] = u;
        edge_index[v.to] = idx;
        if (!in_queue[v.to]) {
          in_queue[v.to] = true;
          q.push(v.to);
        }
      }
      idx++;
    }
  }
  return dist[destiny] != INF;
}
int get_cost() {
  int flow = 0, cost = 0;
  while (dijkstra(source)) {
    int curr_flow = INF, curr = destiny;
    while (curr != source) {
      int p = parent[curr];
      curr_flow = min(curr_flow, adj[p][edge_index[curr]].
           capacity);
      curr = p;
    }
    flow += curr_flow;
    cost += curr_flow * dist[destiny];
    curr = destiny;
    while (curr != source) {
      int p = parent[curr];
      int res_idx = adj[p][edge_index[curr]].res;
      adj[p][edge_index[curr]].capacity -= curr_flow;
      adj[curr][res_idx].capacity += curr_flow;
      curr = p;
    }
  }
  return cost;
}
}  // namespace mcf
```

## scc.cpp

54 lines

```
int n, m;
bool vis[MAXN];
int root[MAXN];
vector<int> order;
vector<int> roots;
vector<int> comp;
vector<vector<int>> comps;
vector<int> adj[MAXN];
vector<int> adj_rev[MAXN];
vector<int> adj_scc[MAXN];
void dfs(int v) {
  vis[v] = true;
  for (auto const &u : adj[v])
    if (!vis[u]) dfs(u);
  order.pb(v);
}
void dfs2(int v) {
  comp.pb(v);
  vis[v] = true;
  for (auto const &u : adj_rev[v])
    if (!vis[u]) dfs2(u);
}
//...
cin >> n >> m;
for (int i = 0; i < m; i++) {
  int a, b;
  cin >> a >> b;
  adj[a].pb(b);
  adj_rev[b].pb(a);
}
```

```
for (int i = 0; i < n; i++) {
  if (!vis[i]) dfs(i);
}
reverse(order.begin(), order.end());
memset(vis, false, sizeof(vis));
for (auto const &v : order) {
  if (!vis[v]) {
    comp.clear();
    dfs2(v);
    comps.pb(comp);
    // making condensation graph
    int r = comp.back();
    for (auto const &u : comp) root[u] = r;
    roots.push_back(r);
  }
}
// making condensation graph
for (int v = 0; v < n; v++) {
  for (auto const &u : adj[v]) {
    int root_v = roots[v];
    int root_u = roots[u];
    if (root_u != root_v) adj_scc[root_v].pb(root_u);
  }
}
```

## articulationpoints.cpp

23 lines

```
int n, m, timer;
vector<int> adj[MAXN];
bool is_cutpoint[MAXN];
int tin[MAXN];  // memset -1
int low[MAXN];  // memset -1
bool vis[MAXN];
void dfs(int v, int p) {  // chama pros nao vis
  vis[v] = true;
  tin[v] = timer, low[v] = timer++;
  int childs = 0;
  for (auto const &u : adj[v]) {
    if (u == p) continue;
    if (vis[u]) {
      low[v] = min(low[v], tin[u]);
    } else {
      dfs(u, v);
      low[v] = min(low[v], low[u]);
      if (low[u] >= tin[v] && p != -1) is_cutpoint[v] = true;
      childs++;
    }
  }
  if (p == -1 && childs > 1) is_cutpoint[v] = true;
}
```

## lca.cpp

39 lines

```
int n;
vector<int> adj[MAXN];
namespace lca {
int l, timer;
vector<int> tin, tout, depth;
vector<vector<int>> up;
void dfs(int v, int p) {
  tin[v] = ++timer;
  up[v][0] = p;
  for (int i = 1; i <= l; i++) up[v][i] = up[up[v][i - 1]][i -
      1];
  for (auto const &u : adj[v]) {
    if (p == u) continue;
    depth[u] = depth[v] + 1;
    dfs(u, v);
  }
  tout[v] = ++timer;
}
bool is_ancestor(int u, int v) { return tin[u] <= tin[v] &&
     tout[u] >= tout[v]; }
int binary_lifting(int u, int v) {
  if (is_ancestor(u, v)) return u;
  if (is_ancestor(v, u)) return v;
  for (int i = l; i >= 0; --i)
    if (!is_ancestor(up[u][i], v)) u = up[u][i];
  return up[u][0];
}
void init() {
  tin.resize(n);
  tout.resize(n);
  depth.resize(n);
  timer = 0;
  l = ceil(log2(n));
  up.assign(n, vector<int>(l + 1));
  dfs(0, 0);
}
int dist(int s, int v) {
  int at = binary_lifting(s, v);
  return (depth[s] + depth[v] - 2 * depth[at]);
}
}  // namespace lca
```

## topsort.cpp

21 lines

```
vector<bool> visited;
vector<int> ans;
void dfs(int v) {
  visited[v] = true;
  for (int u : adj[v]) {
    if (!visited[u]) {
      dfs(u);
    }
  }
  ans.push_back(v);
}
void topological_sort() {
  visited.assign(n, false);
  ans.clear();
  for (int i = 0; i < n; ++i) {
    if (!visited[i]) {
      dfs(i);
    }
  }
  reverse(ans.begin(), ans.end());
}
```

## eulerian.cpp

25 lines

```
void dfs2(int s) {  // caminho euleriano
  while (a[s].size() > 0) {
    auto v = a[s].back();
    a[s].pop_back();
    if (!vis[v.sec]) {
      vis[v.sec] = 1;
      dfs2(v.fir);
    }
  }
  path.pb(s);
}
void dfs(int i) {  // ajeitar para que todo mundo tenha grau
     par
  vis[i] = 1;
  for (auto const &j : adj[i]) {
    if (!vis[j.fir]) {
      dfs(j.fir);
      if (deg[j.fir]) {
        ans.pb(edges[j.sec]);
        deg[j.fir] ^= 1;
        deg[i] ^= 1;
      }
    }
  }
}
// se eu cham dfs(0) no final checar se o deg[0] ta safe
```

## virtualtree.cpp

29 lines

```
// virtual tree
// dado um conjunto de vertices v
// montar uma arvore comprimida
// tal que escolhendo dois vertices do conjunto v[i] e v[j]
// lca(v[i], v[j]) tambem ta na arvore
// se o conjunto v tem k vertices
// entao a arvore comprimida tem menos do que 2k vertices
// O(k log(k)), sem considerar a complexidade de achar lca
int build_virt(vector<int> v) {
  auto cmp = [&](int i, int j) { return lca::tin[i] < lca::tin[
      j]; };
  sort(v.begin(), v.end(), cmp);
  for (int i = v.size() - 1; i > 0; i--) {
    v.pb(lca::find_lca(v[i], v[i - 1]));
  }
  sort(v.begin(), v.end(), cmp);
  v.erase(unique(v.begin(), v.end()), v.end());
  for (int i = 0; i < v.size(); i++) {
    virt[v[i]].clear();
  }
  for (int i = 1; i < v.size(); i++) {
    virt[lca::find_lca(v[i - 1], v[i])].clear();
  }
  for (int i = 1; i < v.size(); i++) {
    int parent = lca::find_lca(v[i - 1], v[i]);
    int d = lca::dist(parent, v[i]);
    virt[parent].pb({v[i], d});
  }
  return v[0];
}
```

## notes.md

53 lines

```
## Bipartite Graph
A bipartite graph is a graph that does not contain any odd-
     length cycles.
## Directed acyclic graph (DAG)
Is a directed graph with no directed cycles.
## Independent Set
Is a set of vertices in a graph, no two of which are adjacent.
     That is, it is a set S of vertices such that for every two
      vertices in S, there is no edge connecting the two.
## Clique
Is a subset of vertices of an undirected graph such that every
     two distinct vertices in the clique are adjacent.
## Vertex Cover
Is a set of vertices that includes at least one endpoint of
     every edge of the graph.
## Edge Cover
Is a set of edges such that every vertex of the graph is
     incident to at least one edge of the set.
## Path Cover
Given a directed graph G = (V, E), a path cover is a set of
     directed paths such that every vertex v belongs to at
     least one path.
## Koning's Theorem
```

