Contents

1 Basic

1.1 .vimrc

```
syntax on
set nu ai bs=2 sw=2 ts=2 et ve=all cb=unnamed mouse=a
    ruler incsearch hlsearch
```

1.2 IncStack

```
//stack resize (linux)
#include <sys/resource.h>
void increase_stack_size() {
  const rlim_t ks = 64*1024*1024;
  struct rlimit rl;
  int res=getrlimit(RLIMIT_STACK, &rl);
  if(res==0){
    if(rl.rlim_cur<ks){
      rl.rlim_cur=ks;
      res=setrlimit(RLIMIT_STACK, &rl);
    }
  }
```

1.3 IncStack windows

```
//stack resize
asm( "mov %0,%%esp\n" ::"g"(mem+10000000) );
//change esp to rsp if 64-bit system
```

1.4 random

```
#include <random>
mt19937 rng(0x5EED);
int randint(int lb, int ub)
{ return uniform_int_distribution<int>(lb, ub)(rng); }
```

1.5 time

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

using namespace std;

int main() {
  clock_t t;
  t = clock();
  // code here
  t = clock() - t;
  cout << 1.0 * t / CLOCKS_PER_SEC << "\n";

  // execute time for entire program
  cout << 1.0 * clock() / CLOCKS_PER_SEC << "\n";
}
```

2 Math

2.1 basic

```
PLL exd_gcd(LL a, LL b) { // what about b.zero? = =
  if (a % b == 0) return {0, 1};
  PLL T = exd_gcd(b, a % b);
  return {T.second, T.first - a / b * T.second};
}
LL powmod(LL x, LL p, LL mod) {
  LL s = 1, m = x % mod;
  for (; p; m = m * m % mod, p >>= 1)
    if (p&1) s = s * m % mod; // or consider int128
  return s;
}
```

```
LL LLmul(LL x, LL y, LL mod) {
  LL m = x, s = 0;
  for (; y; y >>= 1, m <<= 1, m = m >= mod? m - mod: m)
    if (y&1) s += m, s = s >= mod? s - mod: s;
  return s;
}
LL dangerous_mul(LL a, LL b, LL mod){ // 10 times faster than the above in average, but could be prone to wrong answer (extreme low prob?)
  return (a * b - (LL)((long double)a * b / mod) * mod) % mod;
}
vector<LL> linear_inv(LL p, int k) { // take k
  vector<LL> inv(min(p, 1ll + k));
  inv[1] = 1;
  for (int i = 2; i < inv.size(); ++i)
    inv[i] = (p - p / i) * inv[p % i] % p;
  return inv;
}

tuple<int, int, int> ext_gcd(int a, int b) {
  if (!b) return {1, 0, a};
  int x, y, g;
  tie(x, y, g) = ext_gcd(b, a % b);
  return {y, x - a / b * y, g};
}
```

2.2 BigNum

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

struct BigNum {
  typedef long long ll;

  int sign;
  ll B; // TODO: assert(N * B * B < LL_LIMIT) if mul is used
  int BW; // base width
  vector<ll> cells;

  BigNum(string s = "0", ll b = 10000) : sign(1), B(b), BW(ceil(log10(b))) {
    if (s[0] == '-') sign = -1, s = s.substr(1);
    cells.resize((s.size() + BW - 1) / BW);
    for (int i = 0; i < cells.size(); ++i) {
      int lb = max(0, int(s.size()) - (i + 1) * BW);
      int len = min(BW, int(s.size()) - i * BW);
      cells[i] = stoi(s.substr(lb, len));
    }
  }
  BigNum(const vector<ll> &v, ll b = 10000) : sign(1), B(b), BW(ceil(log10(b))), cells(v) {}

  friend bool operator<(const BigNum &a, const BigNum &b) {
    if (a.sign != b.sign) return a.sign < b.sign;
    if (a.cells.size() != b.cells.size()) return a.cells.size() < b.cells.size();
    for (int i = a.cells.size() - 1; ~i; --i)
      if (a.cells[i] != b.cells[i]) return a.cells[i] < b.cells[i];
    return false;
  }
  friend bool operator==(const BigNum &a, const BigNum &b) { return a.sign == b.sign && a.cells == b.cells; }
  friend bool operator!=(const BigNum &a, const BigNum &b) { return !(a == b); }
  friend bool operator<=(const BigNum &a, const BigNum &b) { return !(b < a); }
  friend bool operator>(const BigNum &a, const BigNum &b) { return b < a; }
  friend bool operator>=(const BigNum &a, const BigNum &b) { return !(a < b); }

  BigNum& normal(int result_sign = 1) {
    ll c = 0;
    for (int i = 0; i < cells.size(); ++i) {
      if (cells[i] < 0) {
        if (i + 1 == cells.size()) cells.emplace_back(0);
        ll u = (abs(cells[i]) + B - 1) / B;
        cells[i + 1] -= u;
        cells[i] += u * B;
      }
      ll u = cells[i] + c;
      cells[i] = u % B;
      c = u / B;
    }
    for (; c; c /= B) cells.emplace_back(c % B);
    while (cells.size() > 1 && cells.back() == 0) cells.pop_back();
    sign = result_sign;
    return *this;
  }

  static vector<ll> add(const vector<ll> &a, const vector<ll> &b, int al = -1, int ar = -1, int bl = -1, int br = -1) {
    if (al == -1) al = 0, ar = a.size(), bl = 0, br = b.size();
    vector<ll> c(max(ar - al, br - bl));
    for (int i = 0; i < c.size(); ++i)
      c[i] = (al + i < a.size() ? a[al + i] : 0) + (bl + i < b.size() ? b[bl + i] : 0);
    return c;
  }
  static vector<ll> sub(const vector<ll> &a, const vector<ll> &b, int al = -1, int ar = -1, int bl = -1, int br = -1) {
    if (al == -1) al = 0, ar = a.size(), bl = 0, br = b.size();
    vector<ll> c(max(ar - al, br - bl));
    for (int i = 0; i < c.size(); ++i)
      c[i] = (al + i < a.size() ? a[al + i] : 0) - (bl + i < b.size() ? b[bl + i] : 0);
    return c;
  }
  static vector<ll> cat_zero(const vector<ll> &a, int k) {
    vector<ll> b(a.size() + k);
    for (int i = 0; i < a.size(); ++i) b[k + i] = a[i];
    return b;
  }

  friend BigNum operator+(BigNum x, BigNum y) {
    if (x.sign == y.sign) return BigNum(add(x.cells, y.cells)).normal();
    if (x.sign == -1) swap(x, y);
    y.sign = 1;
    if (x >= y) return BigNum(sub(x.cells, y.cells)).normal();
    return BigNum(sub(y.cells, x.cells)).normal(-1);
  }
  friend BigNum operator-(BigNum x, BigNum y) {
    y.sign *= -1;
    return x + y;
  }
  friend BigNum operator*(BigNum x, BigNum y) {
    if (x.cells.size() < y.cells.size()) swap(x, y);
    int nn = 31 - __builtin_clz(int(x.cells.size())) + (__builtin_popcount(int(x.cells.size())) > 1);
    function<vector<ll>(const vector<ll> &, const vector<ll> &, int, int, int, int)>
    karatsuba = [&](const vector<ll> &a, const vector<ll> &b, int al, int ar, int bl, int br) {
      if (al + 256 >= ar) {
        vector<ll> r(ar - al << 1);
        for (int i = 0; i < ar - al; ++i)
          for (int j = 0; j < br - bl; ++j)
            r[i + j] += a[al + i] * b[bl + j];
        return r;
      }
      vector<ll> z1 = karatsuba(a, b, al + ar >> 1, ar, bl + br >> 1, br);
      vector<ll> z2 = karatsuba(a, b, al, al + ar >> 1, bl, bl + br >> 1);
```

```
          vector<ll> p = cat_zero(z1, ar - al);
          vector<ll> a12 = add(a, a, al, al + ar >> 1,
               al + ar >> 1, ar);
          vector<ll> b12 = add(b, b, bl, bl + br >> 1,
               bl + br >> 1, br);
          vector<ll> ab12 = karatsuba(a12, b12, 0, a12
              .size(), 0, b12.size());
          vector<ll> q1 = sub(ab12, z1);
          vector<ll> q2 = sub(q1, z2);
          vector<ll> q = cat_zero(q2, ar - al >> 1);
          vector<ll> r1 = add(p, q);
          vector<ll> r = add(r1, z2);
          return r;
        };
    x.cells.resize(1 << nn);
    y.cells.resize(1 << nn);
    vector<ll> k = karatsuba(x.cells, y.cells, 0, 1 <<
         nn, 0, 1 << nn);
    return BigNum(k).normal(x.sign * y.sign);
  }

  friend ostream& operator<<(ostream &os, BigNum x) {
    if (x.sign == -1) os << '-';
    for (auto it = x.cells.rbegin(); it != x.cells.
        rend(); ++it) {
      if (it == x.cells.rbegin()) os << *it;
      else os << setw(x.BW) << setfill('0') << *it;
    }
    return os;
  }
  friend istream& operator>>(istream &is, BigNum &x) {
    string s;
    is >> s;
    x = BigNum(s);
    return is;
  }
};

signed main() {
  BigNum a, b;
  cin >> a >> b;

  BigNum ab("1");
  for (BigNum i; i < b; i = i + BigNum("1")) ab = ab *
       a;

  BigNum ba("1");
  for (BigNum i; i < a; i = i + BigNum("1")) ba = ba *
       b;

  cout << ab - ba << endl;

  return 0;
}
```

2.3 FFT

```
/* p == (a << n) + 1
   g = pow(root, (p - 1) / n)
   n    1<<n        p          a    root
   5    32          97         3    5
   6    64          193        3    5
   7    128         257        2    3
   8    256         257        1    3
   9    512         7681       15   17
   10   1024        12289      12   11
   11   2048        12289      6    11
   12   4096        12289      3    11
   13   8192        40961      5    3
   14   16384       65537      4    3
   15   32768       65537      2    3
   16   65536       65537      1    3
   17   131072      786433     6    10
   18   262144      786433     3    10 (605028353,
        2308, 3)
   19   524288      5767169    11   3
   20   1048576     7340033    7    3
   20   1048576     998244353  952  3
   21   2097152     23068673   11   3
   22   4194304     104857601  25   3
   23   8388608     167772161  20   3
   24   16777216    167772161  10   3
   25   33554432    167772161  5    3 (1107296257, 33,
        10)
   26   67108864    469762049  7    3
*/

// w = root^a mod p for NTT
// w = exp(-complex<double>(0, 2) * PI / N) for FFT

template<typename F = complex<double>>
void FFT(vector<F> &P, F w, bool inv = 0) {
  int n = P.size();
  int lg = __builtin_ctz(n);
  assert(__builtin_popcount(n));

  for (int j = 1, i = 0; j < n - 1; ++j) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(P[i], P[j]);
  } //bit reverse

  vector<F> ws = {inv ? F{1} / w : w};
  for (int i = 1; i < lg; ++i) ws.push_back(ws[i - 1]
       * ws[i - 1]);
  reverse(ws.begin(), ws.end());

  for (int i = 0; i < lg; ++i) {
    for (int k = 0; k < n; k += 2<<i) {
      F base = F{1};
      for (int j = k; j < k + (1<<i); ++j, base = base
           * ws[i]) {
        auto t = base * P[j + (1<<i)];
        auto u = P[j];
        P[j] = u + t;
        P[j + (1<<i)] = u - t;
      }
    }
  }

  if (inv) for_each(P.begin(), P.end(), [&](F& a) { a
       = a / F(n); });
} //faster performance with calling by reference
```

2.4 FWT

```
vector<LL> fast_OR_transform(vector<LL> f, bool
    inverse) {
  for (int i = 0; (2 << i) <= f.size(); ++i)
    for (int j = 0; j < f.size(); j += 2 << i)
      for (int k = 0; k < (1 << i); ++k)
        f[j + k + (1 << i)] += f[j + k] * (inverse? -1
            : 1);
  return f;
}
vector<LL> rev(vector<LL> A) {
  for (int i = 0; i < A.size(); i += 2) swap(A[i], A[i
       ^ (A.size() - 1)]);
  return A;
}
vector<LL> fast_AND_transform(vector<LL> f, bool
    inverse) {
  return rev(fast_OR_transform(rev(f), inverse));
}
vector<LL> fast_XOR_transform(vector<LL> f, bool
    inverse) {
  for (int i = 0; (2 << i) <= f.size(); ++i)
    for (int j = 0; j < f.size(); j += 2 << i)
      for (int k = 0; k < (1 << i); ++k) {
        int u = f[j + k], v = f[j + k + (1 << i)];
        f[j + k + (1 << i)] = u - v, f[j + k] = u + v;
      }
  if (inverse) for (auto &a : f) a /= f.size();
  return f;
}
```

2.5 Lagrange Polynomial

```cpp
template<typename F>
struct Lagrange_poly {
  vector<F> fac, p;
  int n;
  Lagrange_poly(vector<F> p) : p(p) { // f(i) = p[i]
    n = p.size();
    fac.resize(n), fac[0] = 1;
    for (int i = 1; i < n; ++i) fac[i] = fac[i - 1] * F(i);
  }
  F operator()(F x) const {
    F ans(0), to_mul(1);
    for (int j = 0; j < n; ++j) to_mul = to_mul * (F(j) - x);
    assert(not(to_mul == F(0)));
    for (int j = 0; j < n; ++j) {
      ans = ans + p[j] * to_mul / (F(j) - x) / fac[n - 1 - j] / (j&1 ? -fac[j] : fac[j]);
    }
    return ans;
  }
};
```

2.6 Lucas

```cpp
LL fac[100000] = {1};
LL C(LL a, LL b, LL p) {
  for (int i = 1; i <= p; ++i) fac[i] = fac[i - 1] * i % p;
  LL ans = 1;
  for (;a; a /= p, b /= p) {
    LL A = a % p, B = b % p;
    if (A < B) return 0;
    (ans *= fac[A] * powmod(fac[B] * fac[A - B] % p, p - 2, p) % p) %= p;
  }
  return ans;
}
```

2.7 Miller Rabin with Pollard rho

```cpp
bool miller_rabin(LL n, int s = 7) {
  const LL wits[7] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
  auto witness = [=](LL a, LL n, LL u, int t) {
    LL x = powmod(a, u, n), nx; // use LLmul, remember
    for (int i = 0; i < t; ++i, x = nx){
      nx = LLmul(x, x, n);
      if (nx == 1 and x != 1 and x != n - 1) return true;
    }
    return x != 1;
  };
  if (n < 2) return 0;
  if (n&1^1) return n == 2;
  LL u = n - 1, t = 0, a; // n == (u << t) + 1
  while (u&1^1) u >>= 1, ++t;
  while (s--)
    if ((a = wits[s] % n) and witness(a, n, u, t))
      return 0;
  return 1;
}
// Pollard_rho
LL pollard_rho(LL n) {
  auto f = [=](LL x, LL n) { return LLmul(x, x, n) + 1; };
  if (n&1^1) return 2;
  while (true) {
    LL x = rand() % (n - 1) + 1, y = 2, d = 1;
    for (int sz = 2; d == 1; y = x, sz <<= 1)
      for (int i = 0; i < sz and d <= 1; ++i)
        x = f(x, n), d = __gcd(abs(x - y), n);
    if (d and n - d) return d;
  }
}
vector<pair<LL, int>> factor(LL m) {
  vector<pair<LL, int>> ans;
  while (m != 1) {
    LL cur = m;
    while (not miller_rabin(cur)) cur = pollard_rho(cur);
    ans.emplace_back(cur, 0);
    while (m % cur == 0) ++ans.back().second, m /= cur;
  }
  sort(ans.begin(), ans.end());
  return ans;
}
```

