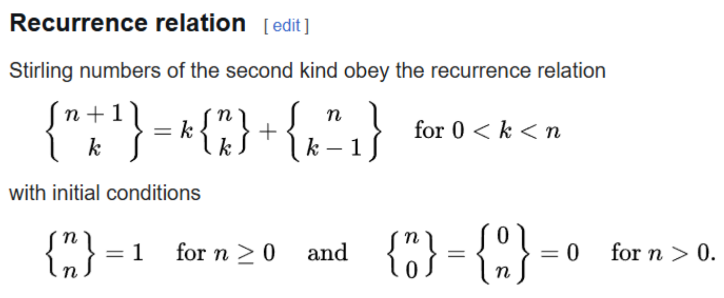
```
// ### NILOY
ll mod = 1e9 + 7, MOD = 998244353;
double eps = 1e-12;

// Checks if kth bit of x is set (1) or not (0)
int check_kth_bit(int x, int k)
{
    return (x >> k) & 1;
}

// Prints the positions of all set (1) bits in binary
representation of x
void print_on_bits(int x)
{
    for (int k = 0; k < 32; k++) if (check_kth_bit(x, k))
     cout << k << ' '; // prints the position of the set bit
    cout << '\n';
}

// Returns the count of set (1) bits in binary representation
of x
int count_on_bits(int x)
{
    int ans = 0;
    for (int k = 0; k < 32; k++) if (check_kth_bit(x, k))
ans++;
    return ans;
}

// Sets the kth bit of x to 1 and returns the result
int set_kth_bit(int x, int k)
{
    return x | (1 << k);
}

// Sets the kth bit of x to 0 and returns the result
int unset_kth_bit(int x, int k)
{
    return x & (~(1 << k));
}

// Toggles the kth bit of x and returns the result
int toggle_kth_bit(int x, int k)
{
    return x ^ (1 << k);
}

int ncr(int n, int r)
{
    if (n < r) return 0;
    if (n == r or r == 0) return 1;
    else return ncr(n - 1, r) + ncr(n - 1, r - 1);
}

const int N = 1e6 + 2;
int power(int x, long long n,int mod) { // O(log n)
  int ans = 1 % mod;
  while (n > 0) {
    if (n & 1) ans = 1LL * ans * x % mod;
    x = 1LL * x * x % mod;
    n >>= 1;
  }
  return ans;
}

int inverse(int a,int mod) {
  return power(a, mod - 2,mod);
}

int fac[N], ifac[N];

void prec(int mod) { // O(n)
  fac[0] = 1;
  for (int i = 1; i < N; i++) fac[i] = 1LL * fac[i - 1] * i %
mod;
  ifac[N - 1] = inverse(fac[N - 1],mod);
  for (int i = N - 2; i >= 0; i--) ifac[i] = 1LL * ifac[i +
1] * (i + 1) % mod;
}

int nCr(int n, int r,int mod) { // O(1)
  if (n < 0 or n < r) return 0;
  return 1LL * fac[n] * ifac[r] % mod * ifac[n - r] % mod;
}

int nPr(int n, int r,int mod) {
  if (n < 0 or n < r) return 0;
  return 1LL * fac[n] * ifac[n - r] % mod;
}
const int N = 1e7 + 10;
vector<bool> isPrime(N, true);
vector<int> lp(N, 0), hp(N, 0);
void sieve()
{
  isPrime[0] = isPrime[1] = false;
  for (int i = 2; i < N; i++)
  {
    if(isPrime[i])
    {
      lp[i] = hp[i] = i;
      for (int j = 2 * i; j < N; j += i)
      {
        isPrime[j] = false;
        hp[j] = i;
        if(lp[j] == 0) lp[j] = i;
      }
    }
  }
}

// Linear Sieve
int spf[N];
vector<int> primes;
void sieve() {
  for(int i = 2; i < N; i++) {
    if (spf[i] == 0) spf[i] = i, primes.push_back(i);
    int sz = primes.size();
    for (int j = 0; j < sz && i * primes[j] < N && primes[j]
<= spf[i]; j++) {
      spf[i * primes[j]] = primes[j];
    }
  }
}

// Miller Robin
namespace MillerRabin {
  mt19937
rnd(chrono::steady_clock::now().time_since_epoch().count());
  const int P = 1e6 + 9;
  int primes[P], spf[P];
  inline ll mul_mod(ll x, ll y, ll m) {
    ll res = __int128(x) * y % m;
    return res;
    // ll res = x * y - (ll)((long double)x * y / m + 0.5) *
m;
    // return res < 0 ? res + m : res;
  }
  inline ll pow_mod(ll x, ll n, ll m) {
    ll res = 1 % m;
    for (; n; n >>= 1) {
      if (n & 1) res = mul_mod(res, x, m);
      x = mul_mod(x, x, m);
    }
    return res;
  }
  // O(it * (logn)^3), it = number of rounds performed (but
faster in practice)
  inline bool miller_rabin(ll n) {
    if (n <= 2 || (n & 1 ^ 1)) return (n == 2);
    if (n < P) return spf[n] == n;
    ll c, d, s = 0, r = n - 1;
    for (; !(r & 1); r >>= 1, s++) {}
    // each iteration is a round
    for (int i = 0; primes[i] < n && primes[i] < 32; i++) {
      c = pow_mod(primes[i], r, n);
      for (int j = 0; j < s; j++) {
        d = mul_mod(c, c, n);
        if (d == 1 && c != 1 && c != (n - 1)) return false;
        c = d;
      }
      if (c != 1) return false;
    }
    return true;
  }
  void init() {
    int cnt = 0;
    for (int i = 2; i < P; i++) {
      if (!spf[i]) primes[cnt++] = spf[i] = i;
      for (int j = 0, k; (k = i * primes[j]) < P; j++) {
        spf[k] = primes[j];
        if (spf[i] == spf[k]) break;
      }
    }
  }
}


namespace PollardRho {
  mt19937
rnd(chrono::steady_clock::now().time_since_epoch().count());
  const int P = 1e6 + 9;
  ll seq[P];
  int primes[P], spf[P];
  inline ll add_mod(ll x, ll y, ll m) {
    return (x += y) < m ? x : x - m;
  }
  inline ll mul_mod(ll x, ll y, ll m) {
    ll res = __int128(x) * y % m;
    return res;
  }
  inline ll pow_mod(ll x, ll n, ll m) {
    ll res = 1 % m;
    for (; n; n >>= 1) {
      if (n & 1) res = mul_mod(res, x, m);
      x = mul_mod(x, x, m);
    }
    return res;
  }
  // O(it * (logn)^3), it = number of rounds performed
  inline bool miller_rabin(ll n) {
    if (n <= 2 || (n & 1 ^ 1)) return (n == 2);
    if (n < P) return spf[n] == n;
    ll c, d, s = 0, r = n - 1;
    for (; !(r & 1); r >>= 1, s++) {}
    // each iteration is a round
    for (int i = 0; primes[i] < n && primes[i] < 32; i++) {
      c = pow_mod(primes[i], r, n);
      for (int j = 0; j < s; j++) {
        d = mul_mod(c, c, n);
        if (d == 1 && c != 1 && c != (n - 1)) return false;
        c = d;
      }
      if (c != 1) return false;
    }
    return true;
  }
  void init() {
    int cnt = 0;
    for (int i = 2; i < P; i++) {
      if (!spf[i]) primes[cnt++] = spf[i] = i;
      for (int j = 0, k; (k = i * primes[j]) < P; j++) {
        spf[k] = primes[j];
        if (spf[i] == spf[k]) break;
      }
    }
  }
  // returns O(n^(1/4))
  ll pollard_rho(ll n) {
    while (1) {
      ll x = rnd() % n, y = x, c = rnd() % n, u = 1, v, t =
0;
      ll *px = seq, *py = seq;
      while (1) {
        *py++ = y = add_mod(mul_mod(y, y, n), c, n);
        *py++ = y = add_mod(mul_mod(y, y, n), c, n);
        if ((x = *px++) == y) break;
        v = u;
        u = mul_mod(u, abs(y - x), n);
        if (!u) return __gcd(v, n);
        if (++t == 32) {
          t = 0;
          if ((u = __gcd(u, n)) > 1 && u < n) return u;
        }
      }
      if (t && (u = __gcd(u, n)) > 1 && u < n) return u;
    }
  }
  vector<ll> factorize(ll n) {
    if (n == 1) return vector <ll>();
    if (miller_rabin(n)) return vector<ll> {n};
    vector <ll> v, w;
    while (n > 1 && n < P) {
      v.push_back(spf[n]);
      n /= spf[n];
    }
    if (n >= P) {
      ll x = pollard_rho(n);
      v = factorize(x);
      w = factorize(n / x);
      v.insert(v.end(), w.begin(), w.end());
    }
    return v;
  }
}

int phi[N]; // 1e5 + 9
void totient() {
  for (int i = 1; i < N; i++) phi[i] = i;
  for (int i = 2; i < N; i++) {
    if (phi[i] == i) {
      for (int j = i; j < N; j += i) phi[j] -= phi[j] / i;
    }
  }
}
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}

// +++ power tower
const int N = 1e5 + 9;
map<ll, ll> mp;
ll phi(ll n) {
  if (mp.count(n)) return mp[n];
  ll ans = n, m = n;
  for (ll i = 2; i * i <= m; i++) {
    if (m % i == 0) {
      while (m % i == 0) m /= i;
      ans = ans / i * (i - 1);
    }
  }
  if (m > 1) ans = ans / m * (m - 1);
  return mp[n] = ans;
}
inline ll MOD(ll x, ll m) {
  if (x < m) return x;
  return x % m + m;
}
ll power(ll n, ll k, ll mod) {
  ll ans = MOD(1, mod);
  while (k) {
    if (k & 1) ans = MOD(ans * n, mod);
    n = MOD(n * n, mod);
    k >>= 1;
  }
  return ans;
}
int a[N];
// if x >= log2(m), then a^x = a^(MOD(x, phi(m))) % m
ll yo(ll l, ll r, ll m) {
  if (l == r) return MOD(a[l], m);
  if (m == 1) return 1;
  return power(a[l], yo(l + 1, r, phi(m)), m);
}

// +++ extended euclidian
ll extended_euclid(ll a, ll b, ll &x, ll &y) {
  if (b == 0) {
    x = 1; y = 0;
    return a;
  }
  ll x1, y1;
  ll d = extended_euclid(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
ll inverse(ll a, ll m) {
  ll x, y;
  ll g = extended_euclid(a, m, x, y);
  if (g != 1) return -1;
  return (x % m + m) % m;
}



// +++ crt (chinese remainder theorem)
using T = __int128;
// ax + by = __gcd(a, b)
// returns __gcd(a, b)
T extended_euclid(T a, T b, T &x, T &y) {
  T xx = y = 0;
  T yy = x = 1;
  while (b) {
    T q = a / b;
    T t = b; b = a % b; a = t;
    t = xx; xx = x - q * xx; x = t;
    t = yy; yy = y - q * yy; y = t;
  }
  return a;
}
// finds x such that x % m1 = a1, x % m2 = a2. m1 and m2 may
not be coprime
// here, x is unique modulo m = lcm(m1, m2). returns (x, m).
on failure, m = -1.
pair<T, T> CRT(T a1, T m1, T a2, T m2) {
  T p, q;
  T g = extended_euclid(m1, m2, p, q);
  if (a1 % g != a2 % g) return make_pair(0, -1);
  T m = m1 / g * m2;
  p = (p % m + m) % m;
  q = (q % m + m) % m;
  return make_pair((p * a2 % m * (m1 / g) % m + q * a1 % m *
(m2 / g) % m) %  m, m);
}

// +++ sum of floors
// \sum{k=1}^{n}{floor(n/k)}
// count of numbers such that n/i = k -> n/k - n/(k+1)
ll floor_sum(ll n) {
  ll sum = 0;
  for (ll i = 1, last; i <= n; i = last + 1) {
    last = n / (n / i);
    sum += (n / i) * (last - i + 1);
    // n / x yields the same value for i <= x <= last.
  }
  return sum;
}

// +++ dearangement
const int N = 1e6 + 9, mod = 1e9 + 7;
d[0] = 1; d[1] = 0;
for (int i = 1; i < N; i++) {
  d[i] = 1LL * (i - 1) * (d[i - 1] + d[i - 2]) % mod;
}

// +++ Lucas Theorem
const int N = 1e6 + 3, mod = 1e6 + 3;
using ll = long long;

template <const int32_t MOD>
struct modint {
  int32_t value;
  modint() = default;
  modint(int32_t value_) : value(value_) {}
  inline modint<MOD> operator + (modint<MOD> other) const {
int32_t c = this->value + other.value; return modint<MOD>(c
>= MOD ? c - MOD : c); }
  inline modint<MOD> operator - (modint<MOD> other) const {
int32_t c = this->value - other.value; return modint<MOD>(c <
0 ? c + MOD : c); }
  inline modint<MOD> operator * (modint<MOD> other) const {
int32_t c = (int64_t)this->value * other.value % MOD; return
modint<MOD>(c < 0 ? c + MOD : c); }
  inline modint<MOD> & operator += (modint<MOD> other) {
this->value += other.value; if (this->value >= MOD) this-
>value -= MOD; return *this; }
  inline modint<MOD> & operator -= (modint<MOD> other) {
this->value -= other.value; if (this->value <    0) this-
>value += MOD; return *this; }
  inline modint<MOD> & operator *= (modint<MOD> other) {
this->value = (int64_t)this->value * other.value % MOD; if
(this->value < 0) this->value += MOD; return *this; }
  inline modint<MOD> operator - () const { return
modint<MOD>(this->value ? MOD - this->value : 0); }
  modint<MOD> pow(uint64_t k) const { modint<MOD> x = *this,
y = 1; for (; k; k >>= 1) { if (k & 1) y *= x; x *= x; }
return y; }
  modint<MOD> inv() const { return pow(MOD - 2); }  // MOD
must be a prime
  inline modint<MOD> operator /  (modint<MOD> other) const {
return *this *  other.inv(); }
  inline modint<MOD> operator /= (modint<MOD> other)       {
return *this *= other.inv(); }
  inline bool operator == (modint<MOD> other) const { return
value == other.value; }
  inline bool operator != (modint<MOD> other) const { return
value != other.value; }
  inline bool operator < (modint<MOD> other) const { return
value < other.value; }
  inline bool operator > (modint<MOD> other) const { return
value > other.value; }
};
template <int32_t MOD> modint<MOD> operator * (int32_t value,
modint<MOD> n) { return modint<MOD>(value) * n; }
template <int32_t MOD> modint<MOD> operator * (int64_t value,
modint<MOD> n) { return modint<MOD>(value % MOD) * n; }
template <int32_t MOD> istream & operator >> (istream & in,
modint<MOD> &n) { return in >> n.value; }
template <int32_t MOD> ostream & operator << (ostream & out,
modint<MOD> n) { return out << n.value; }

using mint = modint<mod>;

struct combi{
  int n; vector<mint> facts, finvs, invs;
  combi(int _n): n(_n), facts(_n), finvs(_n), invs(_n){
    facts[0] = finvs[0] = 1;
    invs[1] = 1;
    for (int i = 2; i < n; i++) invs[i] =  invs[mod % i] * (-
mod / i);
    for(int i = 1; i < n; i++){
      facts[i] = facts[i - 1] * i;
      finvs[i] = finvs[i - 1] * invs[i];
    }
  }
  inline mint fact(int n) { return facts[n]; }
  inline mint finv(int n) { return finvs[n]; }
  inline mint inv(int n) { return invs[n]; }
  inline mint ncr(int n, int k) { return n < k or k < 0 ? 0 :
facts[n] * finvs[k] * finvs[n-k]; }
};
combi C(N);

// returns nCr modulo mod where mod is a prime
// Complexity: log(n)
mint lucas(ll n, ll r) {
  if (r > n) return 0;
  if (n < mod) return C.ncr(n, r);
  return lucas(n / mod, r / mod) * lucas(n % mod, r % mod);
}

// +++ Stirling Number
```