```
In any bipartite graph, the number of edges in a maximum
     matching equals the number of vertices in a minimum vertex
      cover.
## Properties
- Every tree is a bipartite graph.
- Any NxM grid is a bipartite graph.
- A set of vertices is a vertex cover if and only if its
     complement is an independent set.
- The number of vertices of a graph is equal to its minimum
     vertex cover number plus the size of a maximum independent
      set.
- In bipartite graphs, the size of the minimum edge cover is
     equal to the size of the maximum independent set
- In bipartite graphs, the size of the minimum edge cover plus
     the size of the minimum vertex cover is equal to the
     number of vertices.
- In bipartite graphs, maximum clique size is two.
## Min-cut
The smallest total weight of the edges which if removed would
     disconnect the source from the sink.
## Max-flow min-cut theorem
In a flow network, the maximum amount of flow passing from the
     source to the sink is equal to the total weight of the
     edges in a minimum cut.
## Maximum flow with vertex capacities
In other words, the amount of flow passing through a vertex
     cannot exceed its capacity. To find the maximum flow, we
     can transform the problem into the maximum flow problem by
      expanding the network. Each vertex v is replaced by v-in
     and v-out, where v-in is connected by edges going into v
     and v-out is connected to edges coming out from v. Then
     assign capacity c(v) to the edge connecting v-in and v-out
     .
## Undirected edge-disjoint paths problem
We are given an undirected graph G = (V, E) and two vertices s
     and t, and we have to find the maximum number of edge-
     disjoint s-t paths in G.
## Undirected vertex-disjoint paths problem
We are given an undirected graph G = (V, E) and two vertices s
     and t, and we have to find the maximum number of vertex-
     disjoint (except for s and t) paths in G.
## Menger's theorem
The maximum number of edge-disjoint s-t paths in an undirected
     graph is equal to the minimum number of edges in an s-t
     cut-set.
## Undirected vertex-disjoint paths solution
We can construct a network N=(V,E) from G with vertex
     capacities, where the capacities of all vertices and all
     edges are 1. Then the value of the maximum flow is equal
     to the maximum number of independent paths from s to t.
## Minimum vertex-disjoint path cover in directed acyclic graph
     (DAG)
Given a directed acyclic graph G=(V, E), we are to find the
     minimum number of vertex-disjoint paths to cover each
     vertex in V. We can construct a bipartite graph G' from G.
      Each vertex v is replaced by v-in and v-out, where v-in
     is connected by edges going into v and v-out is connected
     to edges coming out from v. Then it can be shown that G'
     has a matching M of size m if and only if G has a vertex-
     disjoint path cover C of containing m edges and n-m paths.
## Minimum general path cover in directed acyclic graph (DAG)
A general path cover is a path cover where a vertex can belong
     to more than one path. A minimum general path cover may be
      smaller than a minimum vertex-disjoint path cover. A
     minimum general path cover can be found almost like a
     minimum vertex-disjoint path cover. It suffices to add
     some new edges to the matching graph so that there is an
     edge a to b always when there is a path from a to b in the
      original graph.
## Dilworths theorem and maximum antichain
An antichain is a set of nodes of a graph such that there is no
      path from any node to another node using the edges of the
      graph. Dilworths theorem states that in a directed
     acyclic graph, the size of a minimum general path cover
     equals the size of a maximum antichain.
Or in other words: For a DAG G that if has edges from vertex i
     to vertex j and vertex j to k, then it also has a edge
     from vertex i to vertex k, the size of a minimum path
     cover is equal to the size of a maximum independent set.
## Maximum weighted antichain
In this problem, each vertex has a cost a[i]. The cost of an
     antichain is equal to the sum of the costs of the vertices
      present in it. We need to find the maximum cost of a
     antichain. We can construct the same bipartite of the
     maximum antichain problem from a dag G, these edges have
     an infinite capacity. We also need to create a source
     vertex and a sink, and we need to add edges source to v-in
      with capacity a[v] and v-out to sink with capacity a[v].
     The answer is equal to the sum of all a[i] minus the
     maximum flow on this network.
## Halls Theorem
Halls theorem can be used to find out whether a bipartite graph
      has a matching that contains all left or right nodes.
     Assume that we want to find a matching that contains all
     left nodes. Let X be any set of left nodes and let f(X) be
      the set of their neighbors. According to Halls theorem, a
      matching that contains all left nodes exists exactly when
      for
each X, the condition |X| <= |f(X)| holds.
## Extra (Getting Confidence Trick)
<p>If you need to maximize a number x = (a * b * c * ...), then
      you can write it as x = (e^log(a) * e^log(b) * e^log(c) *
     ...), and then the number is x = e^(log(a) + log(b) + log
     (c) + ...), and the problem now becomes a problem of
     maximizing the sum of (log(a) + log(b) + log(c) + ...).<p
     />
Use exp() and log() C++ functions :)
```

# Geometry (5)

## convexhull.cpp

47 lines

```
struct point {
  int x, y, id;
  point(int x, int y, int id) : x(x), y(y), id(id) {}
  point() {}
  point operator-(point const &o) const { return {x - o.x, y -
       o.y, -1}; }
  bool operator<(point const &o) const {
    if (x == o.x) return y < o.y;
    return x < o.x;
  }
  int operator^(point const &o) const { return x * o.y - y * o.
       x; }
};
int ccw(point const &a, point const &b, point const &x) {
  auto p = (b - a) ^ (x - a);
  return (p > 0) - (p < 0);
}
vector<point> convex_hull(vector<point> P)  // sem colineares
{
  sort(P.begin(), P.end());
  vector<point> L, U;
  for (auto p : P) {
    while (L.size() >= 2 && ccw(L.end()[-2], L.end()[-1], p) ==
         -1) L.pop_back();
    L.push_back(p);
  }
  reverse(P.begin(), P.end());
  for (auto p : P) {
    while (U.size() >= 2 && ccw(U.end()[-2], U.end()[-1], p) ==
         -1) U.pop_back();
    U.push_back(p);
  }
  L.insert(L.end(), U.begin(), U.end() - 1);
  return L;
}
vector<point> convex_hull_no_collinears(vector<point> P)  //
     com colineares
{
  sort(P.begin(), P.end());
  vector<point> L, U;
  for (auto p : P) {
    while (L.size() >= 2 && ccw(L.end()[-2], L.end()[-1], p) <=
         0) L.pop_back();
    L.push_back(p);
  }
  reverse(P.begin(), P.end());
  for (auto p : P) {
    while (U.size() >= 2 && ccw(U.end()[-2], U.end()[-1], p) <=
         0) U.pop_back();
    U.push_back(p);
  }
  L.insert(L.end(), U.begin(), U.end() - 1);
  return L;
}
```

## convexhullpointlocation.cpp

44 lines

```
struct pt {
  int x, y;
  pt operator+(pt p) { return {x + p.x, y + p.y}; }
  pt operator-(pt p) { return {x - p.x, y - p.y}; }
  bool operator==(pt p) { return (x == p.x && y == p.y); }
  int cross(pt p) { return x * p.y - y * p.x; }
  int cross(pt a, pt b) { return (a - *this).cross(b - *this);
       }
  int dot(pt p) { return x * p.x + y * p.y; }
};
bool cmp_x(pt a, pt b) {
  if (a.x != b.x) return a.x < b.x;
  return a.y < b.y;
}
// acha o convex hull
vector<pt> convex_hull(vector<pt> pts) {
  if (pts.size() <= 1) return pts;
  sort(pts.begin(), pts.end(), cmp_x);
  vector<pt> h(pts.size() + 1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(pts.begin(), pts.end
       ())) {
    for (auto const &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])};
}
int sgn(int x) { return (x > 0) - (x < 0); }
int side_of(pt s, pt e, pt p) { return sgn(s.cross(e, p)); }
bool on_segment(pt s, pt e, pt p) { return p.cross(s, e) == 0
     && (s - p).dot(e - p) <= 0; }
// retorna se o ponto p esta dentro ou nao do convex hull l
// caso strict = true, entao considera true se tiver na borda
// caso strict = false, entao considera false se tiver na borda
```

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

## linetrick.cpp

13 lines

```
pi get_line(pi x, pi y) {  // um jeito normalizado de
    representar a reta entre 2 pontos
  int xx = x.fir - y.fir;
  int yy = x.sec - y.sec;
  int g = __gcd(abs(xx), abs(yy));
  if (g != 0) {
    xx /= g, yy /= g;
  }
  if (xx < 0) {
    xx *= -1;
    yy *= -1;
  }
  return {xx, yy};
}
```

## polygonarea.cpp

19 lines

```
double area(vector<pi> fig) {
  double res = 0;
  for (unsigned i = 0; i < fig.size(); i++) {
    pi p = i ? fig[i - 1] : fig.back();
    pi q = fig[i];
    res += (p.fir - q.fir) * (p.sec + q.sec);
  }
  return fabs(res) / 2;
}
int cross(pi a, pi b) { return a.fir * b.sec - a.sec * b.fir; }
double area2(vector<pi> fig) {
  double res = 0;
  for (unsigned i = 0; i < fig.size(); i++) {
    pi p = i ? fig[i - 1] : fig.back();
    pi q = fig[i];
    res += cross(p, q);
  }
  return fabs(res) / 2;
}
```

## geometria.cpp

207 lines

```
typedef double ld;
const ld DINF = 1e18;
const ld pi = acos(-1.0);
const ld eps = 1e-9;
#define sq(x) ((x) * (x))
bool eq(ld a, ld b) { return abs(a - b) <= eps; }
struct pt {  // ponto
  ld x, y;
  pt(ld x_ = 0, ld y_ = 0) : x(x_), y(y_) {}
  bool operator<(const pt p) const {
    if (!eq(x, p.x)) return x < p.x;
    if (!eq(y, p.y)) return y < p.y;
    return 0;
  }
  bool operator==(const pt p) const { return eq(x, p.x) and eq(
      y, p.y); }
  pt operator+(const pt p) const { return pt(x + p.x, y + p.y);
      }
  pt operator-(const pt p) const { return pt(x - p.x, y - p.y);
      }
  pt operator*(const ld c) const { return pt(x * c, y * c); }
  pt operator/(const ld c) const { return pt(x / c, y / c); }
  ld operator*(const pt p) const { return x * p.x + y * p.y; }
  ld operator^(const pt p) const { return x * p.y - y * p.x; }
  friend istream& operator>>(istream& in, pt& p) { return in >>
       p.x >> p.y; }
};
struct line {  // reta
  pt p, q;
  line() {}
  line(pt p_, pt q_) : p(p_), q(q_) {}
  friend istream& operator>>(istream& in, line& r) { return in
      >> r.p >> r.q; }
};
// PONTO & VETOR
ld dist(pt p, pt q) {  // distancia
  return hypot(p.y - q.y, p.x - q.x);
}
ld dist2(pt p, pt q) {  // quadrado da distancia
  return sq(p.x - q.x) + sq(p.y - q.y);
}
ld norm(pt v) {  // norma do vetor
  return dist(pt(0, 0), v);
}
ld angle(pt v) {  // angulo do vetor com o eixo x
  ld ang = atan2(v.y, v.x);
  if (ang < 0) ang += 2 * pi;
  return ang;
}
ld sarea(pt p, pt q, pt r) {  // area com sinal
  return ((q - p) ^ (r - q)) / 2;
}
bool col(pt p, pt q, pt r) {  // se p, q e r sao colin.
  return eq(sarea(p, q, r), 0);
}
bool ccw(pt p, pt q, pt r) {  // se p, q, r sao ccw
  return sarea(p, q, r) > eps;
}
pt rotate(pt p, ld th) {  // rotaciona o ponto th radianos
  return pt(p.x * cos(th) - p.y * sin(th), p.x * sin(th) + p.y
      * cos(th));
}
pt rotate90(pt p) {  // rotaciona 90 graus
  return pt(-p.y, p.x);
}
// RETA
bool isvert(line r) {  // se r eh vertical
  return eq(r.p.x, r.q.x);
}
bool isinseg(pt p, line r) {  // se p pertence ao seg de r
  pt a = r.p - p, b = r.q - p;
  return eq((a ^ b), 0) and (a * b) < eps;
}
ld get_t(pt v, line r) {  // retorna t tal que t*v pertence a
    reta r
  return (r.p ^ r.q) / ((r.p - r.q) ^ v);
}
pt proj(pt p, line r) {  // projecao do ponto p na reta r
  if (r.p == r.q) return r.p;
  r.q = r.q - r.p;
  p = p - r.p;
  pt proj = r.q * ((p * r.q) / (r.q * r.q));
  return proj + r.p;
}
pt inter(line r, line s) {  // r inter s
  if (eq((r.p - r.q) ^ (s.p - s.q), 0)) return pt(DINF, DINF);
  r.q = r.q - r.p, s.p = s.p - r.p, s.q = s.q - r.p;
  return r.q * get_t(r.q, s) + r.p;
}
bool interseg(line r, line s) {  // se o seg de r intersecta o
    seg de s
  if (isinseg(r.p, s) or isinseg(r.q, s) or isinseg(s.p, r) or
      isinseg(s.q, r)) return 1;

