```cpp
// ret[i] = a[i] + a[i + 1] + ... (for length times, with looping)
vll loop_vec_accumulate(const vll &a, ll length) {
  int n = a.size();
  vll ret(n, 0);
  if (n == 0) return ret;
  ll p = length / n;
  if (p > 0) {
    Loop(i, n) ret[0] += a[i];
    ret[0] *= p;
  }
  Loop(i, length % n) ret[0] += a[i];
  Loop1(i, n - 1) {
    ret[i] = ret[i - 1] - a[i - 1] + a[(i + length - 1) % n];
  }
  return ret;
}

vvll loop_mx_accumulate(const vvll &A, ll i_length, ll j_length) {
  int m = A.size();
  int n = A[0].size();
  Loop(i, m) A[i] = loop_vec_accumulate(A[i], j_length);
  vvll trans_A(n, vll(m, 0));
  Loop(i, n) {
    Loop(j, m) trans_A[i][j] = A[j][i];
  }
  Loop(i, n) trans_A[i] = loop_vec_accumulate(trans_A[i], i_length);
  Loop(i, m) {
    Loop(j, n) A[i][j] = trans_A[j][i];
  }
  return A;
}
```

```cpp
// range = [l, r), return last value causing "t" in evalfunc that returns l->[t,...,t,f,...,f)->r
// NOTE: if [f,...,f) then return l - 1, if [l, r) = empty set then invalid use
template<class val_t, class bsargv_t>
val_t lower_bsearch(val_t l, val_t r, const bsargv_t &v, bool(*evalfunc)(val_t, const bsargv_t&)) {
  if (r - l == 1) {
    if (evalfunc(l, v)) return l;
    else return l - 1;
  }
  val_t m = (l + r) / 2;
  if (evalfunc(m, v)) return lower_bsearch<val_t, bsargv_t>(m, r, v, evalfunc);
  else return lower_bsearch<val_t, bsargv_t>(l, m, v, evalfunc);
}

// range = [l, r), return first value causing "t" in evalfunc that returns l->[f,...,f,t,...,t)->r
// NOTE: if [f,...,f) then return r, if [l, r) = empty set then invalid use
template<class val_t, class bsargv_t>
val_t upper_bsearch(val_t l, val_t r, const bsargv_t &v, bool(*evalfunc)(val_t, const bsargv_t&)) {
  if (r - l == 1) {
    if (evalfunc(l, v)) return l;
    else return r;
  }
  val_t m = (l + r) / 2;
  if (evalfunc(m, v)) return upper_bsearch<val_t, bsargv_t>(l, m, v, evalfunc);
  else return upper_bsearch<val_t, bsargv_t>(m, r, v, evalfunc);
}

struct bsargv_t {
  //
};

bool evalfunc(int val, const bsargv_t &v) {
  //
  return true;
}
```

```cpp
namespace Fourier_transform {

  vector<cdouble> omegas, iomegas;

  inline int bit_reverse(int x, int digit) {
    int ret = digit ? x & 1 : 0;
    Loop(i, digit - 1) { ret <<= 1; x >>= 1; ret |= x & 1; }
    return ret;
  }

  inline void make_omegas(int n) {
    if (omegas.size() != n) {
      omegas.resize(n);
      Loop(i, n) omegas[i] = exp(cdouble({ 0, 2 * PI * i / n }));
    }
  }

  inline void make_iomegas(int n) {
    if (iomegas.size() != n) {
      iomegas.resize(n);
      Loop(i, n) iomegas[i] = exp(cdouble({ 0, -2 * PI * i / n }));
    }
  }

  // a.size() should be 2^digit
  vector<cdouble> FFT(const vector<cdouble> a) {
    int n = int(a.size());
    int digit = int(rndf(log2(n)));
    vector<cdouble> ret = a;
    make_omegas(n);
    Loop(i, n) {
      int j = bit_reverse(i, digit);
      if (j > i) swap(ret[i], ret[j]);
    }
    Loop(i, digit) {
      int j = 0, m = 1 << i, mw = (digit - i - 1);
      Loop(group_id, n >> (i + 1)) {
        Loop(k, m) {
          cdouble x = ret[j] + omegas[k << mw] * ret[j + m];
          cdouble y = ret[j] - omegas[k << mw] * ret[j + m];
          ret[j] = x; ret[j + m] = y;
          ++j;
        }
        j += m;
      }
    }
    return ret;
  }

  // f.size() should be 2^digit
  vector<cdouble> IFFT(const vector<cdouble>& f) {
    int n = int(f.size());