Recurrence relation [edit]

Stirling numbers of the second kind obey the recurrence relation

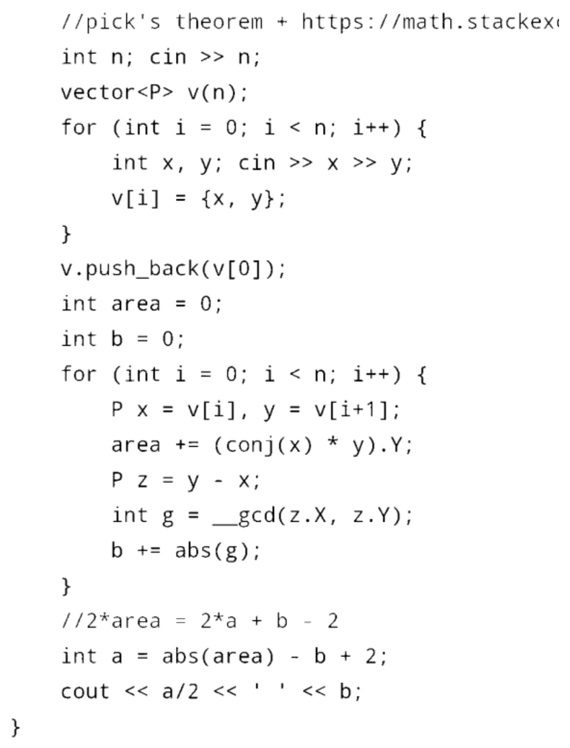
$$\left\{ {n+1 \atop k} \right\} = k \left\{ {n \atop k} \right\} + \left\{ {n \atop k-1} \right\} \quad \text{for } 0 < k < n$$