  return ccw(r.p, r.q, s.p) != ccw(r.p, r.q, s.q) and ccw(s.p,
      s.q, r.p) != ccw(s.p, s.q, r.q);
}
ld disttoline(pt p, line r) {  // distancia do ponto a reta
  return 2 * abs(sarea(p, r.p, r.q)) / dist(r.p, r.q);
}
ld disttoseg(pt p, line r) {  // distancia do ponto ao seg
  if ((r.q - r.p) * (p - r.p) < 0) return dist(r.p, p);
  if ((r.p - r.q) * (p - r.q) < 0) return dist(r.q, p);
  return disttoline(p, r);
}
ld distseg(line a, line b) {  // distancia entre seg
  if (interseg(a, b)) return 0;
  ld ret = DINF;
  ret = min(ret, disttoseg(a.p, b));
  ret = min(ret, disttoseg(a.q, b));
  ret = min(ret, disttoseg(b.p, a));
  ret = min(ret, disttoseg(b.q, a));

  return ret;
}
// POLIGONO
// corta poligono com a reta r deixando os pontos p tal que
// ccw(r.p, r.q, p)
vector<pt> cut_polygon(vector<pt> v, line r) {  // O(n)
  vector<pt> ret;
  for (int j = 0; j < v.size(); j++) {
    if (ccw(r.p, r.q, v[j])) ret.push_back(v[j]);
    if (v.size() == 1) continue;
    line s(v[j], v[(j + 1) % v.size()]);
    pt p = inter(r, s);
    if (isinseg(p, s)) ret.push_back(p);
  }
  ret.erase(unique(ret.begin(), ret.end()), ret.end());
  if (ret.size() > 1 and ret.back() == ret[0]) ret.pop_back();
  return ret;
}
// distancia entre os retangulos a e b (lados paralelos aos
    eixos)
// assume que ta representado (inferior esquerdo, superior
    direito)
ld dist_rect(pair<pt, pt> a, pair<pt, pt> b) {
  ld hor = 0, vert = 0;
  if (a.second.x < b.first.x)
    hor = b.first.x - a.second.x;
  else if (b.second.x < a.first.x)
    hor = a.first.x - b.second.x;
  if (a.second.y < b.first.y)
    vert = b.first.y - a.second.y;
  else if (b.second.y < a.first.y)
    vert = a.first.y - b.second.y;
  return dist(pt(0, 0), pt(hor, vert));
}
ld polarea(vector<pt> v) {  // area do poligono
  ld ret = 0;
  for (int i = 0; i < v.size(); i++) ret += sarea(pt(0, 0), v[i
      ], v[(i + 1) % v.size()]);
  return abs(ret);
```

```
}
// se o ponto ta dentro do poligono: retorna 0 se ta fora,
// 1 se ta no interior e 2 se ta na borda
int inpol(vector<pt>& v, pt p) {  // O(n)
  int qt = 0;
  for (int i = 0; i < v.size(); i++) {
    if (p == v[i]) return 2;
    int j = (i + 1) % v.size();
    if (eq(p.y, v[i].y) and eq(p.y, v[j].y)) {
      if ((v[i] - p) * (v[j] - p) < eps) return 2;
      continue;
    }
    bool baixo = v[i].y + eps < p.y;
    if (baixo == (v[j].y + eps < p.y)) continue;
    auto t = (p - v[i]) ^ (v[j] - v[i]);
    if (eq(t, 0)) return 2;
    if (baixo == (t > eps)) qt += baixo ? 1 : -1;
  }
  return qt != 0;
}
bool interpol(vector<pt> v1, vector<pt> v2) {  // se dois
    poligonos se intersectam - O(n*m)
  int n = v1.size(), m = v2.size();
  for (int i = 0; i < n; i++)
    if (inpol(v2, v1[i])) return 1;
  for (int i = 0; i < n; i++)
    if (inpol(v1, v2[i])) return 1;
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      if (interseg(line(v1[i], v1[(i + 1) % n]), line(v2[j], v2
          [(j + 1) % m]))) return 1;
  return 0;
}
ld distpol(vector<pt> v1, vector<pt> v2) {  // distancia entre
    poligonos
  if (interpol(v1, v2)) return 0;
  ld ret = DINF;
  for (int i = 0; i < v1.size(); i++)
    for (int j = 0; j < v2.size(); j++)
      ret = min(ret, distseg(line(v1[i], v1[(i + 1) % v1.size()
          ]), line(v2[j], v2[(j + 1) % v2.size()])));
  return ret;
}
// CIRCUNFERENCIA
pt getcenter(pt a, pt b, pt c) {  // centro da circunf dado 3
    pontos
  b = (a + b) / 2;
  c = (a + c) / 2;
  return inter(line(b, b + rotate90(a - b)), line(c, c +
      rotate90(a - c)));
}
vector<pt> circ_line_inter(pt a, pt b, pt c, ld r) {  //
    intersecao da circunf (c, r) e reta ab
  vector<pt> ret;
  b = b - a, a = a - c;
  ld A = b * b;
  ld B = a * b;
  ld C = a * a - r * r;
  ld D = B * B - A * C;
  if (D < -eps) return ret;
  ret.push_back(c + a + b * (-B + sqrt(D + eps)) / A);
  if (D > eps) ret.push_back(c + a + b * (-B - sqrt(D)) / A);
  return ret;
}
vector<pt> circ_inter(pt a, pt b, ld r, ld R) {  // intersecao
    da circunf (a, r) e (b, R)
  vector<pt> ret;
  ld d = dist(a, b);
  if (d > r + R or d + min(r, R) < max(r, R)) return ret;
  ld x = (d * d - R * R + r * r) / (2 * d);
  ld y = sqrt(r * r - x * x);
  pt v = (b - a) / d;
  ret.push_back(a + v * x + rotate90(v) * y);
  if (y > 0) ret.push_back(a + v * x - rotate90(v) * y);
  return ret;
}
```

## geometriaint.cpp

105 lines

```
#define sq(x) ((x) * (ll)(x))
struct pt {  // ponto
  int x, y;
  pt(int x_ = 0, int y_ = 0) : x(x_), y(y_) {}
  bool operator<(const pt p) const {
    if (x != p.x) return x < p.x;
    return y < p.y;
  }
  bool operator==(const pt p) const { return x == p.x and y ==
      p.y; }
  pt operator+(const pt p) const { return pt(x + p.x, y + p.y);
      }
  pt operator-(const pt p) const { return pt(x - p.x, y - p.y);
      }
  pt operator*(const int c) const { return pt(x * c, y * c); }
  ll operator*(const pt p) const { return x * (ll)p.x + y * (ll
      )p.y; }
  ll operator^(const pt p) const { return x * (ll)p.y - y * (ll
      )p.x; }
  friend istream& operator>>(istream& in, pt& p) { return in >>
      p.x >> p.y; }
};
struct line {  // reta
  pt p, q;
  line() {}
  line(pt p_, pt q_) : p(p_), q(q_) {}
  friend istream& operator>>(istream& in, line& r) { return in
      >> r.p >> r.q; }
};
// PONTO & VETOR
ll dist2(pt p, pt q) {  // quadrado da distancia
  return sq(p.x - q.x) + sq(p.y - q.y);
}
ll sarea2(pt p, pt q, pt r) {  // 2 * area com sinal
  return (q - p) ^ (r - q);
}
bool col(pt p, pt q, pt r) {  // se p, q e r sao colin.
  return sarea2(p, q, r) == 0;
}
bool ccw(pt p, pt q, pt r) {  // se p, q, r sao ccw
  return sarea2(p, q, r) > 0;
}
int quad(pt p) {  // quadrante de um ponto
  return (p.x < 0) ^ 3 * (p.y < 0);
}
bool compare_angle(pt p, pt q) {  // retorna se ang(p) < ang(q)
  if (quad(p) != quad(q)) return quad(p) < quad(q);
  return ccw(q, pt(0, 0), p);
}
pt rotate90(pt p) {  // rotaciona 90 graus
  return pt(-p.y, p.x);
}
// RETA
bool isinseg(pt p, line r) {  // se p pertence ao seg de r
  pt a = r.p - p, b = r.q - p;
  return (a ^ b) == 0 and (a * b) <= 0;
}
bool interseg(line r, line s) {  // se o seg de r intersecta o
    seg de s
  if (isinseg(r.p, s) or isinseg(r.q, s) or isinseg(s.p, r) or
      isinseg(s.q, r)) return 1;