2.8 ModInt

```cpp
template <int mod>
struct ModInt {
  int val;
  int trim(int x) const { return x >= mod ? x - mod : x < 0 ? x + mod : x; }
  ModInt(int v = 0) : val(trim(v % mod)) {}
  ModInt(long long v) : val(trim(v % mod)) {}
  ModInt &operator=(int v) { return val = trim(v % mod), *this; }
  ModInt &operator=(const ModInt &oth) { return val = oth.val, *this; }
  ModInt operator+(const ModInt &oth) const { return trim(val + oth.val); }
  ModInt operator-(const ModInt &oth) const { return trim(val - oth.val); }
  ModInt operator*(const ModInt &oth) const { return 1 LL * val * oth.val % mod; }
  ModInt operator/(const ModInt &oth) const {
    function<int(int, int, int, int)> modinv = [&](int a, int b, int x, int y) {
      if (b == 0) return trim(x);
      return modinv(b, a - a / b * b, y, x - a / b * y);
    };
    return *this * modinv(oth.val, mod, 1, 0);
  }
  bool operator==(const ModInt &oth) const { return val == oth.val; }
  ModInt operator-() const { return trim(mod - val); }
  template<typename T> ModInt pow(T pw) {
    bool sgn = false;
    if (pw < 0) pw = -pw, sgn = true;
    ModInt ans = 1;
    for (ModInt cur = val; pw; pw >>= 1, cur = cur * cur) {
      if (pw&1) ans = ans * cur;
    }
    return sgn ? ModInt{1} / ans : ans;
  }
};
```

2.9 Mod Mul Group Order

```cpp
#include "Miller_Rabin_with_Pollard_rho.cpp"
LL phi(LL m) {
  auto fac = factor(m);
  return accumulate(fac.begin(), fac.end(), m, [](LL a, pair<LL, int> p_r) {
    return a / p_r.first * (p_r.first - 1);
  });
}
LL order(LL x, LL m) {
  // assert(__gcd(x, m) == 1);
  LL ans = phi(m);
  for (auto P: factor(ans)) {
    LL p = P.first, t = P.second;
    for (int i = 0; i < t; ++i) {
      if (powmod(x, ans / p, m) == 1) ans /= p;
      else break;
    }
  }
  return ans;
}
LL cycles(LL a, LL m) {
  if (m == 1) return 1;
```

```
  return phi(m) / order(a, m);
}
```

2.10 MongeDP

```
template<typename R> // return_type
struct MongeDP { // NOTE: if update like rolling dp, then enclose dp value in wei function and remove dp[] in R.H.S when updating stuff
  int n;
  vector<R> dp;
  vector<int> pre;
  function<bool(R, R)> cmp; // true is left better
  function<R(int, int)> w;  // w(i, j) = cost(dp[i] -> dp[j])
  MongeDP(int _n, function<bool(R, R)> c, function<R(int, int)> get_cost)
      : n(_n), dp(n + 1), pre(n + 1, -1), cmp(c), w(get_cost) {
    deque<tuple<int, int, int>> dcs; // decision
    dcs.emplace_back(0, 1, n); // transition from dp[0] is effective for [1, N]
    for (int i = 1; i <= n; ++i) {
      while (get<2>(dcs.front()) < i) dcs.pop_front(); // right bound is out-dated
      pre[i] = get<0>(dcs.front());
      dp[i] = dp[pre[i]] + w(pre[i], i); // best t is A[dcs.top(), i)
      while (dcs.size()) {
        int x, lb, rb;
        tie(x, lb, rb) = dcs.back();
        if (lb <= i) break; // will be pop_fronted soon anyway
        if (!cmp(dp[x] + w(x, lb), dp[i] + w(i, lb))) {
          dcs.pop_back();
          if (dcs.size()) get<2>(dcs.back()) = n;
        } else break;
      }
      int best = -1;
      for (int lb = i + 1, rb = n, x = get<0>(dcs.back()); lb <= rb; ) {
        int mb = lb + rb >> 1;
        if (cmp(dp[i] + w(i, mb), dp[x] + w(x, mb))) {
          best = mb;
          rb = mb - 1;
        } else lb = mb + 1;
      }
      if (~best) {
        get<2>(dcs.back()) = best - 1;
        dcs.emplace_back(i, best, n);
      }
    }
  }
  void ensure_monge_condition() {
    // Monge Condition: i <= j <= k <= l then w(i, l) + w(j, k) >(<)= w(i, k) + w(j, l)
    for (int i = 0; i <= n; ++i)
      for (int j = i; j <= n; ++j)
        for (int k = j; k <= n; ++k)
          for (int l = k; l <= n; ++l) {
            R w0 = w(i, l), w1 = w(j, k), w2 = w(i, k), w3 = w(j, l);
            assert(w0 + w1 >= w2 + w3); // if maximization, revert the sign
          }
  }
  R operator[](int x) { return dp[x]; }
};

/* Example:
  MongeDP<int64_t> mdp(N, [](int64_t x, int64_t y) {
      return x < y; },
                       [&](int x, int rb) {
                         auto abscub = [](int64_t x) { return abs(x * x * x); };
                         return abscub(A[rb - 1] - X[x]) + abscub(Y[x]);
                       });
  // mdp.ensure_monge_condition();

OR in case rolling dp, remember to remove dp[] in R.H.S. in lines 15, 20, 28 and do the following:
  vector<int64_t> dp(N + 1, 1LL << 60);
  dp[0] = 0;
  for (int i = 1; i < G + 1; ++i) {
    dp = MongeDP<int64_t>(N, [](int64_t x, int64_t y) { return x < y; },
                          [&](int x, int rb) {
                            return dp[x] + cost[x][rb];
                          }).dp;
  }

*/
```

2.11 Chinese Remainder Theorem

```
PLL CRT(PLL eq1, PLL eq2) {
  LL m1, m2, x1, x2;
  tie(x1, m1) = eq1, tie(x2, m2) = eq2;
  LL g = __gcd(m1, m2);
  if ((x1 - x2) % g) return {-1, 0}; // NO SOLUTION
  m1 /= g, m2 /= g;
  auto p = exd_gcd(m1, m2);
  LL lcm = m1 * m2 * g, res = mul(mul(p.first, (x2 - x1), lcm), m1, lcm) + x1;
  return {(res % lcm + lcm) % lcm, lcm};
}
```

2.12 Discrete Log

```
int discrete_log(int a, int m, int p) { // a**x = m mod p
  int magic = sqrt(p) + 2;
  map<int, int> mp;
  int x = 1;
  for (int i = 0; i < magic; ++i) {
    mp[x] = i;
    x = 1LL * x * a % p;
  }
  for (int i = 0, y = 1; i < magic; ++i) {
    int inv = get<0>(ext_gcd(y, p));
    if (inv < 0) inv += p;
    int u = 1LL * m * inv % p;
    if (mp.count(u)) return i * magic + mp[u];
    y = 1LL * y * x % p;
  }
  return -1;
}
```

2.13 Fast Linear Recurrence

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

template<typename T>
vector<T> fast_linear_recurrence(const vector<T> &t, long long p) { // O(lg(p) * t.size()**2)
  auto advance = [&](const vector<T> &u) {
    vector<T> v(t.size());
    v[0] = u.back() * t[0];
    for (int i = 1; i < t.size(); ++i) v[i] = u[i - 1] + u.back() * t[i];
    return v;
  };

  vector<vector<T>> kk(2 * t.size(), vector<T>(t.size())); // kk[i] = lambda(t ** i)
  kk[0][0] = 1;
  for (int i = 1; i < 2 * t.size(); ++i) kk[i] = advance(kk[i - 1]);
  if (p < kk.size()) return kk[p];

  auto square = [&](const vector<T> &u) {
    vector<T> v(2 * t.size());
```

```
    for (int j = 0; j < u.size(); ++j)
      for (int k = 0; k < u.size(); ++k)
        v[j + k] = v[j + k] + u[j] * u[k];
    for (int j = u.size(); j < v.size(); ++j)
      for (int k = 0; k < u.size(); ++k)
        v[k] = v[k] + v[j] * kk[j][k];
    v.resize(u.size());
    return v;
  };

  vector<T> m(kk[1]);
  for (int i = 62 - __builtin_clzll(p); ~i; --i) {
    m = square(m);
    if (p >> i & 1) m = advance(m);
  }

  return m;
}

signed main() { // 405 ms on CF
  vector<int> t(2000);
  t[0] = t[1] = 1; // f[i] = f[i - 2000] + f[i - 1999]
  auto m = fast_linear_recurrence<int>(t, (long long)
      1e18);

  vector<int> v(2000, 1); // f[i] = 1 for i < 2000
  int res = 0;
  for (int i = 0; i < m.size(); ++i) res += v[i] * m[i
      ];
  cout << res << endl;

  return 0;
}
```

2.14 Matrix

```
template<typename F>
struct Matrix {
  int rowNum, colNum;
  vector<vector<F>> cell;

  Matrix(int n) : rowNum(n), colNum(n) { // Identity
      matrix
    cell = vector<vector<F>>(n, vector<F>(n, 0));
    for (int i = 0; i < n; i++) cell[i][i] = F(1);
  }

  Matrix(int n, int m, int fill = 0) : rowNum(n),
      colNum(m) {
    cell.assign(n, vector<F>(m, fill));
  }

  Matrix(const Matrix &mat) : rowNum(mat.rowNum),
      colNum(mat.colNum) {
    cell = mat.cell;
  }

  vector<F>& operator[] (int i) { return cell[i]; }

  const vector<F>& operator[] (int i) const { return
      cell[i]; }

  Matrix& operator= (const Matrix &mat) {
    rowNum = mat.rowNum;
    colNum = mat.colNum;
    cell = mat.cell;
    return *this;
  }

  Matrix& operator*= (const Matrix &mat) {
    assert(colNum == mat.rowNum);
    Matrix res(rowNum, mat.colNum);
    for (int i = 0; i < rowNum; i++) {
      for (int j = 0; j < mat.colNum; j++) {
        for (int k = 0; k < colNum; k++) {
          res[i][j] += cell[i][k] * mat[k][j];
        }
      }
    }
    return *this = res;
  }

  Matrix& operator^= (long long p) {
    assert(rowNum == colNum && p >= 0);
    Matrix res(rowNum);
    for (; p; p >>= 1) {
      if (p&1) res *= *this;
      *this *= *this;
    }
    return *this = res;
  }

  friend istream& operator>> (istream &is, Matrix &mat
      ) {
    for (int i = 0; i < mat.rowNum; i++)
      for (int j = 0; j < mat.colNum; j++)
        is >> mat[i][j];
    return is;
  }

  friend ostream& operator<< (ostream &os, const
      Matrix &mat) {
    for (int i = 0; i < mat.rowNum; i++)
      for (int j = 0; j < mat.colNum; j++)
        os << mat[i][j] << " \n"[j == mat.colNum - 1];
    return os;
  }

  Matrix operator* (const Matrix &b) {
    Matrix res(*this);
    return (res *= b);
  }

  Matrix operator^ (const long long p) {
    Matrix res(*this);
    return (res ^= p);
  }
};
```

2.15 Determinant

```
template<typename T>
vector<T> operator-(vector<T> A, vector<T> B) {
  for (int i = 0; i < A.size(); ++i) A[i] = A[i] - B[i
      ];
  return A;
}

template<typename T>
vector<T> operator*(vector<T> A, T mul) {
  for (int i = 0; i < A.size(); ++i) A[i] = A[i] * mul
      ;
  return A;
}

template<typename T>
vector<T> operator/(vector<T> A, T mul) {
  for (int i = 0; i < A.size(); ++i) A[i] = A[i] / mul
      ;
  return A;
}


template<typename T>
T det(Matrix<T> A) {
  int N = A.rowNum;
  T ans(1);
  for (int r = 0; r < N; ++r) {
    if (A[r][r] == T(0)) return T(0);
    ans = ans * A[r][r];
    for (int pvt = r + 1; pvt < N; ++pvt) {
      A[pvt] = A[pvt] - A[r] * A[pvt][r] / A[r][r];
    }
  }
  return ans;
}
```

2.16 Number Theory Functions

```cpp
vector<int> linear_sieve(const int UPBD) {
  vector<int> primes, last_prime(UPBD, 0);
  for (int p = 2; p < UPBD; ++p) {
    if (not last_prime[p]) primes.push_back(p), last_prime[p] = p;
    for (int j = 0; primes[j] * p < UPBD; ++j) {
      last_prime[primes[j] * p] = primes[j];
      if (p % primes[j] == 0) break;
    }
  }
  return last_prime;
}
template<typename T> vector<T> make_mobius(T limit) {
  auto last_prime = linear_sieve(limit);
  vector<T> mobius(limit, 1);
  mobius[0] = 0;
  for (T p = 2; p < limit; ++p) {
    if (last_prime[p] == last_prime[p / last_prime[p]]) mobius[p] = 0;
    else mobius[p] = mobius[p / last_prime[p]] * -1;
  }
  return mobius;
}
```

2.17 Polynomail root

```cpp
const double eps = 1e-12;
const double inf = 1e+12;
double a[10], x[10];
int n;
int sign(double x) { return (x < -eps) ? (-1) : (x > eps); }
double f(double a[], int n, double x) {
  double tmp = 1, sum = 0;
  for (int i = 0; i <= n; i++) {
    sum = sum + a[i] * tmp;
    tmp = tmp * x;
  }
  return sum;
}
double binary(double l, double r, double a[], int n) {
  int sl = sign(f(a, n, l)), sr = sign(f(a, n, r));
  if (sl == 0) return l;
  if (sr == 0) return r;
  if (sl * sr > 0) return inf;
  while (r - l > eps) {
    double mid = (l + r) / 2;
    int ss = sign(f(a, n, mid));
    if (ss == 0) return mid;
    if (ss * sl > 0)
      l = mid;
    else
      r = mid;
  }
  return l;
}
void solve(int n, double a[], double x[], int &nx) {
  if (n == 1) {
    x[1] = -a[0] / a[1];
    nx = 1;
    return;
  }
  double da[10], dx[10];
  int ndx;
  for (int i = n; i >= 1; i--) da[i - 1] = a[i] * i;
  solve(n - 1, da, dx, ndx);
  nx = 0;
  if (ndx == 0) {
    double tmp = binary(-inf, inf, a, n);
    if (tmp < inf) x[++nx] = tmp;
    return;
  }
  double tmp;
  tmp = binary(-inf, dx[1], a, n);
  if (tmp < inf) x[++nx] = tmp;
  for (int i = 1; i <= ndx - 1; i++) {
    tmp = binary(dx[i], dx[i + 1], a, n);
    if (tmp < inf) x[++nx] = tmp;
  }
  tmp = binary(dx[ndx], inf, a, n);
  if (tmp < inf) x[++nx] = tmp;
}
int main() {
  scanf("%d", &n);
  for (int i = n; i >= 0; i--) scanf("%lf", &a[i]);
  int nx;
  solve(n, a, x, nx);
  for (int i = 1; i <= nx; i++) printf("%.6f\n", x[i]);
}
```

2.18 Subset Zeta Transform