with initial conditions

$$\left\{ {n \atop n} \right\} = 1 \quad \text{for } n \geq 0 \quad \text{and} \quad \left\{ {n \atop 0} \right\} = \left\{ {0 \atop n} \right\} = 0 \quad \text{for } n > 0.$$

```
// +++ FFT (TWO POLY MUL IN N*LOGN
const int N = 3e5 + 9;
const double PI = acos(-1);
struct base {
  double a, b;
  base(double a = 0, double b = 0) : a(a), b(b) {}
  const base operator + (const base &c) const
    { return base(a + c.a, b + c.b); }
  const base operator - (const base &c) const
    { return base(a - c.a, b - c.b); }
  const base operator * (const base &c) const
    { return base(a * c.a - b * c.b, a * c.b + b * c.a); }
};
void fft(vector<base> &p, bool inv = 0) {
  int n = p.size(), i = 0;
  for(int j = 1; j < n - 1; ++j) {
    for(int k = n >> 1; k > (i ^= k); k >>= 1);
    if(j < i) swap(p[i], p[j]);
  }
  for(int l = 1, m; (m = l << 1) <= n; l <<= 1) {
    double ang = 2 * PI / m;
    base wn = base(cos(ang), (inv ? 1. : -1.) * sin(ang)), w;
    for(int i = 0, j, k; i < n; i += m) {
      for(w = base(1, 0), j = i, k = i + l; j < k; ++j, w = w
* wn) {
        base t = w * p[j + l];
        p[j + l] = p[j] - t;
        p[j] = p[j] + t;
      }
    }
  }
  if(inv) for(int i = 0; i < n; ++i) p[i].a /= n, p[i].b /=
n;
}
vector<long long> multiply(vector<int> &a, vector<int> &b) {
  int n = a.size(), m = b.size(), t = n + m - 1, sz = 1;
  while(sz < t) sz <<= 1;
  vector<base> x(sz), y(sz), z(sz);
  for(int i = 0 ; i < sz; ++i) {
    x[i] = i < (int)a.size() ? base(a[i], 0) : base(0, 0);
    y[i] = i < (int)b.size() ? base(b[i], 0) : base(0, 0);
  }
  fft(x), fft(y);
  for(int i = 0; i < sz; ++i) z[i] = x[i] * y[i];
  fft(z, 1);
  vector<long long> ret(sz);
  for(int i = 0; i < sz; ++i) ret[i] = (long long)
round(z[i].a);
  while((int)ret.size() > 1 && ret.back() == 0)
ret.pop_back();
  return ret;
}

long long ans[N];
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, x; cin >> n >> x;
  vector<int> a(n + 1, 0), b(n + 1, 0), c(n + 1, 0);
  int nw = 0;
  a[0]++; b[n]++;
  long long z = 0;
  for (int i = 1; i <= n; i++) {
    int k; cin >> k;
    nw += k < x;
    a[nw]++; b[-nw + n]++;
    z += c[nw] + !nw; c[nw]++;
  }
  auto res = multiply(a, b);
  for (int i = n + 1; i < res.size(); i++) {
    ans[i - n] += res[i];
  }
  ans[0] = z;
  for (int i = 0; i <= n; i++) cout << ans[i] << ' ';
  cout << '\n';
  return 0;
}

// +++ NTT
const int N = 1 << 20;
const int mod = 998244353;
const int root = 3;
int lim, rev[N], w[N], wn[N], inv_lim;
void reduce(int &x) { x = (x + mod) % mod; }
int POW(int x, int y, int ans = 1) {
  for (; y; y >>= 1, x = (long long) x * x % mod) if (y & 1)
ans = (long long) ans * x % mod;
  return ans;
}
void precompute(int len) {
  lim = wn[0] = 1; int s = -1;
  while (lim < len) lim <<= 1, ++s;
  for (int i = 0; i < lim; ++i) rev[i] = rev[i >> 1] >> 1 |
(i & 1) << s;
  const int g = POW(root, (mod - 1) / lim);
  inv_lim = POW(lim, mod - 2);
  for (int i = 1; i < lim; ++i) wn[i] = (long long) wn[i - 1]
* g % mod;
}
void ntt(vector<int> &a, int typ) {
  for (int i = 0; i < lim; ++i) if (i < rev[i]) swap(a[i],
a[rev[i]]);
  for (int i = 1; i < lim; i <<= 1) {
    for (int j = 0, t = lim / i / 2; j < i; ++j) w[j] = wn[j
* t];
    for (int j = 0; j < lim; j += i << 1) {
      for (int k = 0; k < i; ++k) {
        const int x = a[k + j], y = (long long) a[k + j + i]
* w[k] % mod;
        reduce(a[k + j] += y - mod), reduce(a[k + j + i] = x
- y);
      }
    }
  }
  if (!typ) {
    reverse(a.begin() + 1, a.begin() + lim);
    for (int i = 0; i < lim; ++i) a[i] = (long long) a[i] *
inv_lim % mod;
  }
}
vector<int> multiply(vector<int> &f, vector<int> &g) {
  if (f.empty() or g.empty()) return {};
  int n = (int)f.size() + (int)g.size() - 1;
  if (n == 1) return {(int)((long long) f[0] * g[0] % mod)};
  precompute(n);
  vector<int> a = f, b = g;
  a.resize(lim); b.resize(lim);
  ntt(a, 1), ntt(b, 1);
  for (int i = 0; i < lim; ++i) a[i] = (long long) a[i] *
b[i] % mod;
  ntt(a, 0);
  a.resize(n + 1);
  return a;
}

// +++ Matrix Exponention
struct Mat {
  int n, m;
  vector<vector<int>> a;
  Mat() { }
  Mat(int _n, int _m) {n = _n; m = _m; a.assign(n,
vector<int>(m, 0)); }
  Mat(vector< vector<int> > v) { n = v.size(); m = n ?
v[0].size() : 0; a = v; }
  inline void make_unit() {
    assert(n == m);
    for (int i = 0; i < n; i++)  {
      for (int j = 0; j < n; j++) a[i][j] = i == j;
    }
  }
  inline Mat operator + (const Mat &b) {
    assert(n == b.n && m == b.m);
    Mat ans = Mat(n, m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < m; j++) {
        ans.a[i][j] = (a[i][j] + b.a[i][j]) % mod;
      }
    }
    return ans;
  }
  inline Mat operator - (const Mat &b) {
    assert(n == b.n && m == b.m);
    Mat ans = Mat(n, m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < m; j++) {
        ans.a[i][j] = (a[i][j] - b.a[i][j] + mod) % mod;
      }
    }
    return ans;
  }
  inline Mat operator * (const Mat &b) {
    assert(m == b.n);
    Mat ans = Mat(n, b.m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < b.m; j++) {
        for(int k = 0; k < m; k++) {
          ans.a[i][j] = (ans.a[i][j] + 1LL * a[i][k] *
b.a[k][j] % mod) % mod;
        }
      }
    }
    return ans;
  }
  inline Mat pow(long long k) {
    assert(n == m);
    Mat ans(n, n), t = a; ans.make_unit();
    while (k) {
      if (k & 1) ans = ans * t;
      t = t * t;
      k >>= 1;
    }
    return ans;
  }
  inline Mat& operator += (const Mat& b) { return *this =
(*this) + b; }
  inline Mat& operator -= (const Mat& b) { return *this =
(*this) - b; }
  inline Mat& operator *= (const Mat& b) { return *this =
(*this) * b; }
  inline bool operator == (const Mat& b) { return a == b.a; }
  inline bool operator != (const Mat& b) { return a != b.a; }
};

// +++ Gaussian Elimination
const int N = 3e5 + 9;
const double eps = 1e-9;
int Gauss(vector<vector<double>> a, vector<double> &ans) {
  int n = (int)a.size(), m = (int)a[0].size() - 1;
  vector<int> pos(m, -1);
  double det = 1; int rank = 0;
  for(int col = 0, row = 0; col < m && row < n; ++col) {
    int mx = row;
    for(int i = row; i < n; i++) if(fabs(a[i][col]) >
fabs(a[mx][col])) mx = i;
    if(fabs(a[mx][col]) < eps) {det = 0; continue;}
    for(int i = col; i <= m; i++) swap(a[row][i], a[mx][i]);
    if (row != mx) det = -det;
    det *= a[row][col];
    pos[col] = row;
    for(int i = 0; i < n; i++) {
      if(i != row && fabs(a[i][col]) > eps) {
        double c = a[i][col] / a[row][col];
        for(int j = col; j <= m; j++) a[i][j] -= a[row][j] *
c;
      }
    }
    ++row; ++rank;
  }
  ans.assign(m, 0);
  for(int i = 0; i < m; i++) {
    if(pos[i] != -1) ans[i] = a[pos[i]][m] / a[pos[i]][i];
  }
  for(int i = 0; i < n; i++) {
    double sum = 0;
    for(int j = 0; j < m; j++) sum += ans[j] * a[i][j];
    if(fabs(sum - a[i][m]) > eps) return -1; //no solution
  }
  for(int i = 0; i < m; i++) if(pos[i] == -1) return 2;
//infinte solutions
  return 1; //unique solution
}

// +++ Gaussian Elimination Modular
const int N = 105, mod = 1e9 + 7;
int power(long long n, long long k) {
  int ans = 1 % mod; n %= mod; if (n < 0) n += mod;
  while (k) {
    if (k & 1) ans = (long long) ans * n % mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}
int Gauss(vector<vector<int>> a, vector<int> &ans){
  int n = a.size(), m = (int)a[0].size() - 1;
  vector <int> pos(m, -1);
  int free_var = 0;
  const long long MODSQ = (long long)mod * mod;
  int det = 1, rank = 0;
  for (int col = 0, row = 0; col < m && row < n; col++) {
    int mx = row;
    for (int k = row; k < n; k++) if (a[k][col] > a[mx][col])
mx = k;
    if (a[mx][col] == 0) {det = 0; continue;}
    for (int j = col; j <= m; j++) swap(a[mx][j], a[row][j]);
    if (row != mx) det = det == 0 ? 0 : mod - det;
    det = 1LL * det * a[row][col] % mod;
    pos[col] = row;
    int inv = power(a[row][col], mod - 2);
    for (int i = 0; i < n && inv; i++){
      if (i != row && a[i][col]) {
        int x = ((long long)a[i][col] * inv) % mod;
        for (int j = col; j <= m && x; j++){
          if (a[row][j]) a[i][j] = (MODSQ + a[i][j] - ((long
long)a[row][j] * x)) % mod;
        }
      }
    }
    row++; ++rank;
  }
  ans.assign(m, 0);
  for (int i = 0; i < m; i++){
    if (pos[i] == -1) free_var++;
    else ans[i] = ((long long)a[pos[i]][m] *
power(a[pos[i]][i], mod - 2)) % mod;
  }
  for (int i = 0; i < n; i++) {
    long long val = 0;
    for (int j = 0; j < m; j++) val = (val + ((long
long)ans[j] * a[i][j])) % mod;
    if (val != a[i][m]) return -1; //no solution
  }
  return free_var; //has solution
}

// +++ Permutation and Permutation Cycle
const int N = 3e5 + 9;
// 0-indexed, values are also from 0 to n-1
// decompose into cycles
vector<vector<int>> decompose(vector<int> &p) {
  int n = p.size();
  vector<vector<int>> cycles;
  vector<bool> vis(n, 0);
  for (int i = 0; i < n; i++) {
    if (!vis[i]) {
      vector<int> v;
      while (!vis[i]) {
        v.push_back(i);
        vis[i] = 1;
        i = p[i];
      }
      cycles.push_back(v);
    }
  }
  return cycles;
}

// restore the permutation from the cycles
vector<int> restore(int n, vector<vector<int>> &cycles) {
  vector<int> p(n);
  for (auto v : cycles) {
    int m = v.size();
    for (int i = 0; i < m; i++) p[v[i]] = v[(i + 1) % m];
  }
  return p;
}

//cycle decomposition of the k-th power of p
vector<vector<int>> power(vector<int> &p, int k) {
  int n = p.size();
  auto cycles = decompose(p);
  vector<vector<int>> ans;
  for (auto v : cycles) {
    int len = v.size(), g = __gcd(k, len); // g cycles of len
/ g length
    for (int i = 0; i < g; i++) {
      vector<int> w;
      for (int j = i, cnt = 0; cnt < len / g; cnt++, j = (j +
k) % len) {
        w.push_back(v[j]);
      }
      ans.push_back(w);
    }
  }
  return ans;
}


// +++ Meet in the middle (bit mask when n = 40)
int main() {
        int n, x;
        cin >> n >> x;
        vector<int> a(n);
        for (int i = 0; i < n; i++) { cin >> a[i]; }

        // stores all possible subset sums in the interval
[l, r]
        auto get_subset_sums = [&](int l, int r) ->
vector<ll> {
                int len = r - l + 1;
                vector<ll> res;

                // loop through all subsets
                for (int i = 0; i < (1 << len); i++) {
                        ll sum = 0;
                        for (int j = 0; j < len; j++) {
                                if (i & (1 << j)) { sum +=
a[l + j]; }
                        }
                        res.push_back(sum);
                }

                return res;
        };

        vector<ll> left = get_subset_sums(0, n / 2 - 1);
        vector<ll> right = get_subset_sums(n / 2, n - 1);
        sort(left.begin(), left.end());
        sort(right.begin(), right.end());

        ll ans = 0;
        for (ll i : left) {
                auto low_iterator =
lower_bound(right.begin(), right.end(), x - i);
                auto high_iterator =
upper_bound(right.begin(), right.end(), x - i);
                ans += high_iterator - low_iterator;
        }

        cout << ans << endl;
}

// +++ Gray Code
int g (int n) {
    return n ^ (n >> 1);
}
int rev_g (int g) {
  int n = 0;
  for (; g; g >>= 1)
    n ^= g;
  return n;
}

// +++ Inversion Count
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;

template <typename T> using o_set = tree<T, null_type,
less<T>, rb_tree_tag, tree_order_statistics_node_update>;

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int t; cin >> t;
  while (t--) {
    int n; cin >> n;
    int a[n + 1];
    for (int i = 1; i <= n; i++) {
      cin >> a[i];
    }
    o_set<int> se;
    long long ans = 0;
    for (int i = n; i >= 1; i--) {
      ans += se.order_of_key(a[i]);
      se.insert(a[i]);
    }
    cout << ans << '\n';
  }
  return 0;
}
```

```
// +++ picks theorem
```



```
    //pick's theorem + https://math.stackex
    int n; cin >> n;
    vector<P> v(n);
    for (int i = 0; i < n; i++) {
        int x, y; cin >> x >> y;
        v[i] = {x, y};
    }
    v.push_back(v[0]);
    int area = 0;
    int b = 0;
    for (int i = 0; i < n; i++) {
        P x = v[i], y = v[i+1];
        area += (conj(x) * y).Y;
        P z = y - x;
        int g = __gcd(z.X, z.Y);
        b += abs(g);
    }
    //2*area = 2*a + b - 2
    int a = abs(area) - b + 2;
    cout << a/2 << ' ' << b;
  }
```

```
// +++ Point Line Circle

const int N = 3e5 + 9;

const double inf = 1e100;
const double eps = 1e-9;
const double PI = acos((double)-1.0);
int sign(double x) { return (x > eps) - (x < -eps); }
struct PT {
    double x, y;
    PT() { x = 0, y = 0; }
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y)    {}
    PT operator + (const PT &a) const { return PT(x + a.x, y
+ a.y); }
```

```
    PT operator - (const PT &a) const { return PT(x - a.x, y
- a.y); }
    PT operator * (const double a) const { return PT(x * a, y
* a); }
    friend PT operator * (const double &a, const PT &b) {
return PT(a * b.x, a * b.y); }
    PT operator / (const double a) const { return PT(x / a, y
/ a); }
    bool operator == (PT a) const { return sign(a.x - x) == 0
&& sign(a.y - y) == 0; }
    bool operator != (PT a) const { return !(*this == a); }