  return ccw(r.p, r.q, s.p) != ccw(r.p, r.q, s.q) and ccw(s.p,
      s.q, r.p) != ccw(s.p, s.q, r.q);
}
int segpoints(line r) {  // numero de pontos inteiros no
    segmento
  return 1 + __gcd(abs(r.p.x - r.q.x), abs(r.p.y - r.q.y));
}
double get_t(pt v, line r) {  // retorna t tal que t*v pertence
     a reta r
  return (r.p ^ r.q) / (double)((r.p - r.q) ^ v);
}
// POLIGONO
// quadrado da distancia entre os retangulos a e b (lados
    paralelos aos eixos)
// assume que ta representado (inferior esquerdo, superior
    direito)
ll dist2_rect(pair<pt, pt> a, pair<pt, pt> b) {
  int hor = 0, vert = 0;
  if (a.second.x < b.first.x)
    hor = b.first.x - a.second.x;
  else if (b.second.x < a.first.x)
    hor = a.first.x - b.second.x;
  if (a.second.y < b.first.y)
    vert = b.first.y - a.second.y;
  else if (b.second.y < a.first.y)
    vert = a.first.y - b.second.y;
  return sq(hor) + sq(vert);
}
ll polarea2(vector<pt> v) {  // 2 * area do poligono
  ll ret = 0;
  for (int i = 0; i < v.size(); i++) ret += sarea2(pt(0, 0), v[
      i], v[(i + 1) % v.size()]);
  return abs(ret);
}
// se o ponto ta dentro do poligono: retorna 0 se ta fora,
// 1 se ta no interior e 2 se ta na borda
int inpol(vector<pt>& v, pt p) {  // O(n)
  int qt = 0;
  for (int i = 0; i < v.size(); i++) {
    if (p == v[i]) return 2;
    int j = (i + 1) % v.size();
    if (p.y == v[i].y and p.y == v[j].y) {
      if ((v[i] - p) * (v[j] - p) <= 0) return 2;
      continue;
    }
    bool baixo = v[i].y < p.y;
    if (baixo == (v[j].y < p.y)) continue;
    auto t = (p - v[i]) ^ (v[j] - v[i]);
    if (!t) return 2;
    if (baixo == (t > 0)) qt += baixo ? 1 : -1;
  }
  return qt != 0;
}
ll interior_points(vector<pt> v) {  // pontos inteiros dentro
    de um poligono simples
  ll b = 0;
  for (int i = 0; i < v.size(); i++) b += segpoints(line(v[i],
      v[(i + 1) % v.size()])) - 1;
  return (polarea2(v) - b) / 2 + 1;
}
```

## geometria3d.cpp

132 lines

```
typedef double ld;
const ld DINF = 1e18;
const ld eps = 1e-9;
```

```
#define sq(x) ((x) * (x))
bool eq(ld a, ld b) { return abs(a - b) <= eps; }
struct pt {  // ponto
  ld x, y, z;
  pt(ld x_ = 0, ld y_ = 0, ld z_ = 0) : x(x_), y(y_), z(z_) {}
  bool operator<(const pt p) const {
    if (!eq(x, p.x)) return x < p.x;
    if (!eq(y, p.y)) return y < p.y;
    if (!eq(z, p.z)) return z < p.z;
    return 0;
  }
  bool operator==(const pt p) const { return eq(x, p.x) and eq(
      y, p.y) and eq(z, p.z); }
  pt operator+(const pt p) const { return pt(x + p.x, y + p.y,
      z + p.z); }
  pt operator-(const pt p) const { return pt(x - p.x, y - p.y,
      z - p.z); }
  pt operator*(const ld c) const { return pt(x * c, y * c, z *
      c); }
  pt operator/(const ld c) const { return pt(x / c, y / c, z /
      c); }
  ld operator*(const pt p) const { return x * p.x + y * p.y + z
       * p.z; }
  pt operator^(const pt p) const { return pt(y * p.z - z * p.y,
      z * p.x - x * p.z, x * p.y - y * p.x); }
  friend istream& operator>>(istream& in, pt& p) { return in >>
       p.x >> p.y >> p.z; }
};
struct line {  // reta
  pt p, q;
  line() {}
  line(pt p_, pt q_) : p(p_), q(q_) {}
  friend istream& operator>>(istream& in, line& r) { return in
      >> r.p >> r.q; }
};
struct plane {      // plano
  array<pt, 3> p;   // pontos que definem o plano
  array<ld, 4> eq;  // equacao do plano
  plane() {}
  plane(pt p_, pt q_, pt r_) : p({p_, q_, r_}) { build(); }

  friend istream& operator>>(istream& in, plane& P) {
    return in >> P.p[0] >> P.p[1] >> P.p[2];
    P.build();
  }
  void build() {
    pt dir = (p[1] - p[0]) ^ (p[2] - p[0]);
    eq = {dir.x, dir.y, dir.z, dir * p[0] * (-1)};
  }
};
// converte de coordenadas polares para cartesianas
// (angulos devem estar em radianos)
// phi eh o angulo com o eixo z (cima) theta eh o angulo de
    rotacao ao redor de z
pt convert(ld rho, ld th, ld phi) { return pt(sin(phi) * cos(th
    ), sin(phi) * sin(th), cos(phi)) * rho; }
// projecao do ponto p na reta r
pt proj(pt p, line r) {
  if (r.p == r.q) return r.p;
  r.q = r.q - r.p;
  p = p - r.p;
  pt proj = r.q * ((p * r.q) / (r.q * r.q));
  return proj + r.p;
}
// projecao do ponto p no plano P
pt proj(pt p, plane P) {
  p = p - P.p[0], P.p[1] = P.p[1] - P.p[0], P.p[2] = P.p[2] - P
      .p[0];
  pt norm = P.p[1] ^ P.p[2];
  pt proj = p - (norm * (norm * p) / (norm * norm));
  return proj + P.p[0];
}
// distancia
ld dist(pt a, pt b) { return sqrt(sq(a.x - b.x) + sq(a.y - b.y)
     + sq(a.z - b.z)); }
// distancia ponto reta
ld distline(pt p, line r) { return dist(p, proj(p, r)); }
// distancia de ponto para segmento
ld distseg(pt p, line r) {
  if ((r.q - r.p) * (p - r.p) < 0) return dist(r.p, p);
  if ((r.p - r.q) * (p - r.q) < 0) return dist(r.q, p);
  return distline(p, r);
}
// distancia de ponto a plano com sinal
ld sdist(pt p, plane P) { return P.eq[0] * p.x + P.eq[1] * p.y
    + P.eq[2] * p.z + P.eq[3]; }
// distancia de ponto a plano
ld distplane(pt p, plane P) { return abs(sdist(p, P)); }
// se ponto pertence a reta
bool isinseg(pt p, line r) { return eq(distseg(p, r), 0); }
// se ponto pertence ao triangulo definido por P.p
bool isinpol(pt p, vector<pt> v) {
  assert(v.size() >= 3);
  pt norm = (v[1] - v[0]) ^ (v[2] - v[1]);
  bool inside = true;
  int sign = -1;
  for (int i = 0; i < v.size(); i++) {
    line r(v[(i + 1) % 3], v[i]);
    if (isinseg(p, r)) return true;
    pt ar = v[(i + 1) % 3] - v[i];
    if (sign == -1)
      sign = ((ar ^ (p - v[i])) * norm > 0);
    else if (((ar ^ (p - v[i])) * norm > 0) != sign)
      inside = false;
  }
  return inside;
}
// distancia de ponto ate poligono
ld distpol(pt p, vector<pt> v) {
  pt p2 = proj(p, plane(v[0], v[1], v[2]));
  if (isinpol(p2, v)) return dist(p, p2);
  ld ret = DINF;
  for (int i = 0; i < v.size(); i++) {
    int j = (i + 1) % v.size();
    ret = min(ret, distseg(p, line(v[i], v[j])));
  }
  return ret;
}
// intersecao de plano e segmento
// BOTH = o segmento esta no plano
// ONE = um dos pontos do segmento esta no plano
// PARAL = segmento paralelo ao plano
// CONCOR = segmento concorrente ao plano
enum RETCODE { BOTH, ONE, PARAL, CONCOR };
pair<RETCODE, pt> intersect(plane P, line r) {
  ld d1 = sdist(r.p, P);
  ld d2 = sdist(r.q, P);
  if (eq(d1, 0) and eq(d2, 0)) return pair(BOTH, r.p);
  if (eq(d1, 0)) return pair(ONE, r.p);
  if (eq(d2, 0)) return pair(ONE, r.q);
  if ((d1 > 0 and d2 > 0) or (d1 < 0 and d2 < 0)) {
    if (eq(d1 - d2, 0)) return pair(PARAL, pt());
    return pair(CONCOR, pt());
  }
  ld frac = d1 / (d1 - d2);
  pt res = r.p + ((r.q - r.p) * frac);
  return pair(ONE, res);
}
// rotaciona p ao redor do eixo u por um angulo a
pt rotate(pt p, pt u, ld a) {
  u = u / dist(u, pt());
  return u * (u * p) + (u ^ p ^ u) * cos(a) + (u ^ p) * sin(a);
}
```

## kdtree.cpp

52 lines