```cpp
// if f is add function:
// low2high = true -> zeta(a)[s] = sum(a[t] for t in s)
// low2high = false -> zeta(a)[t] = sum(a[s] for t in s)
// else if f is sub function, you get inverse zeta function
template<typename T>
vector<T> subset_zeta_transform(int n, vector<T> a, function<T(T, T)> f, bool low2high = true) {
  assert(a.size() == 1 << n);
  if (low2high) {
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < 1 << n; ++j)
        if (j >> i & 1)
          a[j] = f(a[j], a[j ^ 1 << i]);
  } else {
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < 1 << n; ++j)
        if (~j >> i & 1)
          a[j] = f(a[j], a[j | 1 << i]);
  }
  return a;
}
```

3 Data Structure

3.1 Disjoint Set

```cpp
struct Dsu {
  struct node_struct {
    int par, size;
    node_struct(int p, int s) : par(p), size(s) {}
    void merge(node_struct &b) {
      b.par = par;
      size += b.size;
    }
  };
  vector<node_struct> nodes;
  stack<tuple<int, int, node_struct, node_struct>> stk;
  Dsu(int n) {
    nodes.reserve(n);
    for (int i = 0; i < n; ++i) nodes.emplace_back(i, 1);
  }
  int anc(int x) {
    while (x != nodes[x].par) x = nodes[x].par;
    return x;
  }
  bool unite(int x, int y) {
    int a = anc(x);
    int b = anc(y);
    stk.emplace(a, b, nodes[a], nodes[b]);
    if (a == b) return false;
    if (nodes[a].size < nodes[b].size) swap(a, b);
    nodes[a].merge(nodes[b]);
    return true;
  }
  void revert(int version = -1) {  // 0 index
    if (version == -1) version = stk.size() - 1;
    for (; stk.size() != version; stk.pop()) {
      nodes[get<0>(stk.top())] = get<2>(stk.top());
```

```
      nodes[get<1>(stk.top())] = get<3>(stk.top());
    }
  }
};
```

3.2 Heavy Light Decomposition

```
struct HLD {
  using Tree = vector<vector<int>>;
  vector<int> par, head, vid, len, inv;

  HLD(const Tree &g) : par(g.size()), head(g.size()), vid(g.size()), len(g.size()), inv(g.size()) {
    int k = 0;
    vector<int> size(g.size(), 1);
    function<void(int, int)> dfs_size = [&](int u, int p) {
      for (int v : g[u]) {
        if (v != p) {
          dfs_size(v, u);
          size[u] += size[v];
        }
      }
    };
    function<void(int, int, int)> dfs_dcmp = [&](int u, int p, int h) {
      par[u] = p;
      head[u] = h;
      vid[u] = k++;
      inv[vid[u]] = u;
      for (int v : g[u]) {
        if (v != p && size[u] < size[v] * 2) {
          dfs_dcmp(v, u, h);
        }
      }
      for (int v : g[u]) {
        if (v != p && size[u] >= size[v] * 2) {
          dfs_dcmp(v, u, v);
        }
      }
    };
    dfs_size(0, -1);
    dfs_dcmp(0, -1, 0);
    for (int i = 0; i < g.size(); ++i) {
      ++len[head[i]];
    }
  }

  template<typename T>
  void foreach(int u, int v, T f) {
    while (true) {
      if (vid[u] > vid[v]) {
        if (head[u] == head[v]) {
          f(vid[v] + 1, vid[u], 0);
          break;
        } else {
          f(vid[head[u]], vid[u], 1);
          u = par[head[u]];
        }
      } else {
        if (head[u] == head[v]) {
          f(vid[u] + 1, vid[v], 0);
          break;
        } else {
          f(vid[head[v]], vid[v], 0);
          v = par[head[v]];
        }
      }
    }
  }
};
```

3.3 KD Tree

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

struct KDNode {
  vector<int> v;
  KDNode *lc, *rc;
  KDNode(const vector<int> &_v) : v(_v), lc(nullptr), rc(nullptr) {}
  static KDNode *buildKDTree(vector<vector<int>> &pnts, int lb, int rb, int dpt) {
    if (rb - lb < 1) return nullptr;
    int axis = dpt % pnts[0].size();
    int mb = lb + rb >> 1;
    nth_element(pnts.begin() + lb, pnts.begin() + mb, pnts.begin() + rb, [&](const vector<int> &a, const vector<int> &b) {
      return a[axis] < b[axis];
    });
    KDNode *t = new KDNode(pnts[mb]);
    t->lc = buildKDTree(pnts, lb, mb, dpt + 1);
    t->rc = buildKDTree(pnts, mb + 1, rb, dpt + 1);
    return t;
  }
  static void release(KDNode *t) {
    if (t->lc) release(t->lc);
    if (t->rc) release(t->rc);
    delete t;
  }
  static void searchNearestNode(KDNode *t, KDNode *q, KDNode *&c, int dpt) {
    int axis = dpt % t->v.size();
    if (t->v != q->v && (c == nullptr || dis(q, t) < dis(q, c))) c = t;
    if (t->lc && (!t->rc || q->v[axis] < t->v[axis])) {
      searchNearestNode(t->lc, q, c, dpt + 1);
      if (t->rc && (c == nullptr || 1LL * (t->v[axis] - q->v[axis]) * (t->v[axis] - q->v[axis]) < dis(q, c))) {
        searchNearestNode(t->rc, q, c, dpt + 1);
      }
    } else if (t->rc) {
      searchNearestNode(t->rc, q, c, dpt + 1);
      if (t->lc && (c == nullptr || 1LL * (t->v[axis] - q->v[axis]) * (t->v[axis] - q->v[axis]) < dis(q, c))) {
        searchNearestNode(t->lc, q, c, dpt + 1);
      }
    }
  }
  static int64_t dis(KDNode *a, KDNode *b) {
    int64_t r = 0;
    for (int i = 0; i < a->v.size(); ++i) {
      r += 1LL * (a->v[i] - b->v[i]) * (a->v[i] - b->v[i]);
    }
    return r;
  }
};

signed main() {
  ios::sync_with_stdio(false);
  int T;
  cin >> T;
  for (int ti = 0; ti < T; ++ti) {
    int N;
    cin >> N;
    vector<vector<int>> pnts(N, vector<int>(2));
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < 2; ++j) {
        cin >> pnts[i][j];
      }
    }
    vector<vector<int>> _pnts = pnts;
    KDNode *root = KDNode::buildKDTree(_pnts, 0, pnts.size(), 0);
    for (int i = 0; i < N; ++i) {
      KDNode *q = new KDNode(pnts[i]);
      KDNode *c = nullptr;
      KDNode::searchNearestNode(root, q, c, 0);
      cout << KDNode::dis(c, q) << endl;
      delete q;
    }
    KDNode::release(root);
  }
  return 0;
```

```
}
```

3.4 PST

```
constexpr int PST_MAX_NODES = 1 << 22; // recommended: prepare at least 4nlgn, n to power of 2
struct Pst {
  int maxv;
  Pst *lc, *rc;
  Pst() : lc(nullptr), rc(nullptr), maxv(0) {}
  Pst(const Pst *rhs) : lc(rhs->lc), rc(rhs->rc), maxv(rhs->maxv) {}
  static Pst *build(int lb, int rb) {
    Pst *t = new(mem_ptr++) Pst;
    if (rb - lb == 1) return t;
    t->lc = build(lb, lb + rb >> 1);
    t->rc = build(lb + rb >> 1, rb);
    return t;
  }
  static int query(Pst *t, int lb, int rb, int ql, int qr) {
    if (qr <= lb || rb <= ql) return 0;
    if (ql <= lb && rb <= qr) return t->maxv;
    int mb = lb + rb >> 1;
    return max(query(t->lc, lb, mb, ql, qr), query(t->rc, mb, rb, ql, qr));
  }
  static Pst *modify(Pst *t, int lb, int rb, int k, int v) {
    Pst *n = new(mem_ptr++) Pst(t);
    if (rb - lb == 1) return n->maxv = v, n;
    int mb = lb + rb >> 1;
    if (k < mb) n->lc = modify(t->lc, lb, mb, k, v);
    else n->rc = modify(t->rc, mb, rb, k, v);
    n->maxv = max(n->lc->maxv, n->rc->maxv);
    return n;
  }
  static Pst mem_pool[PST_MAX_NODES];
  static Pst *mem_ptr;
  static void clear() {
    while (mem_ptr != mem_pool) (--mem_ptr)->~Pst();
  }
} Pst::mem_pool[PST_MAX_NODES], *Pst::mem_ptr = Pst::mem_pool;
/*
Usage:

vector<Pst *> version(N + 1);
version[0] = Pst::build(0, C); // [0, C)
for (int i = 0; i < N; ++i) version[i + 1] = modify(version[i], ...);
Pst::query(...);
Pst::clear();

*/
```

3.5 Rbst

```
constexpr int RBST_MAX_NODES = 1 << 20;
struct Rbst {
  int size, val;
  // int minv;
  // int add_tag, rev_tag;
  Rbst *lc, *rc;
  Rbst(int v = 0) : size(1), val(v), lc(nullptr), rc(nullptr) {
    // minv = v;
    // add_tag = 0;
    // rev_tag = 0;
  }
  void push() {
    /*
    if (add_tag) { // unprocessed subtree has tag on root
      val += add_tag;
      minv += add_tag;
      if (lc) lc->add_tag += add_tag;
      if (rc) rc->add_tag += add_tag;
      add_tag = 0;
    }
    if (rev_tag) {
      swap(lc, rc);
      if (lc) lc->rev_tag ^= 1;
      if (rc) rc->rev_tag ^= 1;
      rev_tag = 0;
    }
    */
  }
  void pull() {
    size = 1;
    // minv = val;
    if (lc) {
      lc->push();
      size += lc->size;
      // minv = min(minv, lc->minv);
    }
    if (rc) {
      rc->push();
      size += rc->size;
      // minv = min(minv, rc->minv);
    }
  }
  static int get_size(Rbst *t) { return t ? t->size : 0; }
  static void split(Rbst *t, int k, Rbst *&a, Rbst *&b) {
    if (!t) return void(a = b = nullptr);
    t->push();
    if (get_size(t->lc) >= k) {
      b = t;
      split(t->lc, k, a, b->lc);
      b->pull();
    } else {
      a = t;
      split(t->rc, k - get_size(t->lc) - 1, a->rc, b);
      a->pull();
    }
  } // splits t, left k elements to a, others to b, maintaining order
  static Rbst *merge(Rbst *a, Rbst *b) {
    if (!a || !b) return a ? a : b;
    if (rand() % (a->size + b->size) < a->size) {
      a->push();
      a->rc = merge(a->rc, b);
      a->pull();
      return a;
    } else {
      b->push();
      b->lc = merge(a, b->lc);
      b->pull();
      return b;
    }
  } // merges a and b, maintaing order
  static int lower_bound(Rbst *t, const int &key) {
    if (!t) return 0;
    if (t->val >= key) return lower_bound(t->lc, key);
    return get_size(t->lc) + 1 + lower_bound(t->rc, key);
  }
  static void insert(Rbst *&t, const int &key) {
    int idx = lower_bound(t, key);
    Rbst *tt;
    split(t, idx, tt, t);
    t = merge(merge(tt, new(mem_ptr++) Rbst(key)), t);
  }

  static Rbst mem_pool[RBST_MAX_NODES]; // CAUTION!!
  static Rbst *mem_ptr;
  static void clear() {
    while (mem_ptr != mem_pool) (--mem_ptr)->~Rbst();
  }
} Rbst::mem_pool[RBST_MAX_NODES], *Rbst::mem_ptr = Rbst::mem_pool;

/*
Usage:

Rbst *t = new(Rbst::mem_ptr++) Rbst(val);
t = Rbst::merge(t, new(Rbst::mem_ptr++) Rbst(another_val));
```

```
Rbst *a, *b;
Rbst::split(t, 2, a, b); // a will have first 2 elements, b will have the rest, in order
Rbst::clear(); // wipes out all memory; if you know the mechanism of clear() you can maintain many trees

*/
```

3.6 Link Cut Tree

```
const int MEM = 1<<18;
struct Node {
  static Node mem[MEM], *pmem;
  Node *ch[2], *f;
  int id, size, revTag = 0, val = 0, sum = 0;
  void reverse() { swap(ch[0], ch[1]), revTag ^= 1; }
  void push() {
    if (revTag) {
      for (int i : {0, 1}) if (ch[i]) ch[i]->reverse();
      revTag = 0;
    }
  }
  void pull() {
    size = (ch[0] ? ch[0]->size : 0) + (ch[1] ? ch[1]->size : 0) + 1;
    sum = val;
    for (int i : {0, 1}) if (ch[i]) ch[i]->f = this, sum ^= ch[i]->sum;
  }
  int dir() { return f->ch[1] == this; }
  Node () : id(-1), size(0) { f = ch[0] = ch[1] = nullptr; }
  Node (int id, int _val = 0) : id(id), size(1) {
    val = sum = _val;
    f = ch[0] = ch[1] = nullptr;
  }
  bool isRoot() {
    return f == nullptr or f->ch[dir()] != this;
  } // is root of current splay
  void rotate() {
    Node* u = f;
    f = u->f;
    if (not u->isRoot()) u->f->ch[u->dir()] = this;
    int d = this == u->ch[0];
    u->ch[!d] = ch[d], ch[d] = u;
    u->pull(), pull();
  }
  void splay() {
    auto v = this;
    if (v == nullptr) return;
    {
      vector<Node*> st;
      Node* u = v;
      st.push_back(u);
      while (not u->isRoot()) st.push_back(u = u->f);
      while (st.size()) st.back()->push(), st.pop_back();
    }
    while (not v->isRoot()) {
      Node* u = v->f;
      if (not u->isRoot()) {
        (((u->ch[0] == v) xor (u->f->ch[0] == u)) ? v : u)->rotate();
      }
      v->rotate();
    } v->pull();
  }
  // Splay feature above
  void access() {
    for (Node *u = nullptr, *v = this; v != nullptr; u = v, v = v->f)
      v->splay(), v->ch[1] = u, v->pull();
  }
  Node* findroot() {
    access(), splay();
    auto v = this;
    while (v->ch[0] != nullptr) v = v->ch[0];
    v->splay(); // for complexity assertion
    return v;
  }
  void makeroot() { access(), splay(), reverse(); }
  static void split(Node* x, Node* y) { x->makeroot(), y->access(), y->splay(); }
  static bool link(Node* x, Node* p) {
    x->makeroot();
    if (p->findroot() != x) return x->f = p, true;
    else return false;
  }
  static void cut(Node* x) {
    x->access(), x->splay(), x->push(), x->ch[0] = x->ch[0]->f = nullptr;
  }
  static bool cut(Node* x, Node* p) { // make sure that p is above x
    auto rt = x->findroot();
    x->makeroot();
    bool test = false;
    if (p->findroot() == x and p->f == x and not p->ch[0]) {
      p->f = x->ch[1] = nullptr, x->pull();
      test = true;
    }
    rt->makeroot();
    return test;
  }
  static int path(Node* x, Node* y) { // sum of value on path x-y
    auto tmp = x->findroot();
    split(x, y);
    int ret = y->sum;
    tmp->makeroot();
    return ret;
  }
  static Node* lca(Node* x, Node* y) {
    x->access(), y->access();
    y->splay();
    if (x->f == nullptr) return x;
    else return x->f;
  }
} Node::mem[MEM], *Node::pmem = Node::mem;

Node* vt[MEM];
```