    bool operator < (PT a) const { return sign(a.x - x) == 0
? y < a.y : x < a.x; }
    bool operator > (PT a) const { return sign(a.x - x) == 0
? y > a.y : x > a.x; }
    double norm() { return sqrt(x * x + y * y); }
    double norm2() { return x * x + y * y; }
    PT perp() { return PT(-y, x); }
    double arg() { return atan2(y, x); }
    PT truncate(double r) { // returns a vector with norm r
and having same direction
        double k = norm();
        if (!sign(k)) return *this;
        r /= k;
        return PT(x * r, y * r);
    }
};
istream &operator >> (istream &in, PT &p) { return in >> p.x
>> p.y; }
ostream &operator << (ostream &out, PT &p) { return out <<
"(" << p.x << "," << p.y << ")"; }
inline double dot(PT a, PT b) { return a.x * b.x + a.y * b.y;
}
inline double dist2(PT a, PT b) { return dot(a - b, a - b); }
inline double dist(PT a, PT b) { return sqrt(dot(a - b, a -
b)); }
inline double cross(PT a, PT b) { return a.x * b.y - a.y *
b.x; }
inline double cross2(PT a, PT b, PT c) { return cross(b - a,
c - a); }
inline int orientation(PT a, PT b, PT c) { return
sign(cross(b - a, c - a)); }
PT perp(PT a) { return PT(-a.y, a.x); }
PT rotateccw90(PT a) { return PT(-a.y, a.x); }
PT rotatecw90(PT a) { return PT(a.y, -a.x); }
PT rotateccw(PT a, double t) { return PT(a.x * cos(t) - a.y *
sin(t), a.x * sin(t) + a.y * cos(t)); }
PT rotatecw(PT a, double t) { return PT(a.x * cos(t) + a.y *
sin(t), -a.x * sin(t) + a.y * cos(t)); }
double SQ(double x) { return x * x; }
double rad_to_deg(double r) { return (r * 180.0 / PI); }
double deg_to_rad(double d) { return (d * PI / 180.0); }
double get_angle(PT a, PT b) {
    double costheta = dot(a, b) / a.norm() / b.norm();
    return acos(max((double)-1.0, min((double)1.0,
costheta)));
}
bool is_point_in_angle(PT b, PT a, PT c, PT p) { // does
point p lie in angle <bac
    assert(orientation(a, b, c) != 0);
    if (orientation(a, c, b) < 0) swap(b, c);
    return orientation(a, c, p) >= 0 && orientation(a, b, p)
<= 0;
}
bool half(PT p) {
    return p.y > 0.0 || (p.y == 0.0 && p.x < 0.0);
}
void polar_sort(vector<PT> &v) { // sort points in
counterclockwise
    sort(v.begin(), v.end(), [](PT a,PT b) {
        return make_tuple(half(a), 0.0, a.norm2()) <
make_tuple(half(b), cross(a, b), b.norm2());
    });
}
void polar_sort(vector<PT> &v, PT o) { // sort points in
counterclockwise with respect to point o
    sort(v.begin(), v.end(), [&](PT a,PT b) {
        return make_tuple(half(a - o), 0.0, (a - o).norm2())
< make_tuple(half(b - o), cross(a - o, b - o), (b -
o).norm2());
    });
}
struct line {
    PT a, b; // goes through points a and b
    PT v; double c;  //line form: direction vec [cross] (x,
y) = c
    line() {}
    //direction vector v and offset c
        line(PT v, double c) : v(v), c(c) {
        auto p = get_points();
        a = p.first; b = p.second;
        }
        // equation ax + by + c = 0
        line(double _a, double _b, double _c) : v({_b, -_a}),
c(-_c) {
                auto p = get_points();
        a = p.first; b = p.second;
        }
        // goes through points p and q
        line(PT p, PT q) : v(q - p), c(cross(v, p)), a(p),
b(q) {}
        pair<PT, PT> get_points() { //extract any two points
from this line
                PT p, q; double a = -v.y, b = v.x; // ax + by
= c
                if (sign(a) == 0) {
                    p = PT(0, c / b);
                    q = PT(1, c / b);
                }
                else if (sign(b) == 0) {
                    p = PT(c / a, 0);
                    q = PT(c / a, 1);
                }
                else {
                    p = PT(0, c / b);
                    q = PT(1, (c - a) / b);
                }
                return {p, q};
        }
    // ax + by + c = 0
    array<double, 3> get_abc() {
        double a = -v.y, b = v.x;
        return {a, b, -c};
    }
    // 1 if on the left, -1 if on the right, 0 if on the line
    int side(PT p) { return sign(cross(v, p) - c); }
    // line that is perpendicular to this and goes through
point p
    line perpendicular_through(PT p) { return {p, p +
perp(v)}; }
    // translate the line by vector t i.e. shifting it by
vector t
    line translate(PT t) { return {v, c + cross(v, t)}; }
    // compare two points by their orthogonal projection on
this line
    // a projection point comes before another if it comes
first according to vector v
    bool cmp_by_projection(PT p, PT q) { return dot(v, p) <
dot(v, q); }
        line shift_left(double d) {
                PT z = v.perp().truncate(d);
                return line(a + z, b + z);
        }
};
// find a point from a through b with distance d
PT point_along_line(PT a, PT b, double d) {
    assert(a != b);
    return a + (((b - a) / (b - a).norm()) * d);
}
// projection point c onto line through a and b  assuming a
!= b
PT project_from_point_to_line(PT a, PT b, PT c) {
    return a + (b - a) * dot(c - a, b - a) / (b - a).norm2();
}
// reflection point c onto line through a and b  assuming a
!= b
PT reflection_from_point_to_line(PT a, PT b, PT c) {
    PT p = project_from_point_to_line(a,b,c);
    return p + p - c;
}
// minimum distance from point c to line through a and b
double dist_from_point_to_line(PT a, PT b, PT c) {
    return fabs(cross(b - a, c - a) / (b - a).norm());
}
// returns true if  point p is on line segment ab
bool is_point_on_seg(PT a, PT b, PT p) {
    if (fabs(cross(p - b, a - b)) < eps) {
        if (p.x < min(a.x, b.x) - eps || p.x > max(a.x, b.x)
+ eps) return false;
        if (p.y < min(a.y, b.y) - eps || p.y > max(a.y, b.y)
+ eps) return false;
        return true;
    }
    return false;
}
// minimum distance point from point c to segment ab that
lies on segment ab
PT project_from_point_to_seg(PT a, PT b, PT c) {
    double r = dist2(a, b);
    if (sign(r) == 0) return a;
    r = dot(c - a, b - a) / r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b - a) * r;
}
// minimum distance from point c to segment ab
double dist_from_point_to_seg(PT a, PT b, PT c) {
    return dist(c, project_from_point_to_seg(a, b, c));
}
// 0 if not parallel, 1 if parallel, 2 if collinear
int is_parallel(PT a, PT b, PT c, PT d) {
    double k = fabs(cross(b - a, d - c));
    if (k < eps){
        if (fabs(cross(a - b, a - c)) < eps && fabs(cross(c -
d, c - a)) < eps) return 2;
        else return 1;
    }
    else return 0;
}
// check if two lines are same
bool are_lines_same(PT a, PT b, PT c, PT d) {
    if (fabs(cross(a - c, c - d)) < eps && fabs(cross(b - c,
c - d)) < eps) return true;
    return false;
}
// bisector vector of <abc
PT angle_bisector(PT &a, PT &b, PT &c){
    PT p = a - b, q = c - b;
    return p + q * sqrt(dot(p, p) / dot(q, q));
}
// 1 if point is ccw to the line, 2 if point is cw to the
line, 3 if point is on the line
int point_line_relation(PT a, PT b, PT p) {
    int c = sign(cross(p - a, b - a));
    if (c < 0) return 1;
    if (c > 0) return 2;
    return 3;
}
// intersection point between ab and cd assuming unique
intersection exists
bool line_line_intersection(PT a, PT b, PT c, PT d, PT &ans)
{
    double a1 = a.y - b.y, b1 = b.x - a.x, c1 = cross(a, b);
    double a2 = c.y - d.y, b2 = d.x - c.x, c2 = cross(c, d);
    double det = a1 * b2 - a2 * b1;
    if (det == 0) return 0;
    ans = PT((b1 * c2 - b2 * c1) / det, (c1 * a2 - a1 * c2) /
det);
    return 1;
}
// intersection point between segment ab and segment cd
assuming unique intersection exists
bool seg_seg_intersection(PT a, PT b, PT c, PT d, PT &ans) {
    double oa = cross2(c, d, a), ob = cross2(c, d, b);
    double oc = cross2(a, b, c), od = cross2(a, b, d);
    if (oa * ob < 0 && oc * od < 0){
        ans = (a * ob - b * oa) / (ob - oa);
        return 1;
    }
    else return 0;
}
// intersection point between segment ab and segment cd
assuming unique intersection may not exists
// se.size()==0 means no intersection
// se.size()==1 means one intersection
// se.size()==2 means range intersection
set<PT> seg_seg_intersection_inside(PT a,  PT b,  PT c,  PT
d) {
    PT ans;
    if (seg_seg_intersection(a, b, c, d, ans)) return {ans};
    set<PT> se;
    if (is_point_on_seg(c, d, a)) se.insert(a);
    if (is_point_on_seg(c, d, b)) se.insert(b);
    if (is_point_on_seg(a, b, c)) se.insert(c);
    if (is_point_on_seg(a, b, d)) se.insert(d);
    return se;
}
// intersection  between segment ab and line cd
// 0 if do not intersect, 1 if proper intersect, 2 if segment
intersect
int seg_line_relation(PT a, PT b, PT c, PT d) {
    double p = cross2(c, d, a);
    double q = cross2(c, d, b);
    if (sign(p) == 0 && sign(q) == 0) return 2;
    else if (p * q < 0) return 1;
    else return 0;
}
// intersection between segament ab and line cd assuming
unique intersection exists
bool seg_line_intersection(PT a, PT b, PT c, PT d, PT &ans) {
    bool k = seg_line_relation(a, b, c, d);
    assert(k != 2);
    if (k) line_line_intersection(a, b, c, d, ans);
    return k;
}
// minimum distance from segment ab to segment cd
double dist_from_seg_to_seg(PT a, PT b, PT c, PT d) {
    PT dummy;
    if (seg_seg_intersection(a, b, c, d, dummy)) return 0.0;
    else return min({dist_from_point_to_seg(a, b, c),
dist_from_point_to_seg(a, b, d),
        dist_from_point_to_seg(c, d, a),
dist_from_point_to_seg(c, d, b)});
}
// minimum distance from point c to ray (starting point a and
direction vector b)
double dist_from_point_to_ray(PT a, PT b, PT c) {
    b = a + b;
    double r = dot(c - a, b - a);
    if (r < 0.0) return dist(c, a);
    return dist_from_point_to_line(a, b, c);
}
// starting point as and direction vector ad
bool ray_ray_intersection(PT as, PT ad, PT bs, PT bd) {
    double dx = bs.x - as.x, dy = bs.y - as.y;
    double det = bd.x * ad.y - bd.y * ad.x;
    if (fabs(det) < eps) return 0;
    double u = (dy * bd.x - dx * bd.y) / det;
    double v = (dy * ad.x - dx * ad.y) / det;
    if (sign(u) >= 0 && sign(v) >= 0) return 1;
    else return 0;
}
double ray_ray_distance(PT as, PT ad, PT bs, PT bd) {
    if (ray_ray_intersection(as, ad, bs, bd)) return 0.0;
    double ans = dist_from_point_to_ray(as, ad, bs);
    ans = min(ans, dist_from_point_to_ray(bs, bd, as));
    return ans;
}
struct circle {
    PT p; double r;
    circle() {}
    circle(PT _p, double _r): p(_p), r(_r) {};
    // center (x, y) and radius r
    circle(double x, double y, double _r): p(PT(x, y)), r(_r)
{};
    // circumcircle of a triangle
    // the three points must be unique
    circle(PT a, PT b, PT c) {
        b = (a + b) * 0.5;
        c = (a + c) * 0.5;
        line_line_intersection(b, b + rotatecw90(a - b), c, c
+ rotatecw90(a - c), p);
        r = dist(a, p);
    }
    // inscribed circle of a triangle
    // pass a bool just to differentiate from circumcircle
    circle(PT a, PT b, PT c, bool t) {
        line u, v;
        double m = atan2(b.y - a.y, b.x - a.x), n = atan2(c.y
- a.y, c.x - a.x);
        u.a = a;
        u.b = u.a + (PT(cos((n + m)/2.0), sin((n + m)/2.0)));
        v.a = b;
        m = atan2(a.y - b.y, a.x - b.x), n = atan2(c.y - b.y,
c.x - b.x);
        v.b = v.a + (PT(cos((n + m)/2.0), sin((n + m)/2.0)));
        line_line_intersection(u.a, u.b, v.a, v.b, p);
        r = dist_from_point_to_seg(a, b, p);
    }
    bool operator == (circle v) { return p == v.p && sign(r -
v.r) == 0; }
    double area() { return PI * r * r; }
    double circumference() { return 2.0 * PI * r; }
};
//0 if outside, 1 if on circumference, 2 if inside circle
int circle_point_relation(PT p, double r, PT b) {
    double d = dist(p, b);
    if (sign(d - r) < 0) return 2;
    if (sign(d - r) == 0) return 1;
    return 0;
}
// 0 if outside, 1 if on circumference, 2 if inside circle
int circle_line_relation(PT p, double r, PT a, PT b) {
    double d = dist_from_point_to_line(a, b, p);
    if (sign(d - r) < 0) return 2;
    if (sign(d - r) == 0) return 1;
    return 0;
}
//compute intersection of line through points a and b with
//circle centered at c with radius r > 0
vector<PT> circle_line_intersection(PT c, double r, PT a, PT
b) {
    vector<PT> ret;
    b = b - a; a = a - c;
    double A = dot(b, b), B = dot(a, b);
    double C = dot(a, a) - r * r, D = B * B - A * C;
    if (D < -eps) return ret;
    ret.push_back(c + a + b * (-B + sqrt(D + eps)) / A);
    if (D > eps) ret.push_back(c + a + b * (-B - sqrt(D)) /
A);
    return ret;
}
//5 - outside and do not intersect
//4 - intersect outside in one point
//3 - intersect in 2 points
//2 - intersect inside in one point
//1 - inside and do not intersect
int circle_circle_relation(PT a, double r, PT b, double R) {
    double d = dist(a, b);
    if (sign(d - r - R) > 0)  return 5;
    if (sign(d - r - R) == 0) return 4;
    double l = fabs(r - R);
    if (sign(d - r - R) < 0 && sign(d - l) > 0) return 3;
    if (sign(d - l) == 0) return 2;
    if (sign(d - l) < 0) return 1;
    assert(0); return -1;
}
vector<PT> circle_circle_intersection(PT a, double r, PT b,
double R) {
    if (a == b && sign(r - R) == 0) return {PT(1e18, 1e18)};
    vector<PT> ret;
    double d = sqrt(dist2(a,  b));
    if (d > r + R || d + min(r,  R) < max(r,  R)) return ret;
    double x = (d * d - R * R + r * r) / (2 * d);
    double y = sqrt(r * r - x * x);