```
struct pt {
  int x, y, id;
  pt() {}
  pt(int xx, int yy) { x = xx, y = yy; }
  pt operator-(pt p) const { return pt(x - p.x, y - p.y); }
  bool operator<(pt p) const { return x < p.x; }
  int dist() const { return x * x + y * y; }
};
bool on_x(const pt &a, const pt &b) { return a.x < b.x; }
bool on_y(const pt &a, const pt &b) { return a.y < b.y; }
struct node {
  pt pp;
  int id;
  int x0 = inf, x1 = -inf, y0 = inf, y1 = -inf;
  node *first = 0, *second = 0;
  int distance(const pt &p) {
    int x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    int y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (pt(x, y) - p).dist();
  }
  node(vector<pt> &&vp) : pp(vp[0]) {
    for (pt 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) {
      sort(vp.begin(), vp.end(), x1 - x0 >= y1 - y0 ? on_x :
          on_y);
      int half = vp.size() / 2;
      first = new node({vp.begin(), vp.begin() + half});
      second = new node({vp.begin() + half, vp.end()});
    }
  }
};
struct kd_tree {
  node *root;
  kd_tree(const vector<pt> &vp) : root(new node({vp.begin(), vp
      .end()})) {}
  pi search(node *n, const pt &p) {
    if (!n->first) {
      if (n->pp.x == p.x && n->pp.y == p.y) return make_pair(
          inf, n->pp.id);  // distancia infinita pra pontos
          iguais
      return make_pair((p - n->pp).dist(), n->pp.id);
    }
    node *f = n->first, *s = n->second;
    int bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);
    auto best = search(f, p);
    if (bsec < best.first || (!f->first)) best = min(best,
        search(s, p));
    return best;
  }
  pi nearest(const pt &p) { return search(root, p); }
};
```

halfplaneintersection.cpp 75 lines

```
const long double eps = 1e-9;
const long double inf = 1e9;
struct pt {
  long double x, y;
  pt(long double x = 0, long double y = 0) : x(x), y(y) {}
  friend pt operator+(pt p, pt q) { return pt(p.x + q.x, p.y +
       q.y); }
  friend pt operator-(pt p, pt q) { return pt(p.x - q.x, p.y -
       q.y); }
  friend pt operator*(pt p, long double k) { return pt(p.x * k,
        p.y * k); }
  friend long double dot(pt p, pt q) { return p.x * q.x + p.y *
        q.y; }
  friend long double cross(pt p, pt q) { return p.x * q.y - p.y
       * q.x; }
};
struct halfplane {
  pt p, pq;
  long double angle;
  halfplane() {}
  halfplane(pt a, pt b) : p(a), pq(b - a) { angle = atan2l(pq.y
       , pq.x); }
  bool out(const pt &r) { return cross(pq, r - p) < -eps; }
  bool operator<(halfplane e) const { return angle < e.angle; }
  friend pt inter(halfplane s, halfplane t) {
    long double alpha = cross((t.p - s.p), t.pq) / cross(s.pq,
         t.pq);
    return s.p + (s.pq * alpha);
  }
};
vector<pt> hp_intersect(vector<halfplane> &h) {
  pt box[4] = {pt(inf, inf), pt(-inf, inf), pt(-inf, -inf), pt(
       inf, -inf)};  // Bounding box in CCW order
  for (int i = 0; i < 4; i++) {
    halfplane aux(box[i], box[(i + 1) % 4]);
    h.pb(aux);
  }
  sort(h.begin(), h.end());
  deque<halfplane> dq;
  int len = 0;
  for (int i = 0; i < h.size(); i++) {
    while (len > 1 && h[i].out(inter(dq[len - 1], dq[len - 2]))
         ) {
      dq.pop_back();
      --len;
    }
    while (len > 1 && h[i].out(inter(dq[0], dq[1]))) {
      dq.pop_front();
      --len;
    }
    if (len > 0 && fabsl(cross(h[i].pq, dq[len - 1].pq)) < eps)
         {
      if (dot(h[i].pq, dq[len - 1].pq) < 0.0) {
        return vector<pt>();
      }
      if (h[i].out(dq[len - 1].p)) {
        dq.pop_back();
        --len;
      } else {
        continue;
      }
    }
    dq.push_back(h[i]);
    ++len;
  }
  while (len > 2 && dq[0].out(inter(dq[len - 1], dq[len - 2])))
       {
    dq.pop_back();
    --len;
  }
  while (len > 2 && dq[len - 1].out(inter(dq[0], dq[1]))) {
    dq.pop_front();
    --len;
  }
  if (len < 3) {
    return vector<pt>();
  }
  vector<pt> ret(len);
  for (int i = 0; i + 1 < len; i++) {
    ret[i] = inter(dq[i], dq[i + 1]);
  }
  ret.back() = inter(dq[len - 1], dq[0]);
  return ret;
}
// half-plane - regiao planar que consiste de todos os pontos
     que estao de um lado de uma reta
// e dai pros half-planes, considerando que eh a regiao da
     esquerda em relacao ao vetor de direcao
```

# Strings (6)

ahocorasick.cpp 78 lines

```
namespace aho {
int go(int v, char ch);
const int K = 26;  // tamanho do alfabeto
struct trie {
  char me;             // char correspondente ao no atual
  int go[K];           // proximo vertice que eu devo ir
       estando em um estado (v, c)
  int down[K];         // proximo vertice da trie
  int is_leaf = 0;     // se o vertice atual da trie eh uma
       folha (fim de uma ou mais strings)
  int parent = -1;     // no ancestral do no atual
  int link = -1;       // link de sufixo do no atual (outro no
       com o maior matching de sufixo)
  int exit_link = -1;  // folha mais proxima que pode ser
       alcancada a partir de v usando links de sufixo
  trie(int p = -1, char ch = '$') : parent(p), me(ch) {
    fill(begin(go), end(go), -1);
    fill(begin(down), end(down), -1);
  }
};
vector<trie> ac;
void init()  // criar a raiz da trie
{
  ac.resize(1);
}
void add_string(string s)  // adicionar string na trie
{
  int v = 0;
  for (auto const &ch : s) {
    int c = ch - 'a';
    if (ac[v].down[c] == -1) {
      ac[v].down[c] = ac.size();
      ac.emplace_back(v, ch);
    }
    v = ac[v].down[c];
  }
  ac[v].is_leaf++;
}
int get_link(int v)  // pegar o suffix link saindo de v
{
  if (ac[v].link == -1) ac[v].link = (!v || !ac[v].parent) ? 0
       : go(get_link(ac[v].parent), ac[v].me);
  return ac[v].link;
}
int go(int v, char ch)  // proximo estado saindo do estado(v,
     ch)
{
  int c = ch - 'a';
  if (ac[v].go[c] == -1) {
    if (ac[v].down[c] != -1)
      ac[v].go[c] = ac[v].down[c];
    else
      ac[v].go[c] = (!v) ? 0 : go(get_link(v), ch);
  }
  return ac[v].go[c];
}
int get_exit_link(int v)  // suffix link mais proximo de v que
     seja uma folha
{
  if (ac[v].exit_link == -1) {
    int curr = get_link(v);
    if (!v || !curr)
      ac[v].exit_link = 0;
    else if (ac[curr].is_leaf)
      ac[v].exit_link = curr;
    else
      ac[v].exit_link = get_exit_link(curr);
  }
  return ac[v].exit_link;
}
int query(string s)  // query O(n + ans)
{
  int ans = 0, curr = 0, at;
  for (auto const &i : s) {
    curr = go(curr, i);
    ans += ac[curr].is_leaf;
    at = get_exit_link(curr);
    while (at) {
      ans += ac[at].is_leaf;
      at = get_exit_link(at);
    }
  }
  return ans;
}
}  // namespace aho
```

kmp.cpp 27 lines

```
string s;
int n, m;
string a, b;
int c[MAXN][26];
vector<int> kmp(string &s) {
  int n = s.size();
  vector<int> p(n);
  for (int i = 1; i < n; i++) {
    int j = p[i - 1];
    while (j > 0 && s[i] != s[j]) j = p[j - 1];
    if (s[i] == s[j]) j++;
    p[i] = j;
  }
  return p;
}
void compute(string s) {
  s.pb('*');
  vector<int> p = kmp(s);
  for (int i = 0; i < s.size(); i++) {
    for (int cc = 0; cc < 26; cc++) {
      int j = i;
      while (j > 0 && 'a' + cc != s[j]) j = p[j - 1];
      if ('a' + cc == s[j]) j++;
```

```
      c[i][cc] = j;
    }
  }
}
```

## manacher.cpp

22 lines

```
vector<int> d1;
vector<int> d2;
void manacher(string s) {
  d1.resize(s.size());
  d2.resize(s.size());
  int l = 0, r = -1;
  for (int i = 0; i < s.size(); i++) {
    int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
    while (0 <= i - k && i + k < s.size() && s[i - k] == s[i +
        k]) k++;
    d1[i] = k;
    k = k - 1;
    if (i + k > r) l = i - k, r = i + k;
  }
  l = 0, r = -1;
  for (int i = 0; i < s.size(); i++) {
    int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
    while (0 <= i - k - 1 && i + k < s.size() && s[i - k - 1]
        == s[i + k]) k++;
    d2[i] = k;
    k = k - 1;
    if (i + k > r) l = i - k - 1, r = i + k;
  }
}
```

## minsuffix.cpp

28 lines

```
int max_suffix(string s, bool mi = false) {
  s.push_back(*min_element(s.begin(), s.end()) - 1);
  int ans = 0;
  for (int i = 1; i < s.size(); i++) {
    int j = 0;
    while (ans + j < i and s[i + j] == s[ans + j]) j++;
    if (s[i + j] > s[ans + j]) {
      if (!mi or i != s.size() - 2) ans = i;
    } else if (j)
      i += j - 1;
  }
  return ans;
}
int min_suffix(string s) {
  for (auto &i : s) i *= -1;
  s.push_back(*max_element(s.begin(), s.end()) + 1);
  return max_suffix(s, true);
}
int max_cyclic_shift(string s) {
  int n = s.size();
  for (int i = 0; i < n; i++) s.pb(s[i]);
  return max_suffix(s);
}
int min_cyclic_shift(string s) {
  for (auto &i : s) i *= -1;
  return max_cyclic_shift(s);
}
// retorna a posicao de inicio menor/maior sufixo/shift de uma
    string
```

## hashing.cpp

29 lines