3.7 mos

```
template<typename D, D zero, typename Q, typename M>
vector<D> mos(const vector<D> &dat, vector<Q> q, M sum, function<void(M&, D, int)> fadd) {
  int bs = sqrt(q.size()) + 1;
  vector<D> ans(q.size(), zero);
  vector<int> qord(q.size());
  iota(qord.begin(), qord.end(), 0);
  sort(qord.begin(), qord.end(), [&](int i, int j) {
    if (get<0>(q[i]) / bs != get<0>(q[j]) / bs) return get<0>(q[i]) < get<0>(q[j]);
    return get<1>(q[i]) < get<1>(q[j]);
  });
  for (int qi = 0, lb = 0, rb = 0; qi < q.size(); ++qi) { // [lb, rb)
    int i = qord[qi];
    while (get<0>(q[i]) < lb) fadd(sum, dat[--lb], 1);
    while (get<1>(q[i]) < rb) fadd(sum, dat[--rb], -1);
    while (lb < get<0>(q[i])) fadd(sum, dat[lb++], -1);
    while (rb < get<1>(q[i])) fadd(sum, dat[rb++], 1);
    ans[i] = get<0>(sum);
  }
  return ans;
}

/* example
using maintain_type = tuple<int64_t, array<int, 1 << 17>>;
auto mt_add = [&](maintain_type &s, int d, int sign) {
  int w = 0;
  for (int i = 0; i < 17; ++i) w += get<1>(s)[d ^ 1 << i];
```

```
  get<0>(s) += sign * w;
  get<1>(s)[d] += sign;
};
maintain_type mt_zero = make_tuple(0, array<int, 1 << 17>());
vector<int> res = mos<int, 0, tuple<int, int>, maintain_type>(dat, query, mt_zero, mt_add);
*/
```

3.8 pbds

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

// Example 1:
// key type, mapped policy, key comparison functor, data structure, order functions
typedef tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update> rbtree;
  rbtree tree;
  tree.insert(5);
  tree.insert(6);
  tree.insert(-100);
  tree.insert(5);
  assert(*tree.find_by_order(0) == -100);
  assert(tree.find_by_order(4) == tree.end());
  assert(tree.order_of_key(4) == 1); // lower_bound
  tree.erase(6);

  rbtree x;
  x.insert(9);
  x.insert(10);
  tree.join(x);
  assert(x.size() == 0);
  assert(tree.size() == 4);

  tree.split(9, x);
  assert(*x.begin() == 10);
  assert(*tree.begin() == -100);

// Example 2:
template <class Node_CItr, class Node_Itr, class Cmp_Fn, class _Alloc>
struct my_node_update {
  typedef int metadata_type; // maintain size with int

  int order_of_key(pair<int, int> x) {
    int ans = 0;
    auto it = node_begin();
    while (it != node_end()) {
      auto l = it.get_l_child();
      auto r = it.get_r_child();
      if (Cmp_Fn()(x, **it)) { // x < it->size
        it = l;
      } else {
        if (x == **it) return ans; // x == it->size
        ++ans;
        if (l != node_end()) ans += l.get_metadata();
        it = r;
      }
    }
    return ans;
  }
  // update policy
  void operator()(Node_Itr it, Node_CItr end_it) {
    auto l = it.get_l_child();
    auto r = it.get_r_child();
    int left = 0, right = 0;
    if (l != end_it) left = l.get_metadata();
    if (r != end_it) right = r.get_metadata();
    const_cast<int &>(it.get_metadata()) = left + right + 1;
  }

  virtual Node_CItr node_begin() const = 0;
  virtual Node_CItr node_end() const = 0;
};

typedef tree<pair<int, int>, null_type, less<pair<int, int>>, rb_tree_tag, my_node_update> rbtree;
  rbtree g;
  g.insert({3, 4});
  assert(g.order_of_key({3, 4}) == 0);
```

4 Flow

4.1 CostFlow

```
template <class TF, class TC>
struct CostFlow {
  static const int MAXV = 205;
  static const TC INF = 0x3f3f3f3f;
  struct Edge {
    int v, r;
    TF f;
    TC c;
    Edge(int _v, int _r, TF _f, TC _c) : v(_v), r(_r), f(_f), c(_c) {}
  };
  int n, s, t, pre[MAXV], pre_E[MAXV], inq[MAXV];
  TF fl;
  TC dis[MAXV], cost;
  vector<Edge> E[MAXV];
  CostFlow(int _n, int _s, int _t) : n(_n), s(_s), t(_t), fl(0), cost(0) {}
  void add_edge(int u, int v, TF f, TC c) {
    E[u].emplace_back(v, E[v].size(), f, c);
    E[v].emplace_back(u, E[u].size() - 1, 0, -c);
  }
  pair<TF, TC> flow() {
    while (true) {
      for (int i = 0; i < n; ++i) {
        dis[i] = INF;
        inq[i] = 0;
      }
      dis[s] = 0;
      queue<int> que;
      que.emplace(s);
      while (not que.empty()) {
        int u = que.front();
        que.pop();
        inq[u] = 0;
        for (int i = 0; i < E[u].size(); ++i) {
          int v = E[u][i].v;
          TC w = E[u][i].c;
          if (E[u][i].f > 0 and dis[v] > dis[u] + w) {
            pre[v] = u;
            pre_E[v] = i;
            dis[v] = dis[u] + w;
            if (not inq[v]) {
              inq[v] = 1;
              que.emplace(v);
            }
          }
        }
      }
      if (dis[t] == INF) break;
      TF tf = INF;
      for (int v = t, u, l; v != s; v = u) {
        u = pre[v];
        l = pre_E[v];
        tf = min(tf, E[u][l].f);
      }
      for (int v = t, u, l; v != s; v = u) {
        u = pre[v];
        l = pre_E[v];
        E[u][l].f -= tf;
        E[v][E[u][l].r].f += tf;
      }
      cost += tf * dis[t];
      fl += tf;
    }
    return {fl, cost};
  }
};
```

4.2 Dinic

```
template <class T>
struct Dinic {
  static const int MAXV = 10000;
  static const T INF = 0x3f3f3f3f;
  struct Edge {
    int v;
    T f;
    int re;
    Edge(int _v, T _f, int _re) : v(_v), f(_f), re(_re) {}
  };
  int n, s, t, level[MAXV];
  vector<Edge> E[MAXV];
  int now[MAXV];
  Dinic(int _n, int _s, int _t) : n(_n), s(_s), t(_t) {}
  void add_edge(int u, int v, T f, bool bidirectional = false) {
    E[u].emplace_back(v, f, E[v].size());
    E[v].emplace_back(u, 0, E[u].size() - 1);
    if (bidirectional) {
      E[v].emplace_back(u, f, E[u].size() - 1);
    }
  }
  bool BFS() {
    memset(level, -1, sizeof(level));
    queue<int> que;
    que.emplace(s);
    level[s] = 0;
    while (not que.empty()) {
      int u = que.front();
      que.pop();
      for (auto it : E[u]) {
        if (it.f > 0 and level[it.v] == -1) {
          level[it.v] = level[u] + 1;
          que.emplace(it.v);
        }
      }
    }
    return level[t] != -1;
  }
  T DFS(int u, T nf) {
    if (u == t) return nf;
    T res = 0;
    while (now[u] < E[u].size()) {
      Edge &it = E[u][now[u]];
      if (it.f > 0 and level[it.v] == level[u] + 1) {
        T tf = DFS(it.v, min(nf, it.f));
        res += tf;
        nf -= tf;
        it.f -= tf;
        E[it.v][it.re].f += tf;
        if (nf == 0) return res;
      } else
        ++now[u];
    }
    if (not res) level[u] = -1;
    return res;
  }
  T flow(T res = 0) {
    while (BFS()) {
      T temp;
      memset(now, 0, sizeof(now));
      while (temp = DFS(s, INF)) {
        res += temp;
        res = min(res, INF);
      }
    }
    return res;
  }
};
```

4.3 KM matching

```
template<typename T>
struct Hungarian { // minimum weight matching
  public:
  int n, m;
  vector< vector<T> > a;
  vector<T> u, v;
  vector<int> pa, pb, way;
  vector<T> minv;
  vector<bool> used;
  T inf;

  Hungarian(int _n, int _m) : n(_n), m(_m) {
    assert(n <= m);
    a = vector< vector<T> >(n, vector<T>(m));
    v = u = vector<T>(n + 1);
    pb = pa = vector<int>(n + 1, -1);
    way = vector<int>(m, -1);
    minv = vector<T>(m);
    used = vector<bool>(m + 1);
    inf = numeric_limits<T>::max();
  }

  inline void add_row(int i) {
    fill(minv.begin(), minv.end(), inf);
    fill(used.begin(), used.end(), false);
    pb[m] = i, pa[i] = m;
    int j0 = m;
    do {
      used[j0] = true;
      int i0 = pb[j0], j1 = -1;
      T delta = inf;
      for (int j = 0; j < m; j++) {
        if (!used[j]) {
          T cur = a[i0][j] - 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 <= m; j++) {
        if (used[j]) {
          u[pb[j]] += delta, v[j] -= delta;
        } else {
          minv[j] -= delta;
        }
      }
      j0 = j1;
    } while (pb[j0] != -1);
    do {
      int j1 = way[j0];
      pb[j0] = pb[j1], pa[pb[j0]] = j0, j0 = j1;
    } while (j0 != m);
  }

  inline T current_score() {
    return -v[m];
  }

  inline T solve() {
    for (int i = 0; i < n; i++) {
      add_row(i);
    }
    return current_score();
  }
};
```

4.4 Matching

```
class matching {
  public:
  vector< vector<int> > g;
  vector<int> pa, pb, was;
  int n, m, res, iter;

  matching(int _n, int _m) : n(_n), m(_m) {
    assert(0 <= n && 0 <= m);
    pa = vector<int>(n, -1);
    pb = vector<int>(m, -1);
    was = vector<int>(n, 0);
    g.resize(n);
```

```
    res = 0, iter = 0;
  }

  void add_edge(int from, int to) {
    assert(0 <= from && from < n && 0 <= to && to < m)
        ;
    g[from].push_back(to);
  }

  bool dfs(int v) {
    was[v] = iter;
    for (int u : g[v])
      if (pb[u] == -1)
        return pa[v] = u, pb[u] = v, true;
    for (int u : g[v])
      if (was[pb[u]] != iter && dfs(pb[u]))
        return pa[v] = u, pb[u] = v, true;
    return false;
  }

  int solve() {
    while (true) {
      iter++;
      int add = 0;
      for (int i = 0; i < n; i++)
        if (pa[i] == -1 && dfs(i))
          add++;
      if (add == 0) break;
      res += add;
    }
    return res;
  }

  int run_one(int v) {
    if (pa[v] != -1) return 0;
    iter++;
    return (int) dfs(v);
  }
  pair<vector<bool>, vector<bool>> vertex_cover() {
    solve();
    vector<bool> a_cover(n, true), b_cover(m, false);
    function<void(int)> dfs_aug = [&](int v) {
      a_cover[v] = false;
      for (int u: g[v])
        if (not b_cover[u])
          b_cover[u] = true, dfs_aug(pb[u]);
    };
    for (int v = 0; v < n; ++v)
      if (a_cover[v] and pa[v] == -1)
        dfs_aug(v);
    return {a_cover, b_cover};
  }
};
```

5 Geometry

5.1 Convex Envelope

```
using F = long long;
struct Line {
  static const F QUERY = numeric_limits<F>::max();
  F m, b;
  Line(F m, F b) : m(m), b(b) {}
  mutable function<const Line*()> succ;
  bool operator<(const Line& rhs) const {
    if (rhs.b != QUERY) return m == rhs.m ? b < rhs.b
        : m < rhs.m;
    const Line* s = succ();
    return s and b - s->b < (s->m - m) * rhs.m;
  }
  F operator()(F x) const { return m * x + b; };
};

struct HullDynamic : public multiset<Line> {
  bool isOnHull(iterator y) { //Mathematically,
      Strictly
    auto z = next(y);
    if (y == begin()) return z == end() or y->m != z->
        m or z->b < y->b;
    auto x = prev(y);
    if (z == end()) return x->m != y->m or x->b < y->b
        ;
    if (y->m == z->m) return y->b > z->b;
    if (x->m == y->m) return x->b < y->b;
    return (x->b - y->b) * (z->m - y->m) < (y->b - z->
        b) * (y->m - x->m);
    // Beware long long overflow
  }
  void insertLine(F m, F b) {
    auto y = insert(Line(m, b));
    y->succ = [=] { return next(y) == end() ? nullptr
        : &*next(y); };
    if (not isOnHull(y)) { erase(y); return; }
    while (next(y) != end() and not isOnHull(next(y)))
        erase(next(y));
    while (y != begin() and not isOnHull(prev(y)))
        erase(prev(y));
  }
  F operator()(F x) { return (*lower_bound(Line{x,
      Line::QUERY}))(x); }
};
```

5.2 3D ConvexHull

```
#define SIZE(X) (int(X.size()))
#define PI 3.14159265358979323846264338327950288
struct Pt{
  Pt cross(const Pt &p) const
  { return Pt(y * p.z - z * p.y, z * p.x - x * p.z, x
      * p.y - y * p.x); }
} info[N];
int mark[N][N],n, cnt;;
double mix(const Pt &a, const Pt &b, const Pt &c)
{ return a * (b ^ c); }
double area(int a, int b, int c)
{ return norm((info[b] - info[a]) ^ (info[c] - info[a
    ])); }
double volume(int a, int b, int c, int d)
{ return mix(info[b] - info[a], info[c] - info[a],
    info[d] - info[a]); }
struct Face{
  int a, b, c; Face(){}
  Face(int a, int b, int c): a(a), b(b), c(c) {}
  int &operator [](int k)
  { if (k == 0) return a; if (k == 1) return b; return
      c; }
};
vector<Face> face;
void insert(int a, int b, int c)
{ face.push_back(Face(a, b, c)); }
void add(int v) {
  vector <Face> tmp; int a, b, c; cnt++;
  for (int i = 0; i < SIZE(face); i++) {
    a = face[i][0]; b = face[i][1]; c = face[i][2];
    if(Sign(volume(v, a, b, c)) < 0)
    mark[a][b] = mark[b][a] = mark[b][c] = mark[c][b]
        = mark[c][a] = mark[a][c] = cnt;
    else tmp.push_back(face[i]);
  } face = tmp;
  for (int i = 0; i < SIZE(tmp); i++) {
    a = face[i][0]; b = face[i][1]; c = face[i][2];
    if (mark[a][b] == cnt) insert(b, a, v);
    if (mark[b][c] == cnt) insert(c, b, v);
    if (mark[c][a] == cnt) insert(a, c, v);
}}
int Find(){
  for (int i = 2; i < n; i++) {
    Pt ndir = (info[0] - info[i]) ^ (info[1] - info[i
        ]);
    if (ndir == Pt()) continue; swap(info[i], info[2])
        ;
    for (int j = i + 1; j < n; j++) if (Sign(volume(0,
        1, 2, j)) != 0) {
      swap(info[j], info[3]); insert(0, 1, 2); insert
          (0, 2, 1); return 1;
} } return 0; }
int main() {
  for (; scanf("%d", &n) == 1; ) {
    for (int i = 0; i < n; i++) info[i].Input();
```