    PT v = (b - a) / d;
    ret.push_back(a + v * x  +  rotateccw90(v) * y);
    if (y > 0) ret.push_back(a + v * x - rotateccw90(v) * y);
    return ret;
}
// returns two circle c1, c2 through points a, b and of
radius r
// 0 if there is no such circle, 1 if one circle, 2 if two
circle
int get_circle(PT a, PT b, double r, circle &c1, circle &c2)
{
    vector<PT> v = circle_circle_intersection(a, r, b, r);
    int t = v.size();
    if (!t) return 0;
    c1.p = v[0], c1.r = r;
    if (t == 2) c2.p = v[1], c2.r = r;
    return t;
}
// returns two circle c1, c2 which is tangent to line u,
goes through
// point q and has radius r1; 0 for no circle, 1 if c1 = c2 ,
2 if c1 != c2
int get_circle(line u, PT q, double r1, circle &c1, circle
&c2) {
    double d = dist_from_point_to_line(u.a, u.b, q);
    if (sign(d - r1 * 2.0) > 0) return 0;
    if (sign(d) == 0) {
        cout << u.v.x << ' ' << u.v.y << '\n';
        c1.p = q + rotateccw90(u.v).truncate(r1);
        c2.p = q + rotatecw90(u.v).truncate(r1);
        c1.r = c2.r = r1;
        return 2;
    }
    line u1 = line(u.a + rotateccw90(u.v).truncate(r1), u.b +
rotateccw90(u.v).truncate(r1));
    line u2 = line(u.a + rotatecw90(u.v).truncate(r1), u.b +
rotatecw90(u.v).truncate(r1));
    circle cc = circle(q, r1);
    PT p1, p2; vector<PT> v;
    v = circle_line_intersection(q, r1, u1.a, u1.b);
    if (!v.size()) v = circle_line_intersection(q, r1, u2.a,
u2.b);
    v.push_back(v[0]);
    p1 = v[0], p2 = v[1];
    c1 = circle(p1, r1);
    if (p1 == p2) {
        c2 = c1;
        return 1;
    }
    c2 = circle(p2, r1);
    return 2;
}
// returns the circle such that for all points w on the
circumference of the circle
// dist(w, a) : dist(w, b) = rp : rq
// rp != rq
// https://en.wikipedia.org/wiki/Circles_of_Apollonius
circle get_apollonius_circle(PT p, PT q, double rp, double rq
){
    rq *= rq ;
    rp *= rp ;
    double a = rq - rp ;
    assert(sign(a));
    double g = rq * p.x - rp * q.x ; g /= a ;
    double h = rq * p.y - rp * q.y ; h /= a ;
    double c = rq * p.x * p.x - rp * q.x * q.x + rq * p.y *
p.y - rp * q.y * q.y ;
    c /= a ;
    PT o(g, h);
    double r = g * g + h * h - c ;
    r = sqrt(r);
    return circle(o,r);
}
// returns area of intersection between two circles
double circle_circle_area(PT a, double r1, PT b, double r2) {
    double d = (a - b).norm();
    if(r1 + r2 < d + eps) return 0;
    if(r1 + d < r2 + eps) return PI * r1 * r1;
    if(r2 + d < r1 + eps) return PI * r2 * r2;
    double theta_1 = acos((r1 * r1 + d * d - r2 * r2) / (2 *
r1 * d)),
        theta_2 = acos((r2 * r2 + d * d - r1 * r1)/(2 * r2 *
d));
    return r1 * r1 * (theta_1 - sin(2 * theta_1)/2.) + r2 *
r2 * (theta_2 - sin(2 * theta_2)/2.);
}
// tangent lines from point q to the circle
int tangent_lines_from_point(PT p, double r, PT q, line &u,
line &v) {
    int x = sign(dist2(p, q) - r * r);
    if (x < 0) return 0; // point in cricle
    if (x == 0) { // point on circle
        u = line(q, q + rotateccw90(q - p));
        v = u;
        return 1;
    }
    double d = dist(p, q);
    double l = r * r / d;
    double h = sqrt(r * r - l * l);
    u = line(q, p + ((q - p).truncate(l) + (rotateccw90(q -
p).truncate(h))));
    v = line(q, p + ((q - p).truncate(l) + (rotatecw90(q -
p).truncate(h))));
```

```
    return 2;
}
// returns outer tangents line of two circles
// if inner == 1 it returns inner tangent lines
int tangents_lines_from_circle(PT c1, double r1, PT c2,
double r2, bool inner, line &u, line &v) {
    if (inner) r2 = -r2;
    PT d = c2 - c1;
    double dr = r1 - r2, d2 = d.norm2(), h2 = d2 - dr * dr;
    if (d2 == 0 || h2 < 0) {
        assert(h2 != 0);
        return 0;
    }
    vector<pair<PT, PT>>out;
    for (int tmp: {- 1, 1}) {
        PT v = (d * dr + rotateccw90(d) * sqrt(h2) * tmp) /
d2;
        out.push_back({c1 + v * r1, c2 + v * r2});
    }
    u = line(out[0].first, out[0].second);
    if (out.size() == 2) v = line(out[1].first,
out[1].second);
    return 1 + (h2 > 0);
}

// +++ convex hull
vector<PT> convex_hull(vector<PT> &p) {
        if (p.size() <= 1) return p;
        vector<PT> v = p;
    sort(v.begin(), v.end());
    vector<PT> up, dn;
    for (auto& p : v) {
        while (up.size() > 1 && orientation(up[up.size() -
2], up.back(), p) >= 0) {
            up.pop_back();
        }
        while (dn.size() > 1 && orientation(dn[dn.size() -
2], dn.back(), p) <= 0) {
            dn.pop_back();
        }
        up.push_back(p);
        dn.push_back(p);
    }
    v = dn;
    if (v.size() > 1) v.pop_back();
    reverse(up.begin(), up.end());
    up.pop_back();
    for (auto& p : up) {
        v.push_back(p);
    }
    if (v.size() == 2 && v[0] == v[1]) v.pop_back();
    return v;
}
 //checks if convex or not
bool is_convex(vector<PT> &p) {
    bool s[3]; s[0] = s[1] = s[2] = 0;
    int n = p.size();
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        int k = (j + 1) % n;
        s[sign(cross(p[j] - p[i], p[k] - p[i])) + 1] = 1;
        if (s[0] && s[2]) return 0;
    }
    return 1;
}
// -1 if strictly inside, 0 if on the polygon, 1 if strictly
outside
// it must be strictly convex, otherwise make it strictly
convex first
int is_point_in_convex(vector<PT> &p, const PT& x) { // O(log
n)
    int n = p.size(); assert(n >= 3);
    int a = orientation(p[0], p[1], x), b = orientation(p[0],
p[n - 1], x);
    if (a < 0 || b > 0) return 1;
    int l = 1, r = n - 1;
    while (l + 1 < r) {
        int mid = l + r >> 1;
        if (orientation(p[0], p[mid], x) >= 0) l = mid;
        else r = mid;
    }
    int k = orientation(p[l], p[r], x);
    if (k <= 0) return -k;
    if (l == 1 && a == 0) return 0;
    if (r == n - 1 && b == 0) return 0;
    return -1;
}

// +++ closest pair of points nlogn
#include<bits/stdc++.h>
using namespace std;

const int N = 3e5 + 9;
#define x first
#define y second
long long dist2(pair<int, int> a, pair<int, int> b) {
 return 1LL * (a.x - b.x) * (a.x - b.x) + 1LL * (a.y - b.y) *
(a.y - b.y);
}
pair<int, int> closest_pair(vector<pair<int, int>> a) {
 int n = a.size();
 assert(n >= 2);
 vector<pair<pair<int, int>, int>> p(n);
 for (int i = 0; i < n; i++) p[i] = {a[i], i};
 sort(p.begin(), p.end());
 int l = 0, r = 2;
 long long ans = dist2(p[0].x, p[1].x);
 pair<int, int> ret = {p[0].y, p[1].y};
 while (r < n) {
  while (l < r && 1LL * (p[r].x.x - p[l].x.x) * (p[r].x.x -
p[l].x.x) >= ans) l++;
  for (int i = l; i < r; i++) {
   long long nw = dist2(p[i].x, p[r].x);
   if (nw < ans) {
    ans = nw;
    ret = {p[i].y, p[r].y};
   }
  }
  r++;
 }
 return ret;
}
int32_t main() {
 ios_base::sync_with_stdio(0);
 cin.tie(0);
 int n; cin >> n;
 vector<pair<int, int>> p(n);
 for (int i = 0; i < n; i++) cin >> p[i].x >> p[i].y;
 pair<int, int> z = closest_pair(p);
 if (z.x > z.y) swap(z.x, z.y);
 cout << z.x << ' ' << z.y << ' ' << fixed << setprecision(6)
<< sqrtl(dist2(p[z.x], p[z.y])) << '\n';
  return 0;
}

// +++ Geometry 3d
// https://victorlecomte.com/cp-geo.pdf
const double inf = 1e100;
const double eps = 1e-9;
const double PI = acos((double)-1.0);
int sign(double x) { return (x > eps) - (x < -eps); }

struct PT {
    double x, y;
    PT() { x = 0, y = 0; }
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y)    {}
    void scan() { cin >> x >> y; }
    PT operator + (const PT &a) const { return PT(x + a.x, y
+ a.y); }
    PT operator - (const PT &a) const { return PT(x - a.x, y
- a.y); }
    PT operator * (const double a) const { return PT(x * a, y
* a); }
    friend PT operator * (const double &a, const PT &b) {
return PT(a * b.x, a * b.y); }
    PT operator / (const double a) const { return PT(x / a, y
/ a); }
    bool operator == (PT a) const { return sign(a.x - x) == 0
&& sign(a.y - y) == 0; }
    bool operator != (PT a) const { return !(*this == a); }
    bool operator < (PT a) const { return sign(a.x - x) == 0
? y < a.y : x < a.x; }
    bool operator > (PT a) const { return sign(a.x - x) == 0
? y > a.y : x > a.x; }
    double norm() { return sqrt(x * x + y * y); }
    double norm2() { return x * x + y * y; }
    PT perp() { return PT(-y, x); }
    double arg() { return atan2(y, x); }
    PT truncate(double r) { // returns a vector with norm r
and having same direction
        double k = norm();
        if (!sign(k)) return *this;
        r /= k;
        return PT(x * r, y * r);
    }
};
inline double dot(PT a, PT b) { return a.x * b.x + a.y * b.y;
}
inline double dist2(PT a, PT b) { return dot(a - b, a - b); }
inline double dist(PT a, PT b) { return sqrt(dot(a - b, a -
b)); }
inline double cross(PT a, PT b) { return a.x * b.y - a.y *
b.x; }
inline int orientation(PT a, PT b, PT c) { return
sign(cross(b - a, c - a)); }
PT perp(PT a) { return PT(-a.y, a.x); }
PT rotateccw90(PT a) { return PT(-a.y, a.x); }
PT rotatecw90(PT a) { return PT(a.y, -a.x); }
PT rotateccw(PT a, double t) { return PT(a.x * cos(t) - a.y *
sin(t), a.x * sin(t) + a.y * cos(t)); }
PT rotatecw(PT a, double t) { return PT(a.x * cos(t) + a.y *
sin(t), -a.x * sin(t) + a.y * cos(t)); }
double SQ(double x) { return x * x; }
double rad_to_deg(double r) { return (r * 180.0 / PI); }
double deg_to_rad(double d) { return (d * PI / 180.0); }
double get_angle(PT a, PT b) {
    double costheta = dot(a, b) / a.norm() / b.norm();
    return acos(max((double)-1.0, min((double)1.0,
costheta)));
}
struct p3 {
    double x, y, z;
    p3() { x = 0, y = 0; z = 0; }
    p3(double x, double y, double z) : x(x), y(y), z(z) {}
    p3(const p3 &p) : x(p.x), y(p.y), z(p.z)    {}
    void scan() { cin >> x >> y >> z; }
    p3 operator + (const p3 &a) const { return p3(x + a.x, y
+ a.y, z + a.z); }
    p3 operator - (const p3 &a) const { return p3(x - a.x, y
- a.y, z - a.z); }
    p3 operator * (const double a) const { return p3(x * a, y
* a, z * a); }
    friend p3 operator * (const double &a, const p3 &b) {
return p3(a * b.x, a * b.y, a * b.z); }
    p3 operator / (const double a) const { return p3(x / a, y
/ a, z / a); }
    bool operator == (p3 a) const { return sign(a.x - x) == 0
&& sign(a.y - y) == 0 && sign(a.z - z) == 0; }
    bool operator != (p3 a) const { return !(*this == a); }
    double abs() { return sqrt(x * x + y * y + z * z); }
    double sq() { return x * x + y * y + z * z; }
    p3 unit() { return *this / abs(); }
}zero(0, 0, 0);
double operator | (p3 v, p3 w) { //dot product
    return v.x * w.x + v.y * w.y + v.z * w.z;
}
p3 operator * (p3 v, p3 w) { //cross product
    return {v.y * w.z - v.z * w.y, v.z * w.x - v.x * w.z, v.x
* w.y - v.y * w.x};
}
double sq(p3 v) { return v | v; }
double abs(p3 v) { return sqrt(sq(v)); }
p3 unit(p3 v) { return v / abs(v); }
inline double dot(p3 a, p3 b) { return a.x * b.x + a.y * b.y
+ a.z * b.z; }
inline double dist2(p3 a, p3 b) { return dot(a - b, a - b); }
inline double dist(p3 a, p3 b) { return sqrt(dot(a - b, a -
b)); }
inline p3 cross(p3 a, p3 b) { return p3(a.y * b.z - a.z *
b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x); }
// if s is on the same side of the plane pqr as the vector pq
* pr then it will be positive
// otherwise negative or 0 if on the plane
double orient(p3 p, p3 q, p3 r, p3 s) { return (q - p) * (r -
p) | (s - p); }
// returns orientation of p to q to r on the plane
perpendicular to n
// assuming p, q, r are on the plane
double orient_by_normal(p3 p, p3 q, p3 r, p3 n) { return (q -
p) * (r - p) | n; }
double get_angle(p3 a, p3 b) {
    double costheta = dot(a, b) / a.abs() / b.abs();
    return acos(max((double)-1.0, min((double)1.0,
costheta)));
}
double small_angle(p3 v, p3 w) {
    return acos(min(fabs(v | w) / abs(v) / abs(w),
(double)1.0));
}

// +++ 3d line
struct line3d {
    // d is the direction vector of the line
    p3 d, o; // p = o + k * d (k is a real parameter)
    line3d() {}
    // From two points P, Q
    line3d(p3 p, p3 q) : d(q - p), o(p) {}
    // From two planes p1, p2
    // assuming they are not parallel
    line3d(plane p1, plane p2) {
        d = p1.n * p2.n;
        o = (p2.n * p1.d - p1.n * p2.d) * d / sq(d);
        // o is actually the closest point on the line to the
origin
    }
    double dist2(p3 p) { return sq(d * (p - o)) / sq(d); }
    double dist(p3 p) { return sqrt(dist2(p)); }
    // compare points by their projection on the line
    // so you can sort points on the line using this
    bool cmp_proj(p3 p, p3 q) { return (d | p) < (d | q); }
    // orthogonal projection of point p onto line
    p3 proj(p3 p) { return o + d * (d|(p - o)) / sq(d); }
    // orthogonal reflection of point p onto line
    p3 refl(p3 p) { return proj(p) * 2 - p; }
    // returns the intersection point of the line with plane
p
    // assuming plane and line are not parallel
    p3 inter(plane p) {
        // assert((d | p.n) != 0); // no intersection if
parallel
        return o - d * p.side(o) / (d | p.n);
    }
};

// smallest distance between two lines
double dist(line3d l1, line3d l2) {
    p3 n = l1.d * l2.d;
    if (n == zero) return l1.dist(l2.o); // parallel
    return fabs((l2.o - l1.o) | n) / abs(n);
}
// closest point from line l2 to line l1
p3 closest_on_l1(line3d l1, line3d l2) {
    p3 n2 = l2.d * (l1.d * l2.d);
    return l1.o + l1.d * ((l2.o - l1.o) | n2) / (l1.d | n2);