```
const int MOD = (1ll << 61) - 1;
int P;
int mulmod(int a, int b) {
  const static int LOWER = (1ll << 30) - 1, GET31 = (1ll << 31)
      - 1;
  int l1 = a & LOWER, h1 = a >> 30, l2 = b & LOWER, h2 = b >>
      30;
  int m = l1 * h2 + l2 * h1, h = h1 * h2;
  int ans = l1 * l2 + (h >> 1) + ((h & 1) << 60) + (m >> 31) +
      ((m & GET31) << 30) + 1;
  ans = (ans & MOD) + (ans >> 61), ans = (ans & MOD) + (ans >>
      61);
  return ans - 1;
}
mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().
    count());
int uniform(int l, int r) {
  uniform_int_distribution<int> uid(l, r);
  return uid(rng);
}
struct string_hashing {
  vector<int> h, p;
  string_hashing() {}
  string_hashing(string s) : h(s.size()), p(s.size()) {
    p[0] = 1, h[0] = s[0];
    for (int i = 1; i < s.size(); i++) p[i] = mulmod(p[i - 1],
        P), h[i] = (mulmod(h[i - 1], P) + s[i]) % MOD;
  }
  int get(int l, int r) {
    int hash = h[r] - (l ? mulmod(h[l - 1], p[r - l + 1]) : 0);
    return hash < 0 ? hash + MOD : hash;
  }
  int append(int h, int hb, int blen) { return (hb + mulmod(h,
      p[blen])) % MOD; }
};
// lembrar do P = uniform(256, MOD - 1);
```

## suffixarray.cpp

44 lines

```
vector<int> suffix_array(string s) {
  s += "$";  // menor do que todos os chars da string st
  int n = s.size(), N = max(n + 1, 260ll);
  vector<int> sa(n), ra(n);
  for (int i = 0; i < n; i++) {
    sa[i] = i, ra[i] = s[i];
  }
  for (int k = 0; k < n; k ? k *= 2 : k++) {
    vector<int> nsa(sa), nra(n), cnt(N);
    for (int i = 0; i < n; i++) {
      nsa[i] = (nsa[i] - k + n) % n;
      cnt[ra[i]]++;
    }
    for (int i = 1; i < N; i++) {
      cnt[i] += cnt[i - 1];
    }
    for (int i = n - 1; i + 1; i--) {
      sa[--cnt[ra[nsa[i]]]] = nsa[i];
    }
    for (int i = 1, r = 0; i < n; i++) {
      nra[sa[i]] = r += (ra[sa[i]] != ra[sa[i - 1]] || ra[(sa[i
          ] + k) % n] != ra[(sa[i - 1] + k) % n]);
    }
    ra = nra;
    if (ra[sa[n - 1]] == n - 1) break;
  }
  return vector<int>(sa.begin() + 1, sa.end());
}
vector<int> kasai(string s, vector<int> sa) {
  int n = s.size(), k = 0;
  vector<int> ra(n), lcp(n);
  for (int i = 0; i < n; i++) {
    ra[sa[i]] = i;
  }
  for (int i = 0; i < n; i++, k -= !!k) {
    if (ra[i] == n - 1) {
      k = 0;
      continue;
    }
    int j = sa[ra[i] + 1];
    while (i + k < n and j + k < n and s[i + k] == s[j + k]) k
        ++;
    lcp[ra[i]] = k;
  }
  return lcp;
}
```

## zfunction.cpp

12 lines

```
vector<int> z_function(string &s) {
  int n = s.size();
  vector<int> z(n);
  z[0] = 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;
}
// z[i] = o tamanho de lcp(s, s.substr(i, n - i))
```

## suffixautomaton.cpp

90 lines

```
namespace sa {
struct state {
  int len, suf_link;
  map<char, int> nxt;
};
vector<int> term;
state st[2 * MAXN];
int dp[2 * MAXN];
int sz, last;
void init() {
  memset(dp, -1, sizeof(dp));
  st[0].len = 0;
  st[0].suf_link = -1;
  sz++;
  last = 0;
}
void get_link(int curr, int p, char c) {
  while (p != -1 && !st[p].nxt.count(c)) {
    st[p].nxt[c] = curr;
    p = st[p].suf_link;
  }
  if (p == -1) {
    st[curr].suf_link = 0;
    return;
  }
  int q = st[p].nxt[c];
  if (st[p].len + 1 == st[q].len) {
    st[curr].suf_link = q;
    return;
  }
  int clone = sz;
  sz++;
  st[clone].len = st[p].len + 1;
  st[clone].nxt = st[q].nxt;
  st[clone].suf_link = st[q].suf_link;
  while (p != -1 && st[p].nxt[c] == q) {
    st[p].nxt[c] = clone;
    p = st[p].suf_link;
```

```
  }
  st[q].suf_link = clone;
  st[curr].suf_link = clone;
}
void build(string &s) {
  for (auto const &c : s) {
    int curr = sz;
    sz++;
    st[curr].len = st[last].len + 1;
    get_link(curr, last, c);
    last = curr;
  }
  // achar os estados terminais
  // um estado terminal eh aquele que representa um sufixo da
       string s
  int p = last;
  while (p != -1) {
    term.pb(p);
    p = st[p].suf_link;
  }
}
void dfs2(int v) {
  if (dp[v] != -1) return;
  dp[v] = 1;
  for (auto const &u : st[v].nxt) {
    if (!u.sec) continue;
    dfs2(u.sec);
    dp[v] += dp[u.sec];
  }
}
void dfs(int v, int k, int &at, string &curr) {
  if (at == k) return;
  for (auto const &u : st[v].nxt) {
    if (!u.sec) continue;
    if (at + dp[u.sec] < k) {
      at += dp[u.sec];
      continue;
    }
    curr.pb(u.fir);
    at++;
    dfs(u.sec, k, at, curr);
    if (at == k) return;
    curr.pop_back();
  }
}
void find_kth(int k) {
  int at = 0;
  string curr = "";
  dfs(0, k, at, curr);
  cout << curr << endl;
}
}  // namespace sa
// chamar sa::init(); e sa::build(s);
```

### eertree.cpp

44 lines

```
struct eertree {
  vector<vector<int>> t;
  int n, last, sz;
  vector<int> s, len, link, qt;
  eertree(int N) {
    t = vector(N + 2, vector<int>(26, 0));
    s = len = link = qt = vector<int>(N + 2);
    s[0] = -1;
    link[0] = 1, len[0] = 0, link[1] = 1, len[1] = -1;
    sz = 2, last = 0, n = 1;
  }
  int add(char c) {
    int ret = -1;
    s[n++] = c -= 'a';
    while (s[n - len[last] - 2] != c) last = link[last];
    if (!t[last][c]) {
      int prev = link[last];
      while (s[n - len[prev] - 2] != c) prev = link[prev];
      link[sz] = t[prev][c];
      len[sz] = len[last] + 2;
      ret = len[sz];
      t[last][c] = sz++;
    }
    qt[last = t[last][c]]++;
    return ret;
  }
  int size() { return sz - 2; }
  int propagate() {
    int ret = 0;
    for (int i = n; i > 1; i--) {
      qt[link[i]] += qt[i];
      ret += qt[i];
    }
    return ret;
  }
  vector<pi> get_palindromes(string &s) {
    vector<pi> ans;
    for (int i = 0; i < s.size(); i++) {
      int len = add(s[i]);
      if (len != -1) ans.pb({i - len + 1, i});
    }
    return ans;
  }
};
```

# Various (7)

### binarysearch.cpp

26 lines

```
// busca ternaria para int, usando busca binaria:
int l = 0, r = 1e9;
while (l < r) {
  int mid = (l + r) >> 1;
  (calc(mid) < calc(mid + 1)) ? r = mid : l = mid + 1;
}
return calc(l);
// first element >= x
int lower(int l, int r, int x)  // first element >= x
{
  while (l < r) {
    //  int mid = l + (r - l) / 2; se tiver numero negativo
    int mid = (l + r) >> 1;
    (x <= k[mid]) ? r = mid : l = mid + 1;
  }
  return k[l];
}
// last element <= x
vector<int> k(MAXN);
int upper(int l, int r, int x) {
  while (l < r) {
    int mid = (l + r + 1) >> 1;
    (k[mid] <= x) ? l = mid : r = mid - 1;
  }
  return k[l];
}
```

### ternarysearch.cpp

12 lines

```
double eps = 1e-9;
while (r - l > eps) {
  double m1 = l + (r - l) / 3;
  double m2 = r - (r - l) / 3;
  double f1 = f(m1);
  double f2 = f(m2);
  if (f1 < f2)
    l = m1;
  else
    r = m2;
}
return f(l);  // o maximo de f(l)
```

### parallelbinarysearch.cpp

37 lines

```
vector<pii> qry(q);
for (int i = 0; i < q; i++) {
  cin >> qry[i].sec.fir >> qry[i].sec.sec >> qry[i].fir;
  qry[i].sec.fir--, qry[i].sec.sec--;
}
vector<int> l(n);
vector<int> r(n);
vector<vector<int>> on(q);
for (int i = 0; i < n; i++) {
  l[i] = 0;
  r[i] = q;
}
while (1) {
  bool ok = 1;
  for (int i = 0; i < n; i++) {
    if (l[i] < r[i]) {
      ok = 0;
      int mid = (l[i] + r[i]) >> 1;
      on[mid].pb(i);
    }
  }
  if (ok) break;
  reset();
  for (int mid = 0; mid < q; mid++) {
    upd(qry[mid].sec.fir, qry[mid].sec.sec, qry[mid].fir);
    for (auto const &j : on[mid]) {
      int val = 0;
      for (auto const &k : adj[j]) {
        val += sum(k);
        if (val >= need[j]) break;
      }
      (val >= need[j]) ? r[j] = mid : l[j] = mid + 1;
    }
    on[mid].clear();
  }
}
// a resposta eh o l[i] para cada busca
```

### alienstrick.cpp

32 lines