```
    sort(info, info + n); n = unique(info, info + n) - info;
    face.clear(); random_shuffle(info, info + n);
    if (Find()) { memset(mark, 0, sizeof(mark)); cnt = 0;
      for (int i = 3; i < n; i++) add(i); vector<Pt> Ndir;
      for (int i = 0; i < SIZE(face); ++i) {
        Pt p = (info[face[i][0]] - info[face[i][1]]) ^
               (info[face[i][2]] - info[face[i][1]]);
        p = p / norm( p ); Ndir.push_back(p);
      } sort(Ndir.begin(), Ndir.end());
      int ans = unique(Ndir.begin(), Ndir.end()) - Ndir.begin();
      printf("%d\n", ans);
    } else printf("1\n");
} }
double calcDist(const Pt &p, int a, int b, int c)
{ return fabs(mix(info[a] - p, info[b] - p, info[c] - p) / area(a, b, c)); }
//compute the minimal distance of center of any faces
double findDist() { //compute center of mass
  double totalWeight = 0; Pt center(.0, .0, .0);
  Pt first = info[face[0][0]];
  for (int i = 0; i < SIZE(face); ++i) {
    Pt p = (info[face[i][0]]+info[face[i][1]]+info[face[i][2]]+first)*.25;
    double weight = mix(info[face[i][0]] - first, info[face[i][1]]
        - first, info[face[i][2]] - first);
    totalWeight += weight; center = center + p * weight;
  } center = center / totalWeight;
  double res = 1e100; //compute distance
  for (int i = 0; i < SIZE(face); ++i)
    res = min(res, calcDist(center, face[i][0], face[i][1], face[i][2]));
    return res; }
```

5.3 Half plane intersection

```
template<typename T, typename Real = double>
Poly<Real> halfplane_intersection(vector<Line<T, Real>> s) {
  sort(s.begin(), s.end());
  const Real eps = 1e-10;
  int n = 1;
  for (int i = 1; i < s.size(); ++i) {
    if ((s[i].vec()&s[n - 1].vec()) < eps or abs(s[i].vec()^s[n - 1].vec()) > eps)
      s[n++] = s[i];
  }
  s.resize(n);
  assert(n >= 3);
  deque<Line<T, Real>> q;
  deque<Pt<Real>> p;
  q.push_back(s[0]);
  q.push_back(s[1]);
  p.push_back(s[0].get_intersection(s[1]));
  for (int i = 2; i < n; ++i) {
    while (q.size() > 1 and s[i].ori(p.back()) < -eps)
      p.pop_back(), q.pop_back();
    while (q.size() > 1 and s[i].ori(p.front()) < -eps)
      p.pop_front(), q.pop_front();
    p.push_back(q.back().get_intersection(s[i]));
    q.push_back(s[i]);
  }
  while (q.size() > 1 and q.front().ori(p.back()) < -eps)
    q.pop_back(), p.pop_back();
  while (q.size() > 1 and q.back().ori(p.front()) < -eps)
    q.pop_front(), p.pop_front();
  p.push_back(q.front().get_intersection(q.back()));
  return Poly<Real>(vector<Pt<Real>>(p.begin(), p.end()));
}
```

5.4 Lines

```
template <typename T, typename Real = double>
struct Line {
  Pt<T> st, ed;
  Pt<T> vec() const { return ed - st; }
  T ori(const Pt<T> p) const { return (ed - st)^(p - st); }
  Line(const Pt<T> x, const Pt<T> y) : st(x), ed(y) {}
  template<class F> operator Line<F> () const {
    return Line<F>((Pt<F>)st, (Pt<F>)ed);
  }

  // sort by arg, the left is smaller for parallel lines
  bool operator<(Line B) const {
    Pt<T> a = vec(), b = B.vec();
    auto sgn = [](const Pt<T> t) { return (t.y == 0? t.x: t.y) < 0; };
    if (sgn(a) != sgn(b)) return sgn(a) < sgn(b);
    if (abs(a^b) == 0) return B.ori(st) > 0;
    return (a^b) > 0;
  }

  // Regard a line as a function
  template<typename F> Pt<F> operator()(const F x) const {
    return Pt<F>(st) + vec() * x;
  }

  bool isSegProperIntersection(const Line l) const {
    return l.ori(st) * l.ori(ed) < 0 and ori(l.st) * ori(l.ed) < 0;
  }

  bool isPtOnSegProperly(const Pt<T> p) const {
    return ori(p) == 0 and ((st - p)&(ed - p)) < 0;
  }

  Pt<Real> getIntersection(const Line<Real> l) {
    Line<Real> h = *this;
    return l(((l.st - h.st)^h.vec()) / (h.vec()^l.vec()));
  }

  Pt<Real> projection(const Pt<T> p) const {
    return operator()(((p - st)&vec()) / (Real)(vec().norm()));
  }
};
```

5.5 Points

```
template <typename T>
struct Pt {
  T x, y;
  Pt() : x(0), y(0) {}
  Pt(const T x, const T y) : x(x), y(y) {}
  template <class F> explicit operator Pt<F> () const {
    return Pt<F>((F)x, (F)y); }

  Pt operator+(const Pt b) const { return Pt(x + b.x, y + b.y); }
  Pt operator-(const Pt b) const { return Pt(x - b.x, y - b.y); }
  template <class F> Pt<F> operator* (const F fac) {
    return Pt<F>(x * fac, y * fac); }
  template <class F> Pt<F> operator/ (const F fac) {
    return Pt<F>(x / fac, y / fac); }

  T operator&(const Pt b) const { return x * b.x + y * b.y; }
  T operator^(const Pt b) const { return x * b.y - y * b.x; }

  bool operator==(const Pt b) const {
    return x == b.x and  y == b.y; }
  bool operator<(const Pt b) const {
```

```
    return x == b.x? y < b.y: x < b.x; }

  Pt operator-() const { return Pt(-x, -y); }
  T norm() const { return *this & *this; }
  Pt prep() const { return Pt(-y, x); }
};
template<class F> istream& operator>>(istream& is, Pt<F> &pt) {
  return is >> pt.x >> pt.y;
}
template<class F> ostream& operator<<(ostream& os, Pt<F> &pt) {
  return os << pt.x << ' ' << pt.y;
}
```

5.6 Polys

```
template <class F> using Polygon = vector<Pt<F>>;

template<typename T>
T twiceArea(Polygon<T> Ps) {
  int n = Ps.size();
  T ans = 0;
  for (int i = 0; i < n; ++i)
    ans += Ps[i] ^ Ps[i + 1 == n ? 0 : i + 1];
  return ans;
}

template <class F>
Polygon<F> getConvexHull(Polygon<F> points) {
  sort(begin(points), end(points));
  Polygon<F> hull;
  hull.reserve(points.size() + 1);
  for (int phase = 0; phase < 2; ++phase) {
    auto start = hull.size();
    for (auto& point : points) {
      while (hull.size() >= start + 2 and
             Line<F>(hull.back(), hull[hull.size() - 2]).ori(point) <= 0)
        hull.pop_back();
      hull.push_back(point);
    }
    hull.pop_back();
    reverse(begin(points), end(points));
  }
  if (hull.size() == 2 and hull[0] == hull[1]) hull.pop_back();
  return hull;
}
```

5.7 Rotating Axis

```
class Rotating_axis{
  struct POINT{
    Pt<LL> p;
    int i;
  };
  struct LINE{
    Line<LL> L;
    int i, j;
    bool operator<(const LINE B) const { return (L.vec()^B.L.vec()) > 0; }
  };
  vector<POINT> Ps;
  vector<LINE> Ls;
  vector<int> idx_at;
  int n, lid = 0;
public:
  Rotating_axis(vector<Pt<LL>> V) {
    n = V.size();
    Ps.resize(n), idx_at.resize(n);
    for (int i = 0; i < n; ++i) Ps[i] = {V[i], i};
    for (int i = 0; i < n; ++i) for (int j = 0; j < i; ++j) {
      auto a = V[i], b = V[j], v = b - a;
      int ii = i, jj = j;
      if (v.y > 0 or (v.y == 0 and v.x > 0)) swap(a, b), swap(ii, jj);
      Ls.push_back({Line<LL>(a, b), ii, jj});
    }
    sort(Ls.begin(), Ls.end());
    sort(Ps.begin(), Ps.end(), [&](POINT A, POINT B) {
      auto a = A.p, b = B.p;
      LL det1 = Ls[0].L.ori(a), det2 = Ls[0].L.ori(b);
      return det1 == det2? ((a - b) & Ls[0].L.vec()) > 0 : det1 > det2;
    });
    for (int i = 0; i < n; ++i) idx_at[Ps[i].i] = i;
  }
  bool next_axis() {
    if (lid == Ls.size()) return false;
    int i = Ls[lid].i, j = Ls[lid].j, wi = idx_at[i], wj = idx_at[j];
    swap(Ps[wi], Ps[wj]);
    swap(idx_at[i], idx_at[j]);
    assert(idx_at[i] == idx_at[j] - 1);
    return ++lid, true;
  }
  Pt<LL> at(size_t i) { return Ps[i].p; }
  Line<LL> cur_axis() { return Ls[lid].L; }
};
```

6 Graph

6.1 2-SAT

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

using namespace std;

class two_SAT {
  public:
  vector< vector<int> > g, rg;
  vector<int> visit, was;
  vector<int> id;
  vector<int> res;
  int n, iter;

  two_SAT(int _n) : n(_n) {
    g.resize(n * 2);
    rg.resize(n * 2);
    was = vector<int>(n * 2, 0);
    id = vector<int>(n * 2, -1);
    res.resize(n);
    iter = 0;
  }

  void add_edge(int from, int to) { // add (a -> b)
    assert(from >= 0 && from < 2 * n && to >= 0 && to < 2 * n);
    g[from].emplace_back(to);
    rg[to].emplace_back(from);
  }

  void add_or(int a, int b) { // add (a V b)
    int nota = (a < n) ? a + n : a - n;
    int notb = (b < n) ? b + n : b - n;
    add_edge(nota, b);
    add_edge(notb, a);
  }

  void dfs(int v) {
    was[v] = true;
    for (int u : g[v]) {
      if (!was[u]) dfs(u);
    }
    visit.emplace_back(v);
  }

  void rdfs(int v) {
    id[v] = iter;
    for (int u : rg[v]) {
      if (id[u] == -1) rdfs(u);
    }
  }

```

```
  int scc() {
    for (int i = 0; i < 2 * n; i++) {
      if (!was[i]) dfs(i);
    }
    for (int i = 2 * n - 1; i >= 0; i--) {
      if (id[ visit[i] ] == -1) {
        rdfs(visit[i]);
        iter++;
      }
    }
    return iter;
  }

  bool solve() {
    scc();
    for (int i = 0; i < n; i++) {
      if (id[i] == id[i + n]) return false;
      res[i] = (id[i] < id[i + n]);
    }
    return true;
  }

};

/*
  usage:
    index 0 ~ n - 1 : True
    index n ~ 2n - 1 : False
    add_or(a, b) : add SAT (a or b)
    add_edge(a, b) : add SAT (a -> b)
    if you want to set x = True, you can add (not X -> X)
    solve() return True if it exist at least one solution
    res[i] store one solution
      false -> choose a
      true -> choose a + n
*/
```

6.2 BCC

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

using namespace std;

class biconnected_component {
  public:
  vector< vector<int> > g;
  vector< vector<int> > comp;
  vector<int> pre, depth;
  int n;

  biconnected_component(int _n) : n(_n) {
    depth = vector<int>(n, -1);
    g.resize(n);
  }

  void add(int u, int v) {
    assert(0 <= u && u < n && 0 <= v && v < n);
    g[u].push_back(v);
    g[v].push_back(u);
  }

  int dfs(int v, int pa, int d) {
    depth[v] = d;
    pre.push_back(v);
    for (int u : g[v]) {
      if (u == pa) continue;
      if (depth[u] == -1) {
        int child = dfs(u, v, depth[v] + 1);
        if (child >= depth[v]) {
          comp.push_back(vector<int>(1, v));
          while (pre.back() != v) {
            comp.back().push_back(pre.back());
            pre.pop_back();
          }
        }
        d = min(d, child);
      }
      else {
        d = min(d, depth[u]);
      }
    }
    return d;
  }

  vector< vector<int> > solve() {
    for (int i = 0; i < n; i++) {
      if (depth[i] == -1) {
        dfs(i, -1, 0);
      }
    }
    return comp;
  }

  vector<int> get_ap() {
    vector<int> res, count(n, 0);
    for (auto c : comp) {
      for (int v : c ) {
        count[v]++;
      }
    }
    for (int i = 0; i < n; i++) {
      if (count[i] > 1) {
        res.push_back(i);
      }
    }
    return res;
  }
};
```

6.3 General Matching

```
#define MAXN 505
struct Blossom {
  vector<int> g[MAXN];
  int pa[MAXN] = {0}, match[MAXN] = {0}, st[MAXN] = {0}, S[MAXN] = {0}, v[MAXN] = {0};
  int t, n;
  Blossom(int _n) : n(_n) {}
  void add_edge(int v, int u) { // 1-index
    g[u].push_back(v), g[v].push_back(u);
  }
  inline int lca(int x, int y) {
    ++t;
    while (v[x] != t) {
      v[x] = t;
      x = st[pa[match[x]]];
      swap(x, y);
      if (x == 0) swap(x, y);
    }
    return x;
  }
  inline void flower(int x, int y, int l, queue<int> &q) {
    while (st[x] != l) {
      pa[x] = y;
      if (S[y = match[x]] == 1) q.push(y), S[y] = 0;
      st[x] = st[y] = l, x = pa[y];
    }
  }
  inline bool bfs(int x) {
    for (int i = 1; i <= n; ++i) st[i] = i;
    memset(S + 1, -1, sizeof(int) * n);
    queue<int> q;
    q.push(x), S[x] = 0;
    while (q.size()) {
      x = q.front(), q.pop();
      for (size_t i = 0; i < g[x].size(); ++i) {
        int y = g[x][i];
        if (S[y] == -1) {
          pa[y] = x, S[y] = 1;
          if (not match[y]) {
            for (int lst; x; y = lst, x = pa[y])
              lst = match[x], match[x] = y, match[y] = x;
            return 1;
          }
          q.push(match[y]), S[match[y]] = 0;
        } else if (not S[y] and st[y] != st[x]) {
```

```
          int l = lca(y, x);
          flower(y, x, l, q), flower(x, y, l, q);
        }
      }
    }
    return 0;
  }
  inline int blossom() {
    int ans = 0;
    for (int i = 1; i <= n; ++i)
      if (not match[i] and bfs(i)) ++ans;
    return ans;
  }
};
```

6.4 Bridge

```
struct Bridge {
  vector<int> imo;
  set<pair<int, int>> bridges; // all bridges (u, v), u < v
  vector<set<int>> bcc; // bcc[i] has all vertices that belong to the i'th bcc
  vector<int> at_bcc; // node i belongs to at_bcc[i]
  int bcc_ctr;

  Bridge(const vector<vector<int>> &g) : bcc_ctr(0) {
    imo.resize(g.size());
    bcc.resize(g.size());
    at_bcc.resize(g.size());
    vector<int> vis(g.size());
    vector<int> dpt(g.size());
    function<void(int, int, int)> mark = [&](int u, int fa, int d) {
      vis[u] = 1;
      dpt[u] = d;
      for (int v : G[u]) {
        if (v == fa) continue;
        if (vis[v]) {
          if (dpt[v] > dpt[u]) {
            ++imo[v];
            --imo[u];
          }
        } else mark(v, u, d + 1);
      }
    };
    mark(0, -1, 0);
    vis.assign(g.size(), 0);
    function<int(int)> expand = [&](int u) {
      vis[u] = 1;
      int s = imo[u];
      for (int v : G[u]) {
        if (vis[v]) continue;
        int e = expand(v);
        if (e == 0) bridges.emplace(make_pair(min(u, v), max(u, v)));
        s += e;
      }
      return s;
    };
    expand(0);
    fill(at_bcc.begin(), at_bcc.end(), -1);
    for (int u = 0; u < N; ++u) {
      if (~at_bcc[u]) continue;
      queue<int> que;
      que.emplace(u);
      at_bcc[u] = bcc_ctr;
      bcc[bcc_ctr].emplace(u);
      while (que.size()) {
        int v = que.front();
        que.pop();
        for (int w : G[v]) {
          if (~at_bcc[w] || bridges.count(make_pair(min(v, w), max(v, w)))) continue;
          que.emplace(w);
          at_bcc[w] = bcc_ctr;
          bcc[bcc_ctr].emplace(w);
        }
      }
      ++bcc_ctr;
    }
  }
};
```

6.5 CentroidDecomposition