}
// small angle between direction vectors of two lines
double get_angle(line3d l1, line3d l2) {
    return small_angle(l1.d, l2.d);
}
bool is_parallel(line3d l1, line3d l2) {
    return l1.d * l2.d == zero;
}
bool is_perpendicular(line3d l1, line3d l2) {
    return sign((l1.d | l2.d)) == 0;
}




// ### JONAYED

// +++ FAST IO
ios_base::sync_with_stdio(false);
cin.tie(NULL);
cout.tie(NULL);

// +++ binexp
long long binexp(long long a, long long n)
{
  long long ans = 1;
  while(n > 0)
  {
    if(n & 1)
      ans = (ans * a) % MOD;
    a = (a * a) % MOD;
    n >>= 1;
  }
  return ans;
}

// +++ binary mul
long long binmul(long long a, long long b)
{
  long long ans = 1;
  while(b > 0)
  {
    if(b & 1)
      ans = (ans + a) % MOD;
    a = (a + a) % MOD;
    b >>= 1;
  }
  return ans;
}

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

int n, m;
vector<pair<int, int>> g[N], r[N];
vector<long long> dijkstra(int s, int t, vector<int> &cnt)
{
    const long long inf = 1e18;
    priority_queue<pair<long long, int>, vector<pair<long
long, int>>, greater<pair<long long, int>>> q;
    vector<long long> d(n + 1, inf);
    vector<bool> vis(n + 1, 0);
    q.push({0, s});
    d[s] = 0;
    cnt.resize(n + 1, 0); // number of shortest paths
    cnt[s] = 1;
    while (!q.empty())
    {
        auto x = q.top();
        q.pop();
        int u = x.second;
        if (vis[u])
            continue;
        vis[u] = 1;
        for (auto y : g[u])
        {
            int v = y.first;
            long long w = y.second;
            if (d[u] + w < d[v])
            {
                d[v] = d[u] + w;
                q.push({d[v], v});
                cnt[v] = cnt[u];
            }
            else if (d[u] + w == d[v])
                cnt[v] = (cnt[v] + cnt[u]) % mod;
        }
    }
    return d;
}

// +++ digit dp
long long a, b, k;
long long dp[90][2][2][90][90]; // n (digits covered till
now), f (is less than b), s (maximum sum), r (current
remainder)
vector<int> digitsOfUpperLimit, digitsOfLowerLimit;

long long solve(long long n, int isSmall, int isBig, long
long sum, long long remainder)
{
    if(n >= digitsOfUpperLimit.size()) return sum % k == 0 &&
remainder % k == 0;
    if(dp[n][isSmall][isBig][sum][remainder] != -1) return
dp[n][isSmall][isBig][sum][remainder];

    long long cur = 0;
    long long nthDigitOfUpperLimit = (isSmall? 9 :
digitsOfUpperLimit[n]);
    long long nthDigitOfLowerLimit = (isBig?   0 :
digitsOfLowerLimit[n]);
    for (int d = nthDigitOfLowerLimit; d <=
nthDigitOfUpperLimit; d++)
    {
        int newIsBig = isBig, newIsSmall = isSmall;
        if(d < nthDigitOfUpperLimit && isSmall == 0)
newIsSmall = 1;
        if(d > nthDigitOfLowerLimit && isBig   == 0) newIsBig
= 1;
        cur += solve(n + 1, newIsSmall, newIsBig, (sum + d) %
k, (remainder * 10 + d) % k);
    }

    return dp[n][isSmall][isBig][sum][remainder] = cur;
}

// +++ kmp
const int N = 3e5 + 9;
// returns the longest proper prefix array of pattern p
// where lps[i]=longest proper prefix which is also suffix of
p[0...i]
vector<int> build_lps(string p)
{
    int sz = p.size();
    vector<int> lps;
    lps.assign(sz + 1, 0);
    int j = 0;
    lps[0] = 0;
    for (int i = 1; i < sz; i++)
    {
        while (j >= 0 && p[i] != p[j])
        {
            if (j >= 1)
                j = lps[j - 1];
            else
                j = -1;
        }
        j++;
        lps[i] = j;
    }
    return lps;
}
vector<int> ans;

// returns matches in vector ans in 0-indexed
void kmp(vector<int> lps, string s, string p)
{
    int psz = p.size(), sz = s.size();
    int j = 0;
    for (int i = 0; i < sz; i++)
    {
        while (j >= 0 && p[j] != s[i])
            if (j >= 1)
                j = lps[j - 1];
            else
                j = -1;
        j++;
        if (j == psz)
        {
            j = lps[j - 1];
            // pattern found in string s at position i-psz+1
            ans.push_back(i - psz + 1);
        }
        // after each loop we have j=longest common suffix of
s[0..i] which is also prefix of p
    }
}

int main()
{
    int i, j, k, n, m, t;
    cin >> t;
    while (t--)
    {
        string s, p;
        cin >> s >> p;
        vector<int> lps = build_lps(p);
        kmp(lps, s, p);
        if (ans.empty())
            cout << "Not Found\n";
        else
        {
            cout << ans.size() << endl;
            for (auto x : ans)
                cout << x << ' ';
            cout << endl;
        }
        ans.clear();
        cout << endl;
    }
    return 0;
}

// +++ Total Hamiltonian paths
void findAllPaths(int N, vector<vector<int>> &graph)
{
  int dp[N][(1 << N)];
  memset(dp, 0, sizeof(dp));
  dp[0][1] = 1; // Initialize for the first vertex

  for (int i = 2; i < (1 << N); i++)
  // Iterate over all the masks
  {

    if ((i & (1 << 0)) == 0) // If the first vertex is absent
      continue;
    // Only consider the full subsets
    if ((i & (1 << (N - 1))) && i != ((1 << N) - 1))
      continue;

    for (int end = 0; end < N; end++) // Choose the end city
    {
      // If this city is not in the subset
      if (i & (1 << end) == 0)
        continue;
      int prev = i - (1 << end); // Set without the end city
      // Check for the adjacent cities
      for (int it : graph[end])
        if ((i & (1 << it)))
          dp[end][i] += dp[it][prev];
    }
  }

  cout << dp[N - 1][(1 << N) - 1];
}

// +++ String trie
const int MAXN = 260005;
struct arr {
    int a[26];
    arr() {}
    void clear() { memset(a,-1,sizeof(a)); }
    int& operator[](int i) { return a[i]; }
};
struct trie {
    int cnt, prefix_cnt[MAXN], word_cnt[MAXN];
    arr to[MAXN];

    trie() { cnt = MAXN-1; }

    void clear() { for(int i = 0; i < cnt; i++) prefix_cnt[i]
= word_cnt[i] = 0, to[i].clear(); cnt = 1; }

    void add(const string& s) {
        int u = 0;
        for(const char& c: s)  {
            if(to[u][c-'a'] == -1) to[u][c-'a'] = cnt++;
            u = to[u][c-'a'];
            prefix_cnt[u]++;
        }
        word_cnt[u]++;
    }

    int prefix_count(const string& s) {
        int u = 0;
        for (const char& c: s) {
            if (to[u][c-'a'] == -1) return 0;
            u = to[u][c-'a'];
        }
        return prefix_cnt[u];
    }
} tr;
// +++ Hashing
const int N = 1e6 + 9;
const int MOD1 = 127657753, MOD2 = 987654319;
const int p1 = 137, p2 = 277;
int ip1, ip2;
pair<int, int> pw[N], ipw[N];
void prec()
{
    pw[0] = {1, 1};
    for (int i = 1; i < N; i++)
    {
        pw[i].first = 1LL * pw[i - 1].first * p1 % MOD1;
        pw[i].second = 1LL * pw[i - 1].second * p2 % MOD2;
    }
    ip1 = power(p1, MOD1 - 2, MOD1);
    ip2 = power(p2, MOD2 - 2, MOD2);
    ipw[0] = {1, 1};
    for (int i = 1; i < N; i++)
    {
        ipw[i].first = 1LL * ipw[i - 1].first * ip1 % MOD1;
        ipw[i].second = 1LL * ipw[i - 1].second * ip2 % MOD2;
    }
}

struct Hashing
{
    int n;
    string s;                  // 0 - indexed
    vector<pair<int, int>> hs; // 1 - indexed
    Hashing() {}
    Hashing(string _s)
    {
        n = _s.size();
        s = _s;
        hs.emplace_back(0, 0);
        for (int i = 0; i < n; i++)
        {
            pair<int, int> p;
            p.first = (hs[i].first + 1LL * pw[i].first * s[i]
% MOD1) % MOD1;
            p.second = (hs[i].second + 1LL * pw[i].second *
s[i] % MOD2) % MOD2;
            hs.push_back(p);
        }
    }
    // Suffix Hash
    pair<int, int> get_hash(int l, int r)
    { // 1 - indexed
        assert(1 <= l && l <= r && r <= n);
        pair<int, int> ans;
        ans.first = (hs[r].first - hs[l - 1].first + MOD1) *
1LL * ipw[l - 1].first % MOD1;
        ans.second = (hs[r].second - hs[l - 1].second + MOD2)
* 1LL * ipw[l - 1].second % MOD2;
        return ans;
    }
    // Whole Hash
    pair<int, int> get_hash()
    {
        return get_hash(1, n);
    }
};




// ### SHOVON
// +++ Bit
template <class T>
struct BIT { //1-indexed
  int n; vector<T> t;
  BIT() {}
  void init(int _n) {
    n = _n; t.assign(n + 1, 0);
  }
  T query(int i) {
    T ans = 0;
    for (; i >= 1; i -= (i & -i)) ans += t[i];
    return ans;
  }
  void upd(int i, T val) {
    if (i <= 0) return;
    for (; i <= n; i += (i & -i)) t[i] += val;
  }
  void upd(int l, int r, T val) {
    upd(l, val);
    upd(r + 1, -val);
  }
  T query(int l, int r) {
    return query(r) - query(l - 1);
  }
    ll bit_search(ll v){//lower bound
        ll sum=0;
        ll pos=0;
        ll N=bit.size();
        for(ll i=log2(N)+1;i>=0;i--){
            if(pos+(1<<i)<N && sum+bit[pos+(1<<i)]<v){
                sum+=bit[pos+(1<<i)];
                pos+=(1<<i);
            }
        }
        return pos+1;
    }
};

// +++ Cordinate Compress
int n = a.size();
vector<pair<int, int>> pairs(n);
for(int i = 0; i < n; ++i) {
   pairs[i] = {a[i], i};
}
sort(pairs.begin(), pairs.end());
int nxt = 1;
for(int i = 0; i < n; ++i) {
   if(i > 0 && pairs[i-1].first != pairs[i].first) nxt++;
   a[pairs[i].second] = nxt;
}

// +++ DSU
class DSU {
private:
    vector<ll> parent, size, minElement, maxElement;
    ll n;

public:
    DSU(ll nn) {
        this->n = nn+1;
        parent.resize(n + 1);
        size.resize(n + 1, 1);  // Each element is its own
set initially, so size is 1
        minElement.resize(n + 1);
        maxElement.resize(n + 1);

        for (ll i = 1; i <= n; ++i) {
            parent[i] = i;
            minElement[i] = i;
            maxElement[i] = i;
        }
    }

    ll find(ll u) {
        if (parent[u] != u) {
            parent[u] = find(parent[u]); // Path Compression
        }
        return parent[u];
    }

    void unite(ll u, ll v) {
        ll rootU = find(u);
        ll rootV = find(v);

        if (rootU != rootV) {
            // Union by Size
            if (size[rootU] < size[rootV]) {
                swap(rootU, rootV);  // Ensure rootU is
always the larger set
            }

            parent[rootV] = rootU;  // Merge the smaller set
into the larger set
            size[rootU] += size[rootV];
            minElement[rootU] = min(minElement[rootU],
minElement[rootV]);
            maxElement[rootU] = max(maxElement[rootU],
maxElement[rootV]);
        }
    }

    tuple<int, int, int> get(int v) {
        int rootV = find(v);
        return {minElement[rootV], maxElement[rootV],
size[rootV]};
    }
};

// +++ GAME
vll dp(10005, -1);
ll k;

ll calcGrundy(ll n){
    if(n<=0){
        return 0;
    }
    else if(dp[n]!=-1){
        return dp[n];
    }
    vll v;
    fl(i,k){
        if(n-s[i]>=0){
            v.pb(calcGrundy(n-s[i]));
        }
    }
    return dp[n]=mex2(v);
}

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

const int N = 1e5 + 9, LG = 18, inf = 1e9 + 9;

struct ST {
#define lc (n << 1)
#define rc ((n << 1) | 1)
  int t[4 * N], lazy[4 * N];
  ST() {
    fill(t, t + 4 * N, -inf);
    fill(lazy, lazy + 4 * N, 0);
  }
  inline void push(int n, int b, int e) {
    if(lazy[n] == 0) return;
    t[n] = t[n] + lazy[n];
    if(b != e) {
      lazy[lc] = lazy[lc] + lazy[n];
      lazy[rc] = lazy[rc] + lazy[n];
    }
    lazy[n] = 0;
  }
  inline int combine(int a, int b) {
    return max(a, b); //merge left and right queries
  }
  inline void pull(int n) {
    t[n] = max(t[lc], t[rc]); //merge lower nodes of the tree
to get the parent node
  }
  void build(int n, int b, int e) {
    if(b == e) {
      t[n] = 0;
      return;
    }
    int mid = (b + e) >> 1;
    build(lc, b, mid);
    build(rc, mid + 1, e);
    pull(n);

  }
  void upd(int n, int b, int e, int i, int j, int v) {
    push(n, b, e);
    if(j < b || e < i) return;
    if(i <= b && e <= j) {
      lazy[n] += v;
      push(n, b, e);
      return;
    }
    int mid = (b + e) >> 1;
    upd(lc, b, mid, i, j, v);
    upd(rc, mid + 1, e, i, j, v);
    pull(n);
  }
  int query(int n, int b, int e, int i, int j) {
    push(n, b, e);
    if(i > e || b > j) return -inf;
    if(i <= b && e <= j) return t[n];
    int mid = (b + e) >> 1;
    return combine(query(lc, b, mid, i, j), query(rc, mid +
1, e, i, j));
  }
} t;

vector<int> g[N];
int par[N][LG + 1], dep[N], sz[N];
void dfs(int u, int p = 0) {
  par[u][0] = p;
  dep[u] = dep[p] + 1;
  sz[u] = 1;
  for (int i = 1; i <= LG; i++) par[u][i] = par[par[u][i -
1]][i - 1];
  if (p) g[u].erase(find(g[u].begin(), g[u].end(), p));
  for (auto &v : g[u]) if (v != p) {
      dfs(v, u);
      sz[u] += sz[v];
      if(sz[v] > sz[g[u][0]]) swap(v, g[u][0]);
    }
}
int lca(int u, int v) {
  if (dep[u] < dep[v]) swap(u, v);
  for (int k = LG; k >= 0; k--) if (dep[par[u][k]] >= dep[v])
u = par[u][k];
  if (u == v) return u;
  for (int k = LG; k >= 0; k--) if (par[u][k] != par[v][k]) u
= par[u][k], v = par[v][k];
  return par[u][0];
}
int kth(int u, int k) {
  assert(k >= 0);
  for (int i = 0; i <= LG; i++) if (k & (1 << i)) u =
par[u][i];
  return u;
}
int T, head[N], st[N], en[N];
void dfs_hld(int u) {
  st[u] = ++T;
  for (auto v : g[u]) {
    head[v] = (v == g[u][0] ? head[u] : v);
    dfs_hld(v);
  }
  en[u] = T;
}
int n;
int query_up(int u, int v) {
  int ans = -inf;
  while(head[u] != head[v]) {
    ans = max(ans, t.query(1, 1, n, st[head[u]], st[u]));
    u = par[head[u]][0];
  }