```
pi solve(vector<int> &v, int lambda) {
  // associar um custo lambda para ser subtraido quando
       realizamos uma operacao
  // dp[i] - melhor profit que tivemos considerando as i
       primeiras posicoes
  // cnt[i] - quantas operacoes utilizamos para chegar no valor
        de dp[i]
  vector<int> dp(n + 1);
  vector<int> cnt(n + 1);
  dp[0] = 0;
  cnt[0] = 0;
  for (int i = 1; i <= n; i++) {
    dp[i] = dp[i - 1];
    cnt[i] = cnt[i - 1];
    int id = i - 1;
    dp[i] += v[id];
    int lo = max(0ll, id - l + 1);
```

```
    int s = dp[lo] + (id - lo + 1) - lambda;
    if (s > dp[i]) {
      dp[i] = s;
      cnt[i] = cnt[lo] + 1;
    }
  }
  return {dp[n], cnt[n]};
}
int aliens_trick(vector<int> &v) {
  int l = 0, r = n;
  while (l < r) {
    int mid = (l + r) >> 1;
    pi ans = solve(v, mid);
    (ans.sec > k) ? l = mid + 1 : r = mid;
  }
  pi ans = solve(v, l);
  return ans.fir + (l * k);
}
```

## cht.cpp

63 lines

```
struct line {
  mutable int m, b, p;
  bool operator<(const line &o) const {
    if (m != o.m) return m < o.m;
    return b < o.b;
  }
  bool operator<(const int x) const { return p < x; }
  int eval(int x) const { return m * x + b; }
  int inter(const line &o) const {
    int x = b - o.b, y = o.m - m;
    return (x / y) - ((x ^ y) < 0 && x % y);
  }
};
struct cht {
  int INF = 1e18;
  multiset<line, less<>> l;
  void add(int m, int b) {
    auto y = l.insert({m, b, INF});
    auto z = next(y);
    if (z != l.end() && y->m == z->m) {
      l.erase(y);
      return;
    }
    if (y != l.begin()) {
      auto x = prev(y);
      if (x->m == y->m) x = l.erase(x);
    }
    while (1) {
      if (z == l.end()) {
        y->p = INF;
        break;
      }
      y->p = y->inter(*z);
      if (y->p < z->p)
        break;
      else
        z = l.erase(z);
    }
    if (y == l.begin()) return;
    z = y;
    auto x = --y;
    while (1) {
      int ninter = x->inter(*z);
      if (ninter <= x->p)
        x->p = ninter;
      else {
        l.erase(z);
        break;
      }
      if (x == l.begin()) break;
      y = x;
      x--;
      if (x->p < y->p)
        break;
      else
        l.erase(y);
    }
  }
  int get(int x) {
    if (l.empty()) return 0;
    return l.lower_bound(x)->eval(x);
  }
};
```

## expectedvaluedp.cpp

26 lines

```
double dp[MAXN][MAXN][MAXN];
// sushi problem
double solve(int i, int j, int k) {
  if (!i && !j && !k) return dp[i][j][k] = 0;
  if (dp[i][j][k] != -1) return dp[i][j][k];
  /*
  It is well-known from statistics that for the geometric
       distribution
  (counting number of trials before a success, where each
       independent trial is probability p)
  the expected value is i / p
  */
  double p = ((double)(i + j + k) / n);
  double ret = 1 / p;  // expected number of trials before a
       success
  if (i) {
    double prob = (double)i / (i + j + k);  // probabilidade de
         ser um prato com um sushi
    ret += (solve(i - 1, j, k) * prob);
  }
  if (j) {
    double prob = (double)j / (i + j + k);  // probabilidade de
         ser um prato com dois sushis
    ret += (solve(i + 1, j - 1, k) * prob);
  }
  if (k) {
    double prob = (double)k / (i + j + k);  // probabilidade de
         ser um prato com tres sushis
    ret += (solve(i, j + 1, k - 1) * prob);
  }
  return dp[i][j][k] = ret;
}
```

## kadane.cpp

19 lines

```
int kadane(vector<int> v) {
  int n = v.size(), ans = 0, max_here = 0;
  for (int i = 0; i < n; i++) {
    max_here += v[i];
    if (ans < max_here) ans = max_here;
    if (max_here < 0) max_here = 0;
  }
  return ans;
}
int kadane_circular(vector<int> v) {
  int n = v.size(), max_kadane = kadane(v);
  int max_wrap = 0, i;
  for (i = 0; i < n; i++) {
    max_wrap += v[i];
    v[i] = -v[i];
  }
  max_wrap += kadane(v);
  return max(max_wrap, max_kadane);
}
```

## divideandconquerdp.cpp

16 lines

```
void compute(int l, int r, int optl, int optr, int i) {
  if (l > r) return;
  int mid = (l + r) >> 1;
  pair<int, int> ans = {1e18, -1};  // dp, k
  for (int q = optl; q <= min(mid, optr); q++) {
    if (q > 0)
      ans = min(ans, {dp[i - 1][q - 1] + cost(q, mid), q});
    else
      ans = min(ans, {cost(q, mid), q});
  }
  dp[i][mid] = ans.fir;
  compute(l, mid - 1, optl, ans.sec, i);
  compute(mid + 1, r, ans.sec, optr, i);
}
// dp[i][j] = min(dp[i - 1][k] + c(k, j)), para algum k <= j
// O(k * n * log(n))
```

## sosdp.cpp

20 lines

```
// nesse caso, f[x] eh a funcaoo que soma:
// todos os a[i], tal que, (x & i) == i)
for (int mask = 0; mask < (1 << m); mask++) {
  f[mask] = a[mask];
}
for (int i = 0; i < m; ++i) {
  for (int mask = 0; mask < (1 << m); mask++) {
    if (mask & (1 << i)) f[mask] += f[mask ^ (1 << i)];
  }
}
// nesse caso, f[x] eh a funcao que soma:
// todos os a[i], tal que, (x & i) == x)
for (int mask = 0; mask < (1 << m); mask++) {
  f[mask] = a[mask];
}
for (int i = 0; i < m; ++i) {
  for (int mask = 0; mask < (1 << m); mask++) {
    if (!(mask & (1 << i))) f[mask] += f[mask ^ (1 << i)];
  }
}
```

## lis.cpp

13 lines

```
int lis() {
  vector<int> q;
  for (int i = 0; i < v.size(); i++) {
    vector<int>::iterator it = lower_bound(q.begin(), q.end(),
         v[i]);
    if (it == q.end())
      q.pb(v[i]);
    else
      *it = v[i];
  }
  for (int i = 0; i < q.size(); i++) cout << q[i] << " ";
  cout << endl;
  return q.size();
}
```

## largestsquareofones.cpp

9 lines

```
int ans = 0;
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    dp[i][j] = v[i][j];
```

```
    if (i && j && dp[i][j]) dp[i][j] = min({dp[i][j - 1], dp[i
        - 1][j], dp[i - 1][j - 1]}) + 1;
    ans = max(ans, dp[i][j]);
  }
}
cout << ans * ans << endl;
```

## steinertree.cpp

16 lines

```
for (int i = 0; i <= 100; i++)
  for (int j = 0; j < (1 << 9); j++) dp[i][j] = 1e9;
for (int i = 0; i < k; i++)
  for (int j = 0; j < v.size(); j++) dp[j][1 << i] = d(cap[i],
      v[j]);
for (int mask = 2; mask < (1 << k); mask++) {
  for (int i = 0; i < v.size(); i++) {
    for (int mask2 = 1; mask2 < mask; mask2++) {
      if (!((mask & mask2) == mask2)) continue;
      int mask3 = mask ^ mask2;
      dp[i][mask] = min(dp[i][mask], dp[i][mask2] + dp[i][mask3
          ]);
    }
    for (int j = 0; j < v.size(); j++) dp[j][mask] = min(dp[j][
        mask], dp[i][mask] + d(v[i], v[j]));
  }
}
double ans = 1e9;
for (int i = 0; i <= 100; i++) ans = min(ans, dp[i][(1 << k) -
    1]);
```

## subsetsum.cpp

89 lines

```
// subset sum com bitset de tamanho variado
// with n <= 10^6
template <int len = 1>
int subset_sum(int n, int h) {
  if (n >= len) {
    return subset_sum<std::min(len * 2, (int)MAXN)>(n, h);
  }
  bitset<len> dp;
  dp[0] = 1;
  for (auto const &x : w) {
    dp = dp | (dp << x);
  }
  return dp._Find_next(max(0ll, h - 1));  // retorna o proximo
      bit setado apos a posicao passada como parametro
}
int solve(vector<int> &w, int tot, int h) {
  // tot -> soma de todos os elementos de w
  // h -> valor desejado
  // quero retornar o menor valor x >= h, tal que existe um
      subset com soma x em w
  if (!w.size()) return 0;
  sort(w.rbegin(), w.rend());
  if (w[0] * 2 >= tot) return w[0];
  int n = w.size();
  w.pb(0);
  vector<int> aux;
  int p = 0;
  for (int i = 1; i <= n; i++) {
    if (w[i] != w[i - 1]) {
      int cnt = i - p;
      int x = w[i - 1];
      int j = 1;
      while (j < cnt) {
        aux.pb(x * j);
        cnt -= j;
        j *= 2;
      }
      aux.pb(x * cnt);
      p = i;
    }
  }
  swap(aux, w);
  return subset_sum(tot, h);
}
int f[MAXN];      // f[i] -> quantos "itens" com valor i tem
bitset<MAXN> dp;  // dp[i] = 1, se existe um subset com soma i
// garantir que a soma de todo mundo seja < MAXN
void subset_sum(vector<int> &v) {
  for (auto const &i : v) {
    f[i]++;
  }
  dp[0] = 1;
  for (int i = 1; i < MAXN; i++) {
    while (f[i] > 2) {
      f[i * 2]++;
      f[i] -= 2;
    }
    while (f[i]--) dp |= (dp << i);
  }
}
/*
 Given N non-negative integer weights w and a non-negative
     target t,
 computes the maximum S <= t such that S is the sum of some
     subset of the weights.
 O(N * max(w[i]))
*/
int knapsack(vector<int> w, int t) {
  int a = 0, b = 0;
  while (b < w.size() && a + w[b] <= t) {
    a += w[b++];
  }
  if (b == w.size()) {
    return a;
  }
  int m = *max_element(w.begin(), w.end());
  vector<int> u, v(2 * m, -1);
  v[a + m - t] = b;
  for (int i = b; i < w.size(); i++) {
    u = v;
    for (int x = 0; x < m; x++) {
      v[x + w[i]] = max(v[x + w[i]], u[x]);
    }
    for (int x = 2 * m; --x > m;) {
      for (int j = max(0ll, u[x]); j < v[x]; j++) v[x - w[j]] =
          max(v[x - w[j]], j);
    }
  }
  a = t;
  while (v[a + m - t] < 0) {
    a--;
  }
  return a;
}
```

## bitmasks.cpp

19 lines