```
struct CentroidDecomp {
  vector<vector<int>> g;
  vector<int> p, M, sz;
  vector<bool> vis;
  int n;

  CentroidDecomp(vector<vector<int>> g) : g(g), n(g.size()) {
    p.resize(n);
    vis.assign(n, false);
    sz.resize(n);
    M.resize(n);
  }

  int divideAndConquer(int x) {

    vector<int> q = {x};
    p[x] = x;

    for (int i = 0; i < q.size(); ++i) {
      int u = q[i];
      sz[u] = 1;
      M[u] = 0;
      for (auto v : g[u]) if (not vis[v] and v != p[u]) {
        q.push_back(v), p[v] = u;
      }
    }

    reverse(begin(q), end(q));
    for (int u : q) if (p[u] != u) {
      sz[p[u]] += sz[u];
      M[p[u]] = max(sz[u], M[p[u]]);
    }

    for (int u : q) M[u] = max(M[u], int(q.size()) - sz[u]);

    int cent = *min_element(begin(q), end(q),
                            [&](int x, int y) { return M[x] < M[y]; });

    vis[cent] = true;
    for (int u : g[cent]) if (not vis[u])
      divideAndConquer(u);
    return cent;
  }
};
```

6.6 DirectedGraphMinCycle

```
// works in O(N M)
#define INF 100000000000000LL
#define N 5010
#define M 200010
struct edge{
  int to; LL w;
  edge(int a=0, LL b=0): to(a), w(b){}
};
struct node{
  LL d; int u, next;
  node(LL a=0, int b=0, int c=0): d(a), u(b), next(c){}
}b[M];
struct DirectedGraphMinCycle{
  vector<edge> g[N], grev[N];
  LL dp[N][N], p[N], d[N], mu;
  bool inq[N];
  int n, bn, bsz, hd[N];
  void b_insert(LL d, int u){
    int i = d/mu;
    if(i >= bn) return;
```

```
    b[++bsz] = node(d, u, hd[i]);
    hd[i] = bsz;
  }
  void init( int _n ){
    n = _n;
    for( int i = 1 ; i <= n ; i ++ )
      g[ i ].clear();
  }
  void addEdge( int ai , int bi , LL ci )
  { g[ai].push_back(edge(bi,ci)); }
  LL solve(){
    fill(dp[0], dp[0]+n+1, 0);
    for(int i=1; i<=n; i++){
      fill(dp[i]+1, dp[i]+n+1, INF);
      for(int j=1; j<=n; j++) if(dp[i-1][j] < INF){
        for(int k=0; k<(int)g[j].size(); k++)
          dp[i][g[j][k].to] =min(dp[i][g[j][k].to],
                                 dp[i-1][j]+g[j][k].w)
                                     ;
      }
    }
    mu=INF; LL bunbo=1;
    for(int i=1; i<=n; i++) if(dp[n][i] < INF){
      LL a=-INF, b=1;
      for(int j=0; j<=n-1; j++) if(dp[j][i] < INF){
        if(a*(n-j) < b*(dp[n][i]-dp[j][i])){
          a = dp[n][i]-dp[j][i];
          b = n-j;
        }
      }
      if(mu*b > bunbo*a)
        mu = a, bunbo = b;
    }
    if(mu < 0) return -1; // negative cycle
    if(mu == INF) return INF; // no cycle
    if(mu == 0) return 0;
    for(int i=1; i<=n; i++)
      for(int j=0; j<(int)g[i].size(); j++)
      g[i][j].w *= bunbo;
    memset(p, 0, sizeof(p));
    queue<int> q;
    for(int i=1; i<=n; i++){
      q.push(i);
      inq[i] = true;
    }
    while(!q.empty()){
      int i=q.front(); q.pop(); inq[i]=false;
      for(int j=0; j<(int)g[i].size(); j++){
        if(p[g[i][j].to] > p[i]+g[i][j].w-mu){
          p[g[i][j].to] = p[i]+g[i][j].w-mu;
          if(!inq[g[i][j].to]){
            q.push(g[i][j].to);
            inq[g[i][j].to] = true;
          }
        }
      }
    }
    for(int i=1; i<=n; i++) grev[i].clear();
    for(int i=1; i<=n; i++)
      for(int j=0; j<(int)g[i].size(); j++){
        g[i][j].w += p[i]-p[g[i][j].to];
        grev[g[i][j].to].push_back(edge(i, g[i][j].w))
            ;
      }
    LL mldc = n*mu;
    for(int i=1; i<=n; i++){
      bn=mldc/mu, bsz=0;
      memset(hd, 0, sizeof(hd));
      fill(d+i+1, d+n+1, INF);
      b_insert(d[i]=0, i);
      for(int j=0; j<=bn-1; j++) for(int k=hd[j]; k; k
          =b[k].next){
        int u = b[k].u;
        LL du = b[k].d;
        if(du > d[u]) continue;
        for(int l=0; l<(int)g[u].size(); l++) if(g[u][
            l].to > i){
          if(d[g[u][l].to] > du + g[u][l].w){
            d[g[u][l].to] = du + g[u][l].w;
            b_insert(d[g[u][l].to], g[u][l].to);
          }
        }
      }
      for(int j=0; j<(int)grev[i].size(); j++) if(grev
          [i][j].to > i)
        mldc=min(mldc,d[grev[i][j].to] + grev[i][j].w)
            ;
    }
    return mldc / bunbo;
  }
} graph;
```

6.7 General Weighted Matching

```
struct WeightGraph {
  static const int INF = INT_MAX;
  static const int N = 514;
  struct edge {
    int u, v, w;
    edge() {}
    edge(int ui, int vi, int wi) : u(ui), v(vi), w(wi)
        {}
  };
  int n, n_x;
  edge g[N * 2][N * 2];
  int lab[N * 2];
  int match[N * 2], slack[N * 2], st[N * 2], pa[N *
      2];
  int flo_from[N * 2][N + 1], S[N * 2], vis[N * 2];
  vector<int> flo[N * 2];
  queue<int> q;
  int e_delta(const edge& e) { return lab[e.u] + lab[e
      .v] - g[e.u][e.v].w * 2; }
  void update_slack(int u, int x) {
    if (not slack[x] or e_delta(g[u][x]) < e_delta(g[
        slack[x]][x]))
      slack[x] = u;
  }
  void set_slack(int x) {
    slack[x] = 0;
    for (int u = 1; u <= n; ++u)
      if (g[u][x].w > 0 and st[u] != x and S[st[u]] ==
          0) update_slack(u, x);
  }
  void q_push(int x) {
    if (x <= n)
      q.push(x);
    else
      for (size_t i = 0; i < flo[x].size(); i++)
        q_push(flo[x][i]);
  }
  void set_st(int x, int b) {
    st[x] = b;
    if (x > n)
      for (size_t i = 0; i < flo[x].size(); ++i)
        set_st(flo[x][i], b);
  }
  int get_pr(int b, int xr) {
    int pr = find(flo[b].begin(), flo[b].end(), xr) -
        flo[b].begin();
    if (pr % 2 == 1) {
      reverse(flo[b].begin() + 1, flo[b].end());
      return (int)flo[b].size() - pr;
    } else
      return pr;
  }
  void set_match(int u, int v) {
    match[u] = g[u][v].v;
    if (u <= n) return;
    edge e = g[u][v];
    int xr = flo_from[u][e.u], pr = get_pr(u, xr);
    for (int i = 0; i < pr; ++i) set_match(flo[u][i],
        flo[u][i ^ 1]);
    set_match(xr, v);
    rotate(flo[u].begin(), flo[u].begin() + pr, flo[u
        ].end());
  }
  void augment(int u, int v) {
    for (;;) {
      int xnv = st[match[u]];
      set_match(u, v);
      if (not xnv) return;
```

```
      set_match(xnv, st[pa[xnv]]);
      u = st[pa[xnv]], v = xnv;
    }
  }
  int get_lca(int u, int v) {
    static int t = 0;
    for (++t; u or v; swap(u, v)) {
      if (u == 0) continue;
      if (vis[u] == t) return u;
      vis[u] = t;
      u = st[match[u]];
      if (u) u = st[pa[u]];
    }
    return 0;
  }
  void add_blossom(int u, int lca, int v) {
    int b = n + 1;
    while (b <= n_x and st[b]) ++b;
    if (b > n_x) ++n_x;
    lab[b] = 0, S[b] = 0;
    match[b] = match[lca];
    flo[b].clear();
    flo[b].push_back(lca);
    for (int x = u, y; x != lca; x = st[pa[y]])
      flo[b].push_back(x), flo[b].push_back(y = st[
          match[x]]), q_push(y);
    reverse(flo[b].begin() + 1, flo[b].end());
    for (int x = v, y; x != lca; x = st[pa[y]])
      flo[b].push_back(x), flo[b].push_back(y = st[
          match[x]]), q_push(y);
    set_st(b, b);
    for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b
        ].w = 0;
    for (int x = 1; x <= n; ++x) flo_from[b][x] = 0;
    for (size_t i = 0; i < flo[b].size(); ++i) {
      int xs = flo[b][i];
      for (int x = 1; x <= n_x; ++x)
        if (g[b][x].w == 0 or e_delta(g[xs][x]) <
            e_delta(g[b][x]))
          g[b][x] = g[xs][x], g[x][b] = g[x][xs];
      for (int x = 1; x <= n; ++x)
        if (flo_from[xs][x]) flo_from[b][x] = xs;
    }
    set_slack(b);
  }
  void expand_blossom(int b) {
    for (size_t i = 0; i < flo[b].size(); ++i) set_st(
        flo[b][i], flo[b][i]);
    int xr = flo_from[b][g[b][pa[b]].u], pr = get_pr(b
        , xr);
    for (int i = 0; i < pr; i += 2) {
      int xs = flo[b][i], xns = flo[b][i + 1];
      pa[xs] = g[xns][xs].u;
      S[xs] = 1, S[xns] = 0;
      slack[xs] = 0, set_slack(xns);
      q_push(xns);
    }
    S[xr] = 1, pa[xr] = pa[b];
    for (size_t i = pr + 1; i < flo[b].size(); ++i) {
      int xs = flo[b][i];
      S[xs] = -1, set_slack(xs);
    }
    st[b] = 0;
  }
  bool on_found_edge(const edge& e) {
    int u = st[e.u], v = st[e.v];
    if (S[v] == -1) {
      pa[v] = e.u, S[v] = 1;
      int nu = st[match[v]];
      slack[v] = slack[nu] = 0;
      S[nu] = 0, q_push(nu);
    } else if (S[v] == 0) {
      int lca = get_lca(u, v);
      if (not lca)
        return augment(u, v), augment(v, u), true;
      else
        add_blossom(u, lca, v);
    }
    return false;
  }
  bool matching() {
    memset(S + 1, -1, sizeof(int) * n_x);
    memset(slack + 1, 0, sizeof(int) * n_x);
    q = queue<int>();
    for (int x = 1; x <= n_x; ++x)
      if (st[x] == x and not match[x]) pa[x] = 0, S[x]
          = 0, q_push(x);
    if (q.empty()) return false;
    for (;;) {
      while (q.size()) {
        int u = q.front();
        q.pop();
        if (S[st[u]] == 1) continue;
        for (int v = 1; v <= n; ++v)
          if (g[u][v].w > 0 and st[u] != st[v]) {
            if (e_delta(g[u][v]) == 0) {
              if (on_found_edge(g[u][v])) return true;
            } else
              update_slack(u, st[v]);
          }
      }
      int d = INF;
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b and S[b] == 1) d = min(d, lab[b
            ] / 2);
      for (int x = 1; x <= n_x; ++x)
        if (st[x] == x and slack[x]) {
          if (S[x] == -1)
            d = min(d, e_delta(g[slack[x]][x]));
          else if (S[x] == 0)
            d = min(d, e_delta(g[slack[x]][x]) / 2);
        }
      for (int u = 1; u <= n; ++u) {
        if (S[st[u]] == 0) {
          if (lab[u] <= d) return 0;
          lab[u] -= d;
        } else if (S[st[u]] == 1)
          lab[u] += d;
      }
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b) {
          if (S[st[b]] == 0)
            lab[b] += d * 2;
          else if (S[st[b]] == 1)
            lab[b] -= d * 2;
        }
      q = queue<int>();
      for (int x = 1; x <= n_x; ++x)
        if (st[x] == x and slack[x] and st[slack[x]]
            != x and
            e_delta(g[slack[x]][x]) == 0)
          if (on_found_edge(g[slack[x]][x])) return
              true;
      for (int b = n + 1; b <= n_x; ++b)
        if (st[b] == b and S[b] == 1 and lab[b] == 0)
            expand_blossom(b);
    }
    return false;
  }
  pair<long long, int> solve() {
    memset(match + 1, 0, sizeof(int) * n);
    n_x = n;
    int n_matches = 0;
    long long tot_weight = 0;
    for (int u = 0; u <= n; ++u) st[u] = u, flo[u].
        clear();
    int w_max = 0;
    for (int u = 1; u <= n; ++u)
      for (int v = 1; v <= n; ++v) {
        flo_from[u][v] = (u == v ? u : 0);
        w_max = max(w_max, g[u][v].w);
      }
    for (int u = 1; u <= n; ++u) lab[u] = w_max;
    while (matching()) ++n_matches;
    for (int u = 1; u <= n; ++u)
      if (match[u] and match[u] < u) tot_weight += g[u
          ][match[u]].w;
    return {tot_weight, n_matches};
  }
  void add_edge(int ui, int vi, int wi) { g[ui][vi].w
      = g[vi][ui].w = wi; }
  void init(int _n) {  // 1-index, zero indicates
      unsaturated
    n = _n;
```

```
    for (int u = 1; u <= n; ++u)
      for (int v = 1; v <= n; ++v) g[u][v] = edge(u, v
          , 0);
  }
} graph;
```

6.8 MinMeanCycle

```
/* minimum mean cycle O(VE) */
struct MMC{
#define E 101010
#define V 1021
#define inf 1e9
#define eps 1e-6
  struct Edge { int v,u; double c; };
  int n, m, prv[V][V], prve[V][V], vst[V];
  Edge e[E];
  vector<int> edgeID, cycle, rho;
  double d[V][V];
  void init( int _n ) {
    n = _n;
    m = 0;
    memset(prv, 0, sizeof(prv));
    memset(prve, 0, sizeof(prve));
    memset(vst, 0, sizeof(vst));
  }
  // WARNING: TYPE matters
  void addEdge( int vi , int ui , double ci )
  { e[ m ++ ] = { vi , ui , ci }; }
  void bellman_ford() {
    for(int i=0; i<n; i++) d[0][i]=0;
    for(int i=0; i<n; i++) {
      fill(d[i+1], d[i+1]+n, inf);
      for(int j=0; j<m; j++) {
        int v = e[j].v, u = e[j].u;
        if(d[i][v]<inf && d[i+1][u]>d[i][v]+e[j].c) {
          d[i+1][u] = d[i][v]+e[j].c;
          prv[i+1][u] = v;
          prve[i+1][u] = j;
        }
      }
    }
  }
  double solve(){
    // returns inf if no cycle, mmc otherwise
    double mmc=inf;
    int st = -1;
    bellman_ford();
    for(int i=0; i<n; i++) {
      double avg=-inf;
      for(int k=0; k<n; k++) {
        if(d[n][i]<inf-eps) avg=max(avg,(d[n][i]-d[k][
            i])/(n-k));
        else avg=max(avg,inf);
      }
      if (avg < mmc) tie(mmc, st) = tie(avg, i);
    }
    FZ(vst); edgeID.clear(); cycle.clear(); rho.clear
        ();
    for (int i=n; !vst[st]; st=prv[i--][st]) {
      vst[st]++;
      edgeID.PB(prve[i][st]);
      rho.PB(st);
    }
    while (vst[st] != 2) {
      int v = rho.back(); rho.pop_back();
      cycle.PB(v);
      vst[v]++;
    }
    reverse(ALL(edgeID));
    edgeID.resize(SZ(cycle));
    return mmc;
  }
} mmc;
```

6.9 Prufer code