  ans = max(ans, t.query(1, 1, n, st[v], st[u]));
  return ans;
}
int query(int u, int v) {
  int l = lca(u, v);
  int ans = query_up(u, l);
  if (v != l) ans = max(ans, query_up(v, kth(v, dep[v] -
dep[l] - 1)));
  return ans;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  cin >> n;
  for (int i = 1; i < n; i++) {
    int u, v;
    cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
  }
  dfs(1);
  head[1] = 1;
  dfs_hld(1);
  int q;
  cin >> q;
  t.build(1, 1, n);
  while (q--) {
    string ty;
    int u, v;
    cin >> ty >> u >> v;
    if (ty == "add") {
      t.upd(1, 1, n, st[u], en[u], v);
    } else {
      cout << query(u, v) << '\n';
    }
  }
  return 0;
}

// +++ LAZY SEGMENT TREE
mt19937 rng;
int getRandomNumber(int l, int r)
{
    uniform_int_distribution<int> dist(l, r);
    return dist(rng);
}

void preSolve()
{
    rng =
mt19937(chrono::steady_clock::now().time_since_epoch().count(
));
    // allocateStackMax();
}

struct seg_tree
{
    ll size;
    vector<ll> ops, vals;

    ll NEUTRAL_ELEMENT = 0,NO_OPERATION = LLONG_MAX-1;

    ll modify_op(ll a, ll b, ll len){
        if(b == NO_OPERATION) return a;
        // debug(b);
        return b;
    }

    ll calc_op(ll a, ll b){
        return a|b;
    }

    void apply_op(ll &a, ll b, ll len){
        a = modify_op(a,b,len);
    }

    void init(ll n)
    {
        size = 1;
        while (size < n)
            size *= 2;
        ops.assign(2 * size, 0LL);
        vals.assign(2 * size, 0LL);

        // build(0,0,size);
    }

    void propagate(ll x, ll lx, ll rx){
        if(rx-lx == 1){
            return;
        }
        ll m=(rx+lx)/2;

        apply_op(ops[2*x+1],ops[x],1);
        apply_op(vals[2*x+1],ops[x],m-lx);
        apply_op(ops[2*x+2],ops[x],1);
        apply_op(vals[2*x+2],ops[x],rx-m);
        ops[x] = NO_OPERATION;
    }

    void modify(ll l, ll r, ll v, ll x, ll lx, ll rx)
    {
        propagate(x,lx,rx);
        if (lx >= r or l >= rx)
        {
            return;
        }
        if (lx >= l and rx <= r)
        {
            ll val = (1LL<<v);
            apply_op(vals[x],val,rx-lx);
            apply_op(ops[x],val,1);// this for range update
            return;
        }

        ll m = (lx + rx) / 2;
        modify(l, r, v, 2 * x + 1, lx, m);
        modify(l, r, v, 2 * x + 2, m, rx);
        vals[x] = calc_op(vals[2 * x + 1], vals[2 * x + 2]);

    }

    void modify(ll l, ll r, ll v)
    {
        modify(l, r, v, 0, 0, size);
    }

    ll calc(ll l, ll r, ll x, ll lx, ll rx){
        propagate(x,lx,rx);
        if(lx >= r or l >= rx){
            return NEUTRAL_ELEMENT;
        }
        if(lx >= l and rx <= r){
            return vals[x];
        }

        ll m = (lx+rx)/2;
        ll m1= calc(l,r,2*x+1,lx,m);
        ll m2 = calc(l,r,2*x+2,m,rx);
        auto res = calc_op(m1,m2);

        return res;
    }

    ll calc(ll l, ll r){
        return calc(l,r,0,0,size);
    }
};

// +++ MO ALGO
inline void remove(idx);  // TODO: remove value at idx from
data structure
inline void add(idx);     // TODO: add value at idx from data
structure
inline int get_answer();  // TODO: extract the current answer
of the data structure

int block_size;

struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the
range [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}

// +++ Random Hash
random_device rd;
mt19937 gen(rd());
map<ull, ull> mapping;
set<ull> s= {0};
for(auto i: v){
    ull random;
    do{
        random = gen();
    }while(s.find(random)!=s.end());
    mapping[i] = random;
    s.insert(random);
}

// +++ Segment Tree
template <typename T>
struct SegmentTree
{
    int n;
    vector<T> tree;
    T neutral_value;           // Neutral value for the
operation (e.g., 0 for sum, INT_MAX for min)
    function<T(T, T)> combine; // Function to combine two
segments

    void init(int size, T neutral, function<T(T, T)> comb)
    {
        n = 1;
        neutral_value = neutral;
        combine = comb;
        while (n < size)
            n *= 2;
        tree.assign(2 * n, neutral_value);
    }

    void build(const vector<T> &a, ll x, ll lx, ll rx)
    {
        if (rx - lx == 1)
        {
            if (lx < (ll)a.size())
            {
                tree[x] = a[lx];
                // debug(lx,a[lx].mn,a[lx].mx,a[lx].inc);
            }
            return;
        }
        ll m = (lx + rx) / 2;
        build(a, 2 * x + 1, lx, m);
        build(a, 2 * x + 2, m, rx);
        tree[x] = combine(tree[2 * x + 1], tree[2 * x + 2]);
    }

    T query(ll l, ll r, ll x, ll lx, ll rx)
    {
        if (lx >= r || l >= rx)
            return neutral_value;
        if (lx >= l && rx <= r)
            return tree[x];
        ll m = (lx + rx) / 2;
        T s1 = query(l, r, 2 * x + 1, lx, m);
        T s2 = query(l, r, 2 * x + 2, m, rx);
        return combine(s1, s2);
    }

    void update(ll i, T v, ll x, ll lx, ll rx)
    {
        if (rx - lx == 1)
        {
            tree[x] = v;
            return;
        }
        ll m = (lx + rx) / 2;
        if (i < m)
        {
            update(i, v, 2 * x + 1, lx, m);
        }
        else
        {
            update(i, v, 2 * x + 2, m, rx);
        }
        tree[x] = combine(tree[2 * x + 1], tree[2 * x + 2]);
    }

    // Helper functions to simplify calling
    void build(const vector<T> &arr)
    {
        build(arr, 0, 0, n);
    }

    T query(int l, int r)
    {
        return query(l, r, 0, 0, n);
    }

    void update(int pos, T new_val)
    {
        update(pos, new_val, 0, 0, n);
    }
};
//SegmentTree<int> segTree(arr.size(), 0, [](int a, int b) {
return a + b; });

// +++ Trie
class Trie {

    class Node {
        Node* link[26];
        int cntEndswith = 0, cntPref = 0;

    public:
        Node() {
            for (int i = 0; i < 26; i++) {
                link[i] = NULL;
            }
        }

        bool containsch(char c) {
            return link[c - 'a'] != NULL;
        }

        Node* getNext(char c) {
            return link[c - 'a'];
        }

        void putchar(char c) {
            link[c - 'a'] = new Node();
        }

        void setEnd() {
            cntEndswith++;
        }

        void increaseCntPref() {
            cntPref++;
        }

        void decreaseCntPref() {
            cntPref--;
        }

        void decEnd() {
            cntEndswith--;
        }

        int getCntWord() {
            return cntEndswith;
        }

        int getCntPref() {
            return cntPref;
        }
    };

    Node* root;

public:

    Trie() {
        root = new Node();
    }

    void insert(string &word) {
        Node* node = root;
        for (auto c : word) {
            if (!node->containsch(c)) {
                node->putchar(c);
            }
             // Only increment prefix count during insertion
            node = node->getNext(c);
            node->increaseCntPref();
        }
        node->setEnd();  // Increment end count to mark word
insertion
    }

    int countWordsEqualTo(string &word) {
        Node* node = root;
        for (auto c : word) {
            if (!node->containsch(c)) {
                return 0;  // Word doesn't exist in Trie
            }
            node = node->getNext(c);
        }
        return node->getCntWord();  // Return how many times
the word was inserted
    }

    int countWordsStartingWith(string &prefix) {
        Node* node = root;
        for (auto c : prefix) {
            if (!node->containsch(c)) {
                return 0;  // Prefix doesn't exist in Trie
            }
            node = node->getNext(c);
        }
        return node->getCntPref();  // Return how many words
share this prefix
    }

    void erase(string &word) {
        Node* node = root;
        for (auto c : word) {
            if (!node->containsch(c)) {
                return;  // Word doesn't exist, can't erase
            }
              // Decrease prefix count along the path
            node = node->getNext(c);
            node->decreaseCntPref();
        }
        node->decEnd();  // Decrease end count to mark the
word as erased
    }
};

// +++ Trieb
class Trie{
    class Node{
        Node* link[2];
        public:
        Node(){ link[0]=link[1]=NULL; }
        bool contains(ll v){ return link[v]!=NULL; }
        void put(ll v){ link[v]=new Node(); }
        Node* getNext(ll v){ return link[v]; }
    };

    public:
    Node* root;
    Trie(){ root=new Node(); }
    void insert(ll num){
        Node* node=root;

        for(ll i=31;i>=0;i--){
            ll bit=(num>>i)&1;
            if(!node->contains(bit)){
                node->put(bit);
            }
            node=node->getNext(bit);
        }
    }
    ll getMax(ll num){
        Node* node=root;
        ll ans=0;
        for(ll i=31;i>=0;i--){
            ll bit=(num>>i)&1;
            if(node->contains(1-bit)){
                ans|=(1<<i);
                node=node->getNext(1-bit);
            }
            else if(node->contains(bit)){
                node=node->getNext(bit);
            }
        }
        return ans;
    }
};

// +++ inversion
ll merge_and_count(vector<ll>& arr, ll left, ll mid, ll
right) {
    ll n1 = mid - left + 1;
    ll n2 = right - mid;

    // Create temp arrays
    vector<ll> left_arr(n1);
    vector<ll> right_arr(n2);

    // Copy data to temp arrays left_arr[] and right_arr[]
    for (ll i = 0; i < n1; i++)
        left_arr[i] = arr[left + i];
    for (ll i = 0; i < n2; i++)
        right_arr[i] = arr[mid + 1 + i];

    ll i = 0; // Initial index of the first subarray
    ll j = 0; // Initial index of the second subarray
    ll k = left; // Initial index of the merged subarray
    ll inv_count = 0;

    // Merge the temp arrays back into arr[l..r]
    while (i < n1 && j < n2) {
        if (left_arr[i] <= right_arr[j]) {
            arr[k++] = left_arr[i++];
        } else {
            arr[k++] = right_arr[j++];
            inv_count += (n1 - i); // Count inversions
        }
    }

    // Copy the remaining elements of left_arr[], if any
    while (i < n1)
        arr[k++] = left_arr[i++];

    // Copy the remaining elements of right_arr[], if any
    while (j < n2)
        arr[k++] = right_arr[j++];

    return inv_count;
}

// Function to use merge sort and count inversions
ll merge_sort_and_count(vector<ll>& arr, ll left, ll right) {
    ll inv_count = 0;
    if (left < right) {
        ll mid = left + (right - left) / 2;

        inv_count += merge_sort_and_count(arr, left, mid);
        inv_count += merge_sort_and_count(arr, mid + 1,
right);
        inv_count += merge_and_count(arr, left, mid, right);
    }
    return inv_count;
}

// +++ kmp
vll kmp(string s){
    ll n=s.size();

    vll lps(n,0);

    fls(i,1,n-1){
        ll j=lps[i-1];
        while(j>0 && s[i]!=s[j]){
            j=lps[j-1];
        }
        if(s[i]==s[j]){
            j++;
        }

        lps[i]=j;
    }
    return lps;
}

// +++ Pref sum formula
So, for 1D prefix sums you simply do S[i + 1] = S[i] + a[i],
nothing special here. For 2D, on the other side, you have to
subtract overlapping region: S[i][j] = S[i - 1][j] + S[i][j -
1] - S[i - 1][j - 1] + a[i][j]. For 3D there will be even
more overlapping: S[i][j][k] = S[i - 1][j][k] + S[i][j -
1][k] + S[i][j][k - 1] - S[i - 1][j - 1][k] - S[i][j - 1][k -
1] - S[i - 1][j][k - 1] + S[i - 1][j - 1][k - 1] + a[i][j][k]

// +++ sqrt
int len = (int) sqrt (n + .0) + 1; // size of the block and
the number of blocks
vector<int> b (len);
for (int i=0; i<n; ++i)
    b[i / len] += a[i];

// answering the queries
for (;;) {
    int l, r;
  // read input data for the next query
    int sum = 0;
    for (int i=l; i<=r; )
        if (i % len == 0 && i + len - 1 <= r) {
            // if the whole block starting at i belongs to
[l, r]
            sum += b[i / len];
            i += len;
        }
        else {
            sum += a[i];
            ++i;
        }
}

// +++ ternary
// Ternary search on integers to find maximum
int ternarySearch(int l, int r) {
    while (r - l > 2) {
        int m1 = l + (r - l) / 3;
        int m2 = r - (r - l) / 3;
        if (f(m1) < f(m2))
        //for minimum use >= instead of <
            l = m1;
        else
            r = m2;
    }
    // Brute force through the remaining 2 or 3 elements
    int maxVal = f(l);
    for (int i = l + 1; i <= r; i++) {
        maxVal = max(maxVal, f(i));
    }
    return maxVal;
}

// Ternary search on doubles to find maximum
double ternarySearch(double l, double r) {
    const double eps = 1e-9;  // Precision threshold
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        if (f(m1) < f(m2))
        //for minimum use >= instead of <
            l = m1;
        else
            r = m2;
    }
    return (l + r) / 2;  // Midpoint will give the maximum
value
}

// +++ z-function
vll z_func(string s)
{
    ll n = s.length();
    vll z(n);
    for (ll i = 1, l = 0, r = 0; i < n; i++)
    {
        if (i < r)
        {
            z[i] = min(r - i, z[i - l]);
        }
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
        {
            z[i]++;
        }
        if (i + z[i] > r)
        {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}
```