```
// quantidade de bits setados na mask
cout << __builtin_popcount(mask) << endl;
// quando eh necessario percorrer todas as submasks ate (1 << n
    )
// e fazer algo com todas as submasks dessa mask O(3^n)
for (int m = 0; m < (1 << n); m++) {
  for (int s = m; s; s = (s - 1) & m) {
    // alguma coisa aqui sabendo que mask s eh uma submask de m
  }
}
// comprimindo as masks de um vector baseada em uma mask
    qualquer
for (int i = 0; i < masks.size(); i++) {
  int compressed = 0, curr_bit = 0;
  for (int j = 0; j < n; j++) {
    if (!(mask & (1LL << j))) continue;
    if (masks[i] & (1LL << j)) compressed |= (1LL << curr_bit);
    curr_bit++;
  }
  // alguma coisa sabendo que a mask compressed eh a mask
      comprimida da mask atual
}
```

## hanoi.cpp

18 lines

```
vector<pair<char, char>> ans;
void solve(int n, char a, char b, char c) {
  if (n == 0) return;
  solve(n - 1, a, c, b);
  ans.pb({a, b});
  solve(n - 1, c, b, a);
}
// chamar pra sol
solve(n, 'A', 'C', 'B');
// 3 pilhas, sendo a pilha A com n discos e as outras duas
    pilhas vazias
// quero mover todo mundo da pilha A para a pilha C
// em cada movimento, vc tira o disco do topo de uma pilha e
    poe no topo de outra pilha
// desde que o raio do disco seja menor do que o raio do disco
    que ta no topo da outra pilha
// os n discos tem raios distintos aos pares
// e printar os movimentos em um par [to, from]
// numero minimo pra resolver pros primeiros n
// 1, 3, 7, 15, 31, 63, 127, 255
// f(n) = 2^n - 1
```

## stacktrick.cpp

9 lines

```
stack<pi> s;
for (int i = n - 1; i >= 0; i--) {
  while (!s.empty() && s.top().fir <= v[i]) s.pop();
  (!s.empty()) ? ans[i] = s.top().sec : ans[i] = -1;
  s.push({v[i], i});
}
// for each index (0 <= i < n), find another index (0 <= j < n)
// which v[j] > v[i] and j > i and j is as close as possible to
    i
// if this index does not exist, print -1
```

## spraguegrundy.cpp

27 lines

```
vector<int> v = {2, 3, 4, 5, 6};
unordered_map<int, bool> vis;
unordered_map<int, int> dp;
int g(int x)  // achar o grundy number na marra
{
  if (x == 0) return 0;
  vector<bool> ok(4, 0);
  int mex = 0;
  for (auto const &i : v) {
    int curr = g(x / i);
    if (curr < 4) ok[curr] = 1;
    while (ok[mex]) mex++;
  }
  vis[x] = 1;
  return dp[x] = mex;
}
```

```
int n;
cin >> n;
int x = 0;
for (int i = 0; i < n; i++) {
  int k;
  cin >> k;
  x ^= solve(k);
}
(x > 0) ? cout << "Henry\n" : cout << "Derek\n";
// nim classico -> o jogador que comeca ganha se o xor dos
     tamanhos das pilhas for != 0
// teorema sprague-grundy (transformar um jogo qualquer em nim)
```

## rectangleunion.cpp

94 lines

```
vector<int> x_vals;
struct segtree {
  vector<int> seg, tag;
  segtree() {
    seg.assign(8 * x_vals.size(), 0);
    tag.assign(8 * x_vals.size(), 0);
  }
  void add(int ql, int qr, int x, int v, int l, int r) {
    if (qr <= l || r <= ql) {
      return;
    }
    if (ql <= l && r <= qr) {
      tag[v] += x;
      if (tag[v] == 0) {
        if (l != r)
          seg[v] = seg[v << 1] + seg[(v << 1) | 1];
        else
          seg[v] = 0;
      } else {
        seg[v] = x_vals[r] - x_vals[l];
      }
    } else {
      int mid = (l + r) >> 1;
      add(ql, qr, x, (v << 1), l, mid);
      add(ql, qr, x, ((v << 1) | 1), mid, r);
      if (tag[v] == 0 && l != r) seg[v] = seg[v << 1] + seg[(v
          << 1) | 1];
    }
  }
  int qry() { return seg[1]; }
  void upd(int l, int r, int x) { add(l, r, x, 1, 0, x_vals.
       size()); }
};
struct rect {
  int x1, y1, x2, y2;
};
struct event {
  int time, l, r, type;
  bool operator<(const event &b) {
    if (time != b.time) return time < b.time;
    return type > b.type;
  }
};
const int inf = 1e9;
signed main() {
  int n;
  cin >> n;
  vector<rect> v(n);
  for (int i = 0; i < n; i++) {
    cin >> v[i].x1 >> v[i].y1 >> v[i].x2 >> v[i].y2;
    x_vals.pb(v[i].x1);
    x_vals.pb(v[i].x2);
  }
  // comprime o x
  sort(x_vals.begin(), x_vals.end());
  x_vals.erase(unique(x_vals.begin(), x_vals.end()), x_vals.end
       ());
  vector<event> ev;
  for (int i = 0; i < n; i++) {
    v[i].x1 = lower_bound(x_vals.begin(), x_vals.end(), v[i].x1
         ) - x_vals.begin();
    v[i].x2 = lower_bound(x_vals.begin(), x_vals.end(), v[i].x2
         ) - x_vals.begin();
    ev.pb({v[i].y1, v[i].x1, v[i].x2, 0});  // adicao
    ev.pb({v[i].y2, v[i].x1, v[i].x2, 1});  // remocao
  }
  segtree s;
  sort(ev.begin(), ev.end());
  int area = 0, l = -inf;
  for (auto const &i : ev) {
    if (l == -inf) {
      l = i.time;
      s.upd(i.l, i.r, 1);
    } else if (i.type == 1) {
      int curr = s.qry();
      s.upd(i.l, i.r, -1);
      if (s.qry() != curr) {
        int new_t = (s.qry() == 0) ? -inf : i.time;
        int lo = l, hi = i.time - 1;
        area += ((hi - lo + 1) * curr);
        l = new_t;
      }
    } else {
      int curr = s.qry();
      s.upd(i.l, i.r, 1);
      if (s.qry() != curr) {
        int lo = l, hi = i.time - 1;
        area += ((hi - lo + 1) * curr);
        l = i.time;
      }
    }
  }
  cout << area << endl;
  return 0;
}
// n <= 5 * 10^5
// 0 <= x, y <= 10^9
// comprime coordenada no x pra montar a segtree dos valores de
     x
// faz o line sweep pelo y
```

## prefixsum2d.cpp

9 lines

```
int v[1001][1001];
int p[1001][1001];
int qry(int x1, int y1, int x2, int y2) { return p[x2 + 1][y2 +
     1] - p[x2 + 1][y1] - p[x1][y2 + 1] + p[x1][y1]; }
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    p[i + 1][j + 1] = p[i][j + 1] + p[i + 1][j] - p[i][j];
    p[i + 1][j + 1] += v[i][j];
  }
}
```

## bellmanfordinequalities.cpp

48 lines

```
struct solver {
  const int inf = 1e18;
  int n, src;
  vector<int> d;
  vector<pii> edges;
  solver(int _n)  // recebe o numero de variaveis, indexadas de
       0 ate _n - 1
  {
    src = _n;  // aqui denotaremos _n como o source
    n = _n + 1;
    for (int i = 0; i < _n; i++)  // arestas de source para
         cada um dos vertices com custo 0
    {
      edges.pb({{src, i}, 0});
    }
  }
  bool solve()  // bellman ford
  {
    d.assign(n, inf);
    d[src] = 0;
    int x;
    for (int i = 0; i < n; i++) {
      x = -1;
      for (auto const &e : edges) {
        auto [a, b] = e.fir;
        int cost = e.sec;
        if (d[a] < inf) {
          if (d[b] > d[a] + cost) {
            d[b] = max(-inf, d[a] + cost);
            x = b;
          }
        }
      }
    }
    return (x == -1);  // false se tem ciclo negativo
  }
  void add_constraint_leq(int i, int j, int c)  // value_i -
       value_j <= c
  {
    edges.pb({{j, i}, c});
  };
  void add_constraint_geq(int i, int j, int c)  // value_i -
       value_j >= c
  {
    edges.pb({{i, j}, -c});
  };
  void add_constraint_eq(int i, int j, int c)  // value_i -
       value_j = c
  {
    add_constraint_leq(i, j, c);
    add_constraint_geq(i, j, c);
  };
};
```