```
vector<int> Prufer_encode(vector<vector<int>> T) {
  int n = T.size();
  assert(n > 1);
  vector<int> deg(n), code;
  priority_queue<int, vector<int>, greater<int>> pq;
  for (int i = 0; i < n; ++i) {
    deg[i] = T[i].size();
    if (deg[i] == 1) pq.push(i);
  }
  while (code.size() < n - 2) {
    int v = pq.top(); pq.pop();
    --deg[v];
    for (int u: T[v]) {
      if (deg[u]) {
        --deg[u];
        code.push_back(u);
        if (deg[u] == 1) pq.push(u);
      }
    }
  }
  return code;
}
vector<vector<int>> Prufer_decode(vector<int> C) {
  int n = C.size() + 2;
  vector<vector<int>> T(n, vector<int>(0));
  vector<int> deg(n, 1); // outdeg
  for (int c: C) ++deg[c];
  priority_queue<int, vector<int>, greater<int>> q;
  for (int i = 0; i < n; ++i) if (deg[i] == 1) q.push(
      i);
  for (int c: C) {
    int v = q.top(); q.pop();
    T[v].push_back(c), T[c].push_back(v);
    --deg[c];
    --deg[v];
    if (deg[c] == 1) q.push(c);
  }
  int u = find(deg.begin(), deg.end(), 1) - deg.begin
      ();
  int v = find(deg.begin() + u + 1, deg.end(), 1) -
      deg.begin();
  T[u].push_back(v), T[v].push_back(u);
  return T;
}
```

6.10 Virtual Tree

```
struct Oracle {
  int lgn;
  vector<vector<int>> g;
  vector<int> dep;
  vector<vector<int>> par;
  vector<int> dfn;

  Oracle(const vector<vector<int>> &_g) : g(_g), lgn(
      ceil(log2(_g.size()))) {
    dep.resize(g.size());
    par.assign(g.size(), vector<int>(lgn + 1, -1));
    dfn.resize(g.size());

    int t = 0;
    function<void(int, int)> dfs = [&](int u, int fa)
        {
      // static int t = 0;
      dfn[u] = t++;
      if (~fa) dep[u] = dep[fa] + 1;
      par[u][0] = fa;
      for (int v : g[u]) if (v != fa) dfs(v, u);
    };
    dfs(0, -1);

    for (int i = 0; i < lgn; ++i)
      for (int u = 0; u < g.size(); ++u)
        par[u][i + 1] = ~par[u][i] ? par[par[u][i]][i]
            : -1;
  }

  int lca(int u, int v) const {
    if (dep[u] < dep[v]) swap(u, v);
    for (int i = lgn; dep[u] != dep[v]; --i) {
```

```
      if (dep[u] - dep[v] < 1 << i) continue;
      u = par[u][i];
    }
    if (u == v) return u;
    for (int i = lgn; par[u][0] != par[v][0]; --i) {
      if (par[u][i] == par[v][i]) continue;
      u = par[u][i];
      v = par[v][i];
    }
    return par[u][0];
  }
};

struct VirtualTree { // O(|C|lg|G|), C is the set of critical points, G is nodes in original graph
  vector<int> cp; // index of critical points in original graph
  vector<vector<int>> g; // simplified tree, i.e. virtual tree
  vector<int> nodes; // i'th node in g has index nodes[i] in original graph
  map<int, int> mp; // inverse of nodes

  VirtualTree(const vector<int> &_cp, const Oracle &oracle) : cp(_cp) {
    sort(cp.begin(), cp.end(), [&](int u, int v) { return oracle.dfn[u] < oracle.dfn[v]; });
    nodes = cp;
    for (int i = 0; i < nodes.size(); ++i) mp[nodes[i]] = i;
    g.resize(nodes.size());

    if (!mp.count(0)) {
      mp[0] = nodes.size();
      nodes.emplace_back(0);
      g.emplace_back(vector<int>());
    }

    vector<int> stk;
    stk.emplace_back(0);

    for (int u : cp) {
      if (u == stk.back()) continue;
      int p = oracle.lca(u, stk.back());
      if (p == stk.back()) {
        stk.emplace_back(u);
      } else {
        while (stk.size() > 1 && oracle.dep[stk.end()[-2]] >= oracle.dep[p]) {
          g[mp[stk.back()]].emplace_back(mp[stk.end()[-2]]);
          g[mp[stk.end()[-2]]].emplace_back(mp[stk.back()]);
          stk.pop_back();
        }
        if (stk.back() != p) {
          if (!mp.count(p)) {
            mp[p] = nodes.size();
            nodes.emplace_back(p);
            g.emplace_back(vector<int>());
          }
          g[mp[p]].emplace_back(mp[stk.back()]);
          g[mp[stk.back()]].emplace_back(mp[p]);
          stk.pop_back();
          stk.emplace_back(p);
        }
        stk.emplace_back(u);
      }
    }
    for (int i = 0; i + 1 < stk.size(); ++i) {
      g[mp[stk[i]]].emplace_back(mp[stk[i + 1]]);
      g[mp[stk[i + 1]]].emplace_back(mp[stk[i]]);
    }
  }
};
```

6.11 Graph Sequence Test

```
bool Erdos_Gallai(vector<LL> d) {
  if (accumulate(d.begin(), d.end(), 0ll)&1) return false;
  sort(d.rbegin(), d.rend());
  const int n = d.size();
  vector<LL> pre(n + 1, 0);
  for (int i = 0; i < n; ++i) pre[i + 1] += pre[i] + d[i];
  for (int k = 1, j = n; k <= n; ++k) {
    while (k < j and (d[j - 1] <= k)) --j; // [0, k), > : [k, j), <= : [j, n)
    j = max(k, j);
    if (pre[k] > (LL)k * (k - 1) + pre[n] - pre[j] + (LL)k * (j - k))
      return false;
  }
  return true;
}
```

6.12 maximal cliques

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

class MaxClique {
 public:
  static const int MV = 100;

  int V;
  int el[MV][MV / 30 + 1];
  int dp[MV];
  int ans;
  int s[MV][MV / 30 + 1];
  vector<int> sol;

  void init(int v) {
    V = v;
    ans = 0;
    memset(el, 0, sizeof(el));
    memset(dp, 0, sizeof(dp));
  }

  /* Zero Base */
  void addEdge(int u, int v) {
    if (u > v) swap(u, v);
    if (u == v) return;
    el[u][v / 32] |= (1 << (v % 32));
  }

  bool dfs(int v, int k) {
    int c = 0, d = 0;
    for (int i = 0; i < (V + 31) / 32; i++) {
      s[k][i] = el[v][i];
      if (k != 1) s[k][i] &= s[k - 1][i];
      c += __builtin_popcount(s[k][i]);
    }
    if (c == 0) {
      if (k > ans) {
        ans = k;
        sol.clear();
        sol.push_back(v);
        return 1;
      }
      return 0;
    }
    for (int i = 0; i < (V + 31) / 32; i++) {
      for (int a = s[k][i]; a; d++) {
        if (k + (c - d) <= ans) return 0;
        int lb = a & (-a), lg = 0;
        a ^= lb;
        while (lb != 1) {
          lb = (unsigned int)(lb) >> 1;
          lg++;
        }
        int u = i * 32 + lg;
        if (k + dp[u] <= ans) return 0;
        if (dfs(u, k + 1)) {
          sol.push_back(v);
          return 1;
        }
      }
```

```
    }
    return 0;
  }

  int solve() {
    for (int i = V - 1; i >= 0; i--) {
      dfs(i, 1);
      dp[i] = ans;
    }
    return ans;
  }
};

signed main() {
  int N;
  cin >> N;
  MaxClique mc;
  mc.init(N);
  mc.addEdge(i, j);
  cout << mc.solve() << endl;
}
```

6.13 scc

```
class Kosaraju {

  vector<vector<int>> g, rg, compo;
  vector<int> order, DAGID;
  vector<bool> vis;
  int n, iter;

  void make_rg() {
    for (int u = 0; u < n; ++u) for (int v : g[u]) rg[v].push_back(u);
  }

  void dfs_all() {
    function<void(int)> dfs = [&](int u) {
      vis[u] = true;
      for (int v : g[u]) if (not vis[v]) dfs(v);
      order.emplace_back(u);
    };
    for (int i = 0; i < n; ++i) if (not vis[i]) dfs(i);
  }

  void rdfs_all() {
    function<void(int)> rdfs = [&](int u) {
      DAGID[u] = iter;
      for (int v : rg[u]) if (DAGID[v] == -1) rdfs(v);
      compo.back().push_back(u);
    };
    for (int u : order) if (DAGID[u] == -1) {
      compo.push_back(vector<int>(0));
      rdfs(u), ++iter;
    }
  }

public:

  // remember that the graph is directed
  Kosaraju(vector<vector<int>> &_g) : n(_g.size()), g(_g) {
    rg.resize(n);
    compo.clear();
    make_rg();
    vis.assign(n, false);
    DAGID.assign(n, -1);
    iter = 0;

    dfs_all();
    reverse(order.begin(), order.end());
    rdfs_all();
  }

  const vector<vector<int>>& get_components() { return compo; }

  const vector<vector<int>> get_condensed_DAG(bool simple = true) {
    vector<vector<int>> ret(iter);
    for (int i = 0; i < iter; ++i) {
      for (int u : compo[i]) for (int v : g[u]) if (DAGID[v] != i) {
        ret[i].push_back(DAGID[v]);
      }
      if (simple) {
        sort(ret[i].begin(), ret[i].end());
        ret[i].resize(unique(ret[i].begin(), ret[i].end()) - ret[i].begin());
      }
    }
    return ret;
  }
};
```

7 String

7.1 AC automaton

```
// SIGMA[0] will not be considered
const string SIGMA = "
    _0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqr
    ";
vector<int> INV_SIGMA;
const int SGSZ = 63;

struct PMA {
  PMA *next[SGSZ];  // next[0] is for fail
  vector<int> ac;
  PMA *last; // state of longest accepted string that is pre of this
  PMA() : last(nullptr) { fill(next, next + SGSZ, nullptr); }
};

template<typename T>
PMA *buildPMA(const vector<T> &p) {
  PMA *root = new PMA;
  for (int i = 0; i < p.size(); ++i) {  // make trie
    PMA *t = root;
    for (int j = 0; j < p[i].size(); ++j) {
      int c = INV_SIGMA[p[i][j]];
      if (t->next[c] == nullptr) t->next[c] = new PMA;
      t = t->next[c];
    }
    t->ac.push_back(i);
  }
  queue<PMA *> que;  // make failure link using bfs
  for (int c = 1; c < SGSZ; ++c) {
    if (root->next[c]) {
      root->next[c]->next[0] = root;
      que.push(root->next[c]);
    } else root->next[c] = root;
  }
  while (!que.empty()) {
    PMA *t = que.front();
    que.pop();
    for (int c = 1; c < SGSZ; ++c) {
      if (t->next[c]) {
        que.push(t->next[c]);
        PMA *r = t->next[0];
        while (!r->next[c]) r = r->next[0];
        t->next[c]->next[0] = r->next[c];
        t->next[c]->last = r->next[c]->ac.size() ? r->next[c] : r->next[c]->last;
      }
    }
  }
  return root;
}

void destructPMA(PMA *root) {
  queue<PMA *> que;
  que.emplace(root);
  while (!que.empty()) {
    PMA *t = que.front();
    que.pop();
```

```
    for (int c = 1; c < SGSZ; ++c) {
      if (t->next[c] && t->next[c] != root) que.emplace(t->next[c]);
    }
    delete t;
  }
}

template<typename T>
map<int, int> match(const T &t, PMA *v) {
  map<int, int> res;
  for (int i = 0; i < t.size(); ++i) {
    int c = INV_SIGMA[t[i]];
    while (!v->next[c]) v = v->next[0];
    v = v->next[c];
    for (int j = 0; j < v->ac.size(); ++j) ++res[v->ac[j]];
    for (PMA *q = v->last; q; q = q->last) {
      for (int j = 0; j < q->ac.size(); ++j) ++res[q->ac[j]];
    }
  }
  return res;
}

signed main() {
  INV_SIGMA.assign(256, -1);
  for (int i = 0; i < SIGMA.size(); ++i) {
    INV_SIGMA[SIGMA[i]] = i;
  }

}
```

7.2 KMP

```
template<typename T>
vector<int> build_kmp(const T &s) {
  vector<int> f(s.size());
  int fp = f[0] = -1;
  for (int i = 1; i < s.size(); ++i) {
    while (~fp && s[fp + 1] != s[i]) fp = f[fp];
    if (s[fp + 1] == s[i]) ++fp;
    f[i] = fp;
  }
  return f;
}
template<typename S>
vector<int> kmp_match(vector<int> fail, const S &P, const S &T) {
  vector<int> res; // start from these points
  const int n = P.size();
  for (int j = 0, i = -1; j < T.size(); ++j) {
    while (~i and T[j] != P[i + 1]) i = fail[i];
    if (P[i + 1] == T[j]) ++i;
    if (i == n - 1) res.push_back(j - n + 1), i = fail[i];
  }
  return res;
}
```

7.3 Manacher

```
template<typename T, int INF>
vector<int> manacher(const T &s) { // p = "INF" + s.join("INF") + "INF", returns radius on p
  vector<int> p(s.size() * 2 + 1, INF);
  for (int i = 0; i < s.size(); ++i) {
    p[i << 1 | 1] = s[i];
  }
  vector<int> w(p.size());
  for (int i = 1, j = 0, r = 0; i < p.size(); ++i) {
    int t = min(r >= i ? w[2 * j - i] : 0, r - i + 1);
    for ( ; i - t >= 0 && i + t < p.size(); ++t) {
      if (p[i - t] != p[i + t]) break;
    }
    w[i] = --t;
    if (i + t > r) r = i + t, j = i;
  }
  return w;
}
```