1 Multiplicative Function

Definition 1.1 (Multiplicative Function). *A number-theoretic function f is* ***multiplicative*** *if $f(1) = 1$ and $f(mn) = f(m)f(n)$, $\forall m, n \in \mathbb{N}$ such that $gcd(m, n) = 1$. Additionally, f is called* ***completely multiplicative*** *if $f(mn) = f(m)f(n)$, $\forall m, n \in \mathbb{N}$.*

Examples.

- $1(n) = 1$: The constant function.
- $Id(n) = n$: The identity function.
- $\varepsilon(n) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } n > 1 \end{cases}$: The unit function
- $\tau(n) = \sum\limits_{d|n} 1$: The number of divisors function.
- $\sigma(n) = \sum\limits_{d|n} d$: The sum of divisors function.
- $\phi(n) = \sum\limits_{i=1}^{n} [gcd(i, n) = 1]$: Euler's Totient Function (here the third brackets serve as a boolean function, which returns 1 if the condition is true, 0 otherwise.)

Note that $1, Id, \varepsilon$ are completely multiplicative as well, while ϕ, τ, σ aren't.

Theorem 1. *If f is a multiplicative function and if $n = \prod\limits_{i=1}^{r} {p_i}^{e_i}$, then $f(n) = \prod\limits_{i=1}^{r} f({p_i}^{e_i})$.*

Proof. Since $gcd({p_i}^{e_i}, {p_j}^{e_j}) = 1$, $\forall i \neq j$, induction on r proves the theorem. □

2 Dirichlet Convolution

Definition 2.1 (Dirichlet Convolution). *If f and g are two arithmetic functions, then their* ***dirichlet convolution****, denoted by $f * g$, is defined as*

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

Alternatively, we can write $(f * g)(n) = \sum\limits_{ab=n} f(a)g(b)$.

Properties of Dirichlet Convolution

- Convolution is Commutative: $f * g = g * f$
- It is Associative: $(f * g) * h = f * (g * h)$

  Proof. $((f * g) * h)(n) = \sum\limits_{dc=n} (f * g)(d)h(c) = \sum\limits_{dc=n} \sum\limits_{ab=d} f(a)g(b)h(c) = \sum\limits_{abc=n} f(a)g(b)h(c)$

  Similarly, $(f * (g * h))(n) = \sum\limits_{ad=n} f(a)(g * h)(d) = \sum\limits_{ad=n} f(a) \sum\limits_{bc=d} g(b)h(c) = \sum\limits_{abc=n} f(a)g(b)h(c)$ □

- $f * \varepsilon = f$

  Proof.
  $$(f * \varepsilon)(n) = \sum_{d|n} f(d)\varepsilon(n/d)$$
  Now if $d < n$, $n/d > 1$, so $\varepsilon(n/d) = 0$. Therefore
  $$(f * \varepsilon)(n) = f(n)\varepsilon(1) = f(n)$$
  □

Theorem 2. *if f and g are both multiplicative, so is $f * g$.*

Proof. Let $h = f * g$. Now, if $gcd(m, n) = 1$,
$$h(mn) = \sum_{d|mn} f(d)g(mn/d)$$
Now since $d|mn$ and $gcd(m, n) = 1$, $d = ab$ where $a|m, b|n$ and $gcd(a, b) = 1$. Hence,
$$\begin{aligned} h(mn) &= \sum_{a|m,b|n} f(ab)g(mn/ab) \\ &= \sum_{a|m,b|n} f(a)f(b)g(m/a)g(n/b) \\ &= \sum_{a|m} f(a)g(m/a) \sum_{b|n} f(b)g(n/b) \\ &= h(m)h(n) \end{aligned}$$
□

Corollary 1. *if f is a multiplicative function, and $F(n) = \sum_{d|n} f(n)$, F is multiplicative as well.*

Proof. $F(n) = \sum\limits_{d|n} f(n) = \sum\limits_{d|n} f(n)1(\frac{n}{d}) = (f * 1)(n)$.
Since $f, 1$ both are multiplicative, by **theorem 2**, F is also multiplicative. □

Theorem 3. *if $h = f * g$ and h, g are both multiplicative, so is f.*

Proof. Suppose f is not multiplicative. So, there exists a pair of positive integers (m, n) with $gcd(m, n) = 1$ such that $f(mn) \neq f(m)f(n)$. We take such a pair with smallest mn.

If $mn = 1$ then $f(1) \neq f(1)f(1)$ which implies $f(1) \neq 1$. Now since g is multiplicative, $g(1) = 1$. But $h(1) = (f*g)(1) = f(1)g(1) \neq 1$ since $f(1) \neq 1$. This is a contradiction since h is multiplicative and $h(1) = 1$.

If $mn > 1$ then

$$\begin{aligned} h(mn) &= \sum_{d|mn} f(d)g(mn/d) \\ &= \sum_{a|m,b|n} f(ab)g(mn/ab) \\ &= \sum_{a|m,b|n,ab<mn} f(a)f(b)g(m/a)g(n/b) + f(mn)g(1) \\ &= \sum_{a|m,b|n} f(a)f(b)g(m/a)g(n/b) - f(m)f(n) + f(mn) \\ &= \sum_{a|m} f(a)g(m/a) \sum_{b|n} f(b)g(n/b) - f(m)f(n) + f(mn) \\ &= h(m)h(n) - f(m)f(n) + f(mn) \end{aligned}$$

Now if h is multiplicative, $f(m)f(n) = f(mn)$, which implies f is multiplicative as well. $\square$

Definition 2.2 (Dirichlet Inverse). *If f is an arithmetic function, we call g* ***dirichlet inverse*** *of f, denoted by f^{-1}, if $f * g = \varepsilon$.*

Recall that ε is the unit function: $\varepsilon(n) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } n > 1 \end{cases}$

Theorem 4. *If f is an arithmetic function where $f(1) \neq 0$, then f^{-1} exists.*

Proof. Let $g = \begin{cases} \frac{1}{f(1)} & \text{if } n = 1 \\ -\frac{\sum_{d|n,d<n} f(n/d)g(d)}{f(1)} & \text{if } n > 1 \end{cases}$
Clearly, $(f * g)(1) = 1$. Now for $n > 1$,

$$\begin{aligned} (f * g)(n) &= (g * f)(n) \\ &= \sum_{d|n} g(d)f(n/d) \\ &= g(n)f(1) + \sum_{d|n,d<n} g(d)f(n/d) \end{aligned}$$

Now,

$$g(n) = -\frac{\sum_{d|n,d<n} g(d)f(n/d)}{f(1)} \implies -g(n)f(1) = \sum_{d|n,d<n} g(d)f(n/d)$$

So, $(f * g)(n) = g(n)f(1) - g(n)f(1) = 0$. Hence, $(f * g) = \varepsilon$, thus $g = f^{-1}$. $\square$

Corollary 2. *If f is multiplicative, so is f^{-1}*

Proof. Since $\varepsilon = f * f^{-1}$, and both ε and f ar multiplicative, by **theorem 3**, f^{-1} is also multiplicative. $\square$

3 More on common multiplicative functions

3.1 Number-of-Divisors Function

Let $\tau(n)$ denote the number of divisors of n. So, $\tau(n) = \sum_{d|n} 1$. It is easy to see that τ is multiplicative, since $\tau(n) = \sum_{d|n} 1 = \sum_{d|n} 1(d)1(n/d) = (1 * 1)(n)$. Recall $1(n)$ is the constant function, which is completely multiplicative. Hence, by **theorem 2**, τ is also multiplicative.
Now it is easy to derive the formula of this function. What is $\tau(p^x)$ where p is prime? Of course the divisors of p^x are $1, p, p^2, \ldots, p^x$, so total (x + 1) divisors. Now for any $n = \prod_{i=1}^{r} {p_i}^{e_i}$, we can simply apply **theorem 1** and get $\tau(n) = \prod_{i=1}^{r}(e_i + 1)$. $\tau(n)$ can be calculated with a simple loop in $O(\sqrt{n})$.

3.2 Sum-of-Divisors Function

The sum of divisors function is denoted by σ. We can write $\sigma(n) = \sum_{d|n} d$. Similar to the previous function, we can write $\sigma(n) = \sum_{d|n} d = \sum_{d|n} Id(d)1(n/d) = (Id * 1)(n)$. Since Id and 1 are both multiplicative, by **theorem 2**, σ is also multiplicative.
Now, note that for a prime p, $\sigma(p^x) = 1 + p + p^2 + \cdots + p^x = \frac{p^{x+1}-1}{p-1}$. So, for $n = \prod_{i=1}^{r} {p_i}^{e_i}$, $\sigma(n) = \prod_{i=1}^{r} \frac{{p_i}^{e_i+1}-1}{p_i-1}$.

3.3 Möbius Function

Definition 3.1. *For a positive integer n, the **möbius function**, denoted by μ, is defined as.*

The next theorem shows a very important property of möbius function.

Theorem 5. $\sum_{d|n} \mu(d) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } n > 1 \end{cases} = \varepsilon(n)$

Proof. For $n = 1$, $\sum_{d|n} \mu(d) = \mu(1) = 1$
When $n > 1$, let $n = \prod_{i=1}^{r} {p_i}^{e_i}$. If $d|n$, then $d = \prod_{i=1}^{r} {p_i}^{f_i}$, where $f_i \le e_i$, for $1 \le i \le r$. Now if $\exists i$ such that $f_i > 1$, $\mu(d)$ will be 0. So, the only divisors we care about are the ones with $f_i \le 1, \forall i$. This means we have r distinct primes, and we will go through each possible combinations and add their contribution to the sum accordingly. For example, if d contains k primes, then we have $\binom{r}{k}$

possible values for d, and each of them will contribute $(-1)^k$ to the sum. So,

$$\begin{aligned}\sum_{d|n}\mu(d) &= \sum_{\substack{(f_1,f_2,\ldots,f_r)\\ f_i\in\{0,1\}}} \mu(\prod_{i=1}^{r} {p_i}^{f_i}) \\ &= (-1)^0\binom{r}{0} + (-1)^1\binom{r}{1} + \cdots + (-1)^r\binom{r}{r} \\ &= \sum_{i=0}^{r}\binom{r}{i}(-1)^i \\ &= (1-1)^r \\ &= 0\end{aligned}$$

□

We also conclude that μ is **multiplicative** from this theorem, since,

$$\sum_{d|n}\mu(d) = \sum_{d|n}\mu(d)1(n/d) = (\mu * 1)(n) = \varepsilon(n)$$

This implies μ is the dirichlet inverse of the constant function. So, by **Corollary 2**, μ is multiplicative. Since μ is the dirichlet inverse of the constant function, using the definition of g in **theorem 4**, we can write that,

$$\mu(n) = \sum_{d|n, d<n} -\mu(d)$$

Now we can use this in sieve to calculate the value of möbius function for integers from 1 to n. Here's an implementation in C++:

```cpp
mu[1] = 1;
for(int i = 1; i < N; i++) {
    for(int j = 2 * i; j < N; j+=i) {
        mu[j] -= mu[i];
    }
}
```

This is basically the standard sieve: instead of iterating over d for each n, for each d, we loop over its multiples and add its contribution to the answer of the multiples. So, it runs in $O(n\log n)$ time.

3.4 Euler's Totient Function

Definition 3.2. *For a positive integer n, **Euler's Totient Function**, denotes as $\phi(n)$, is defined as the number of positive integers i up to n such that $gcd(i, n) = 1$.*

For example, $\phi(4) = 2$, since from 1 to 4, only 1 and 3 are coprimes with 4.

Theorem 6. $\sum_{d|n}\phi(d) = n$

Proof. Let S_d be the set of all positive integers up to n whose gcd with n is d. Now, $gcd(n, m) = d$ implies $gcd(n/d, m/d) = 1$. By definition, the number of such m is $\phi(n/d)$. So,

$$\sum_{d|n}|S_d| = \sum_{d|n}\phi(n/d)$$

Clearly, the sets S_d for all d are pairwise disjoint. Moreover, for every integer $1 \le k \le n$, there exists a d such that $d|n$ and $k \in S_d$. Hence, $\sum_{d|n} |S_d| = \sum_{d|n} \phi(n/d) = \sum_{d|n} \phi(d) = n$. $\square$

Since, $\sum_{d|n} \phi(d) = \sum_{d|n} \phi(d)1(n/d) = (\phi * 1)(n) = Id(n)$, we also see that ϕ is a multiplicative function.
Note that if p is prime, $\phi(p) = p - 1$, and $\phi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1)$. So, if $n = \prod_{i=1}^{r} {p_i}^{e_i}$, by **theorem 1**, we get

$$\phi(n) = \prod_{i=1}^{r} {p_i}^{e_i - 1}(p_i - 1) = \prod_{i=1}^{r} {p_i}^{e_i}\frac{(p_i - 1)}{p_i} = n\prod_{i=1}^{r}(1 - \frac{1}{p_i})$$

which is the well-known formula for calculating $\phi(n)$.
About implementation, we can simply initialize $\phi(n) = n$, and for each prime divisor p, we update its value by multiplying it with $(1 - \frac{1}{p}) = \frac{p-1}{p}$. Here's the code:

```
for (int i = 1; i <= n; i++) phi[i] = i;
for(int p = 2; p <= n; p++) {
  if (phi[p] == p) {
    phi[p] = p - 1;
    for (int i = 2 * p; i <= n; i += p) {
      phi[i] = (phi[i] / p) * (p - 1);
    }
  }
}
```

4 The Möbius Inversion

Theorem 7. *Let f and g be two arithmetic functions. Then, $g(n) = \sum_{d|n} f(d)$ if and only if $f(n) = \sum_{d|n} g(d)\mu(n/d)$.*

Proof. Note that $g(n) = \sum_{d|n} f(d) = \sum_{d|n} f(d)1(n/d) = (f * 1)(n)$, which implies $g = f * 1$. Similarly, $f(n) = \sum_{d|n} g(d)\mu(n/d) = (g * \mu)(n)$, which implies $f = g * \mu$. Hence it suffices to show that $g = f * 1$ if and only if $f = g * \mu$.

$$g * \mu = (f * 1) * \mu = f * (1 * \mu) = f * \varepsilon = f$$

. Now for converse,

$$f * 1 = (g * \mu) * 1 = g * (1 * \mu) = g * \varepsilon = g$$

. $\square$

I kept this single theorem in a separate section only because it is probably the most useful part of the entire note!

5 Problems

Problem 1. Given n, calculate the number of pairs (i, j) such that $i, j \in [1, n]$ and $gcd(i, j) = 1$.
Solution. We basically need to find the value of $\sum_{i=1} \sum_{j=1} [gcd(i, j) = 1]$. Now note that $[gcd(i, j) = 1] = \varepsilon(gcd(i, j))$. Hence,