7.4 Suffix Array

```
// -----------O(NlgNlgN)----------
vector<int> sa_db(const string &s) {
  int n = s.size();
  vector<int> sa(n), r(n), t(n);
  for (int i = 0; i < n; ++i) r[sa[i] = i] = s[i];
  for (int h = 1; t[n - 1] != n - 1; h *= 2) {
    auto cmp = [&](int i, int j) {
      if (r[i] != r[j]) return r[i] < r[j];
      return i + h < n && j + h < n ? r[i + h] < r[j + h] : i > j;
    };
    sort(sa.begin(), sa.end(), cmp);
    for (int i = 0; i + 1 < n; ++i) t[i + 1] = t[i] + cmp(sa[i], sa[i + 1]);
    for (int i = 0; i < n; ++i) r[sa[i]] = t[i];
  }
  return sa;
}

// O(N) -- CF: 1e6->31ms,18MB;1e7->296ms;158MB;3e7->856ms,471MB
bool is_lms(const string &t, int i) {
  return i > 0 && t[i - 1] == 'L' && t[i] == 'S';
}

template<typename T>
vector<int> induced_sort(const T &s, const string &t, const vector<int> &lmss, int sigma = 256) {
  vector<int> sa(s.size(), -1);

  vector<int> bin(sigma + 1);
  for (auto it = s.begin(); it != s.end(); ++it) {
    ++bin[*it + 1];
  }

  int sum = 0;
  for (int i = 0; i < bin.size(); ++i) {
    sum += bin[i];
    bin[i] = sum;
  }

  vector<int> cnt(sigma);
  for (auto it = lmss.rbegin(); it != lmss.rend(); ++it) {
    int ch = s[*it];
    sa[bin[ch + 1] - 1 - cnt[ch]] = *it;
    ++cnt[ch];
  }

  cnt = vector<int>(sigma);
  for (auto it = sa.begin(); it != sa.end(); ++it) {
    if (*it <= 0 || t[*it - 1] == 'S') continue;
    int ch = s[*it - 1];
    sa[bin[ch] + cnt[ch]] = *it - 1;
    ++cnt[ch];
  }

  cnt = vector<int>(sigma);
  for (auto it = sa.rbegin(); it != sa.rend(); ++it) {
    if (*it <= 0 || t[*it - 1] == 'L') continue;
    int ch = s[*it - 1];
    sa[bin[ch + 1] - 1 - cnt[ch]] = *it - 1;
    ++cnt[ch];
  }

  return sa;
}

template<typename T>
vector<int> sa_is(const T &s, int sigma = 256) {
  string t(s.size(), 0);
  t[s.size() - 1] = 'S';
  for (int i = int(s.size()) - 2; i >= 0; --i) {
    if (s[i] < s[i + 1]) t[i] = 'S';
    else if (s[i] > s[i + 1]) t[i] = 'L';
```

```
    else t[i] = t[i + 1];
  }

  vector<int> lmss;
  for (int i = 0; i < s.size(); ++i) {
    if (is_lms(t, i)) {
      lmss.emplace_back(i);
    }
  }

  vector<int> sa = induced_sort(s, t, lmss, sigma);
  vector<int> sa_lms;
  for (int i = 0; i < sa.size(); ++i) {
    if (is_lms(t, sa[i])) {
      sa_lms.emplace_back(sa[i]);
    }
  }

  int lmp_ctr = 0;
  vector<int> lmp(s.size(), -1);
  lmp[sa_lms[0]] = lmp_ctr;
  for (int i = 0; i + 1 < sa_lms.size(); ++i) {
    int diff = 0;
    for (int d = 0; d < sa.size(); ++d) {
      if (s[sa_lms[i] + d] != s[sa_lms[i + 1] + d] ||
          is_lms(t, sa_lms[i] + d) != is_lms(t, sa_lms
              [i + 1] + d)) {
        diff = 1; // something different in range of
            lms
        break;
      } else if (d > 0 && is_lms(t, sa_lms[i] + d) &&
          is_lms(t, sa_lms[i + 1] + d)) {
        break; // exactly the same
      }
    }
    if (diff) ++lmp_ctr;
    lmp[sa_lms[i + 1]] = lmp_ctr;
  }

  vector<int> lmp_compact;
  for (int i = 0; i < lmp.size(); ++i) {
    if (~lmp[i]) {
      lmp_compact.emplace_back(lmp[i]);
    }
  }

  if (lmp_ctr + 1 < lmp_compact.size()) {
    sa_lms = sa_is(lmp_compact, lmp_ctr + 1);
  } else {
    for (int i = 0; i < lmp_compact.size(); ++i) {
      sa_lms[lmp_compact[i]] = i;
    }
  }

  vector<int> seed;
  for (int i = 0; i < sa_lms.size(); ++i) {
    seed.emplace_back(lmss[sa_lms[i]]);
  }

  return induced_sort(s, t, seed, sigma);
} // s must end in char(0)

// O(N) lcp, note that s must end in '\0'
vector<int> build_lcp(const string &s, const vector<
    int> &sa, const vector<int> &rank) {
  int n = s.size();
  vector<int> lcp(n);
  for (int i = 0, h = 0; i < n; ++i) {
    if (rank[i] == 0) continue;
    int j = sa[rank[i] - 1];
    if (h > 0) --h;
    for ( ; j + h < n && i + h < n; ++h) {
      if (s[j + h] != s[i + h]) break;
    }
    lcp[rank[i] - 1] = h;
  }
  return lcp; // lcp[i] := lcp(s[sa[i]..-1], s[sa[i +
      1]..-1])
}

// O(N) build segment tree for lcp
vector<int> build_lcp_rmq(const vector<int> &lcp) {
  vector<int> sgt(lcp.size() << 2);
  function<void(int, int, int)> build = [&](int t, int
      lb, int rb) {
    if (rb - lb == 1) return sgt[t] = lcp[lb], void();
    int mb = lb + rb >> 1;
    build(t << 1, lb, mb);
    build(t << 1 | 1, mb, rb);
    sgt[t] = min(sgt[t << 1], sgt[t << 1 | 1]);
  };
  build(1, 0, lcp.size());
  return sgt;
}

// O(|P| + lg |T|) pattern searching, returns last
    index in sa
int match(const string &p, const string &s, const
    vector<int> &sa, const vector<int> &rmq) { // rmq
    is segtree on lcp
  int t = 1, lb = 0, rb = s.size(); // answer in [lb,
      rb)
  int lcplp = 0; // lcp(char(0), p) = 0
  while (rb - lb > 1) {
    int mb = lb + rb >> 1;
    int lcplm = rmq[t << 1];
    if (lcplp < lcplm) t = t << 1 | 1, lb = mb;
    else if (lcplp > lcplm) t = t << 1, rb = mb;
    else {
      int lcpmp = lcplp;
      while (lcpmp < p.size() && p[lcpmp] == s[sa[mb]
          + lcpmp]) ++lcpmp;
      if (lcpmp == p.size() || p[lcpmp] > s[sa[mb] +
          lcpmp]) t = t << 1 | 1, lb = mb, lcplp =
          lcpmp;
      else t = t << 1, rb = mb;
    }
  }
  if (lcplp < p.size()) return -1;
  return sa[lb];
}
```

7.5 Suffix Automaton

```
template<typename T>
struct SuffixAutomaton {
  vector<map<int, int>> edges;// edges[i]  : the
      labeled edges from node i
  vector<int> link;           // link[i]   : the
      parent of i
  vector<int> length;         // length[i] : the
      length of the longest string in the ith class
  int last;                   // the index of the
      equivalence class of the whole string
  vector<bool> is_terminal;   // is_terminal[i] : some
       suffix ends in node i (unnecessary)
  vector<int> occ;            // occ[i] : number of
      matches of maximum string of node i (unnecessary
      )
  SuffixAutomaton(const T &s) : edges({map<int, int>()
      }), link({-1}), length({0}), last(0), occ({0}) {
    for (int i = 0; i < s.size(); ++i) {
      edges.push_back(map<int, int>());
      length.push_back(i + 1);
      link.push_back(0);
      occ.push_back(1);
      int r = edges.size() - 1;
      int p = last; // add edges to r and find p with
          link to q
      while (p >= 0 && edges[p].find(s[i]) == edges[p
          ].end()) {
        edges[p][s[i]] = r;
        p = link[p];
      }
      if (~p) {
        int q = edges[p][s[i]];
        if (length[p] + 1 == length[q]) { // no need
            to split q
          link[r] = q;
        } else { // split q, add qq
          edges.push_back(edges[q]);  // copy edges of
              q
```

```
            length.push_back(length[p] + 1);
            link.push_back(link[q]);  // copy parent of
                q
            occ.push_back(0);
            int qq = edges.size() - 1; // qq is new
                parent of q and r
            link[q] = qq;
            link[r] = qq;
            while (p >= 0 && edges[p][s[i]] == q) { //
                what points to q points to qq
              edges[p][s[i]] = qq;
              p = link[p];
            }
          }
        }
        last = r;
      } // below unnecessary
      is_terminal = vector<bool>(edges.size());
      for (int p = last; p > 0; p = link[p]) is_terminal
          [p] = 1; // is_terminal calculated
      vector<int> cnt(link.size()), states(link.size());
           // sorted states by length
      for (int i = 0; i < link.size(); ++i) ++cnt[length
          [i]];
      for (int i = 0; i < s.size(); ++i) cnt[i + 1] +=
          cnt[i];
      for (int i = link.size() - 1; i >= 0; --i) states
          [--cnt[length[i]]] = i;
      for (int i = link.size() - 1; i >= 1; --i) occ[
          link[states[i]]] += occ[states[i]]; // occ
          calculated
    }
};
```

8 Formulas

8.1 Pick's theorem

For a polygon:
A: The area of the polygon
B: Boundary Point: a lattice point on the polygon (including vertices) I: Interior Point: a lattice point in the polygon' s interior region

$$A = I + \frac{B}{2} - 1$$

8.2 Graph Properties

1. Euler's Formula $V - E + F = 2$
2. For a planar graph, $F = E - V + n + 1$, n is the numbers of components
3. For a planar graph, $E \leq 3V - 6$

   For a connected graph G: $I(G)$: the size of maximum independent set $M(G)$: the size of maximum matching $Cv(G)$: be the size of minimum vertex cover $Ce(G)$: be the size of minimum edge cover
4. For any connected graph:
   (a) $I(G) + Cv(G) = |V|$
   (b) $M(G) + Ce(G) = |V|$
5. For any bipartite:
   (a) $I(G) = Cv(G)$
   (b) $M(G) = Ce(G)$

8.3 Number Theory

1. $g(m) = \sum_{d|m} f(d) \Leftrightarrow f(m) = \sum_{d|m} \mu(d) \times g(m/d)$
2. $\phi(x), \mu(x)$ are Möbius inverse
3. $\sum_{i=1}^{n} \sum_{j=1}^{m} [\gcd(i,j) = 1] = \sum \mu(d) \left\lfloor \frac{n}{d} \right\rfloor \left\lfloor \frac{m}{d} \right\rfloor$
4. $\sum_{i=1}^{n} \sum_{j=1}^{n} lcm(i,j) = n \sum_{d|n} d \times \phi(d)$

8.4 Combinatorics

1. Gray Code: $= n \oplus (n >> 1)$
2. Catalan Number:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{n!(n+1)!} = \prod_{k=2}^{n} \frac{n+k}{k}$$

3. $\Gamma(n+1) = n!$
4. $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$
5. Stirling number of second kind: the number of ways to partition a set of n elements into k nonempty subsets.
   (a) $\left\{ {0 \atop 0} \right\} = \left\{ {n \atop n} \right\} = 1$
   (b) $\left\{ {n \atop 0} \right\} = 0$
   (c) $\left\{ {n \atop k} \right\} = k\left\{ {n-1 \atop k} \right\} + \left\{ {n-1 \atop k-1} \right\}$
6. Bell numbers count the possible partitions of a set:
   (a) $B_0 = 1$
   (b) $B_n = \sum_{k=0}^{n} \left\{ {n \atop k} \right\}$
   (c) $B_{n+1} = \sum_{k=0}^{n} C_k^n B_k$
   (d) $B_{p+n} \equiv B_n + B_{n+1} \mod p$, p prime
   (e) $B_{p^m+n} \equiv mB_n + B_{n+1} \mod p$, p prime
   (f) From $B_0 : 1, 1, 2, 5, 15, 52$, $203, 877, 4140, 21147, 115975$
7. Derangement
   (a) $D_n = n!(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} \ldots + (-1)^n \frac{1}{n!})$
   (b) $D_n = (n-1)(D_{n-1} + D_{n-2})$
   (c) From $D_0 : 1, 0, 1, 2, 9, 44$, $265, 1854, 14833, 133496$
8. Binomial Equality
   (a) $\sum_k \binom{r}{m+k}\binom{s}{n-k} = \binom{r+s}{m+n}$
   (b) $\sum_k \binom{l}{m+k}\binom{s}{n+k} = \binom{l+s}{l-m+n}$
   (c) $\sum_k \binom{l}{m+k}\binom{s+k}{n}(-1)^k = (-1)^{l+m}\binom{s-m}{n-l}$
   (d) $\sum_{k \leq l} \binom{l-k}{m}\binom{s}{k-n}(-1)^k = (-1)^{l+m}\binom{s-m-1}{l-n-m}$
   (e) $\sum_{0 \leq k \leq l} \binom{l-k}{m}\binom{q+k}{n} = \binom{l+q+1}{m+n+1}$
   (f) $\binom{r}{k} = (-1)^k\binom{k-r-1}{k}$
   (g) $\binom{r}{m}\binom{m}{k} = \binom{r}{k}\binom{r-k}{m-k}$
   (h) $\sum_{k \leq n} \binom{r+k}{k} = \binom{r+n+1}{n}$
   (i) $\sum_{0 \leq k \leq n} \binom{k}{m} = \binom{n+1}{m+1}$
   (j) $\sum_{k \leq m} \binom{m+r}{k} x^k y^k = \sum_{k \leq m} \binom{-r}{k}(-x)^k (x+y)^{m-k}$

8.5 Sum of Powers

1. $a^b \% P = a^{b \% \varphi(p) + \varphi(p)}, b \geq \varphi(p)$
2. $1^3 + 2^3 + 3^3 + \ldots + n^3 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$
3. $1^4 + 2^4 + 3^4 + \ldots + n^4 = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$
4. $1^5 + 2^5 + 3^5 + \ldots + n^5 = \frac{n^6}{6} + \frac{n^5}{2} + \frac{5n^4}{12} - \frac{n^2}{12}$
5. $0^k + 1^k + 2^k + \ldots + n^k = P_k, P_k = \frac{(n+1)^{k+1} - \sum_{i=0}^{k-1} C_i^{k+1} P(i)}{k+1}, P_0 = n+1$
6. $\sum_{k=0}^{m-1} k^n = \frac{1}{n+1} \sum_{k=0}^{n} C_k^{n+1} B_k m^{n+1-k}$
7. $\sum_{j=0}^{m} C_j^{m+1} B_j = 0, B_0 = 1$
8. 除了 $B_1 = -1/2$，剩下的奇數項都是 0
9. $B_2 = 1/6, B_4 = -1/30, B_6 = 1/42, B_8 = -1/30, B_{10} = 5/66, B_{12} = -691/2730, B_{14} = 7/6, B_{16} = -3617/510, B_{18} = 43867/798, B_{20} = -174611/330,$

8.6 Burnside's lemma

1. $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$
2. $X^g = t^{c(g)}$

8.7 Count on a tree

1. Rooted tree: $s_{n+1} = \frac{1}{n} \sum_{i=1}^{n} (i \times a_i \times \sum_{j=1}^{\lfloor n/i \rfloor} a_{n+1-i \times j})$
2. Unrooted tree:
   (a) Odd:$a_n - \sum_{i=1}^{n/2} a_i a_{n-i}$
   (b) Even:$Odd + \frac{1}{2} a_{n/2}(a_{n/2} + 1)$
3. Spanning Tree
   (a) 完全圖 $n^n - 2$
   (b) 一般圖 (Kirchhoff's theorem)$M[i][i] = \deg(V_i)$,$M[i][j] = -1$,if have $E(i,j)$,0 if no edge. delete any one row and col in A, $ans = \det(A)$
4. Ordered Binary Tree with N nodes and Y leaves: $\frac{N-1C_{Y-1} \times N-2C_{Y-1}}{Y}$