    int digit = int(rndf(log2(n)));
    vector<cdouble> ret = f;
    make_iomegas(n);
    Loopr(i, digit) {
      int j = 0, m = 1 << i, mw = (digit - i - 1);
      Loop(group_id, n >> (i + 1)) {
        Loop(k, m) {
          cdouble q = (ret[j] + ret[j + m]) * 0.5;
          cdouble r = (ret[j] - ret[j + m]) * 0.5 * iomegas[k << mw];
          ret[j] = q; ret[j + m] = r;
          ++j;
        }
        j += m;
      }
    }
    Loop(i, n) {
      int j = bit_reverse(i, digit);
      if (j > i) swap(ret[i], ret[j]);
    }
```

```
72      return ret;
73    }
74
75    // a.size() = b.size() should be 2^digit
76    vector<cdouble> mul_convolution(const vector<cdouble> &a, const vector<cdouble> &b) {
77      int n = int(a.size());
78      vector<cdouble> ret;
79      vector<cdouble> g = FFT(a), h = FFT(b);
80      Loop(i, n) g[i] *= h[i];
81      ret = IFFT(g);
82      return ret;
83    }
84
85    int legal_size_of(int n) {
86      int ret = 1 << (int)log2(n);
87      if (ret < n) ret <<= 1;
88      return ret;
89    }
90  }
91
92  using namespace Fourier_transform;
```

```cpp
bool feq(double x, double y) { return abs(x - y) <= eps; }
bool fge(double x, double y) { return x >= y - eps; }
double fsqrt(double x) { return feq(x, 0) ? 0 : sqrt(x); }

// polygon

struct pt_t {
  double x, y;
  pt_t operator+(const pt_t &p) { return { x + p.x, y + p.y }; }
  pt_t operator-(const pt_t &p) { return { x - p.x, y - p.y }; }
  pt_t operator*(const double &c) { return { x * c, y * c }; }
  bool operator<(const pt_t &another) const {
    return (x != another.x ? x < another.x : y < another.y);
  }
};

// aX + bY + c = 0
struct line_t {
  double a, b, c;
};

// (X - x)^2 + (Y - y)^2 = r^2
struct circle_t {
  double x, y, r;
};

// normal vector = (a, b), passing p
line_t solve_line(double a, double b, pt_t p) {
  return { a, b, -a * p.x - b * p.y };
}

// passing p, q
line_t solve_line(pt_t p, pt_t q) {
  return solve_line(q.y - p.y, -q.x + p.x, p);
}

// t should be radius
pt_t rot(pt_t p, double r) {
  return {
    cos(r) * p.x - sin(r) * p.y,
    sin(r) * p.x + cos(r) * p.y
  };
}

double norm2(pt_t p) {
  return p.x * p.x + p.y * p.y;
}

double norm(pt_t p) {
  return sqrt(norm2(p));
}

// angle [0, 2PI) of vector p to vector q
double angle(pt_t p, pt_t q) {
  p = p * (1.0 / norm(p));
  q = q * (1.0 / norm(q));
  double r0 = acos(max(min(p.x * q.x + p.y * q.y, 1.0), -1.0));
  double r1 = asin(max(min(p.x * q.y - p.y * q.x, 1.0), -1.0));
  if (r1 >= 0) return r0;
  else return 2 * M_PI - r0;
}

double dist(line_t l, pt_t p) {
  return abs(l.a * p.x + l.b * p.y + l.c)
    / sqrt(l.a * l.a + l.b * l.b);
}

bool on_same_line(pt_t s, pt_t t, pt_t p) {
  line_t l = solve_line(s, t);
  if (feq(dist(l, p), 0)) return true;
  else return false;
```

```cpp
}

bool in_segment(pt_t s, pt_t t, pt_t p) {
  line_t l = solve_line(s, t);
  if (feq(dist(l, p), 0)
    && fge(p.x, min(s.x, t.x))
    && fge(max(s.x, t.x), p.x)
    && fge(p.y, min(s.y, t.y))
    && fge(max(s.y, t.y), p.y)) return true;
  else return false;
}

// (NAN, NAN) if lines coincide with each other
// (INF, INF) if lines are parallel but not coincide
pt_t cross_point(line_t l, line_t m) {
  double d = l.a * m.b - l.b * m.a;
  if (feq(d, 0)) {
    if (feq(l.a * m.c - l.c * m.a, 0)) return { INF, INF };
    else return { NAN, NAN };
  }
  else {
    double x = l.b * m.c - m.b * l.c;
    double y = l.a * m.c - m.a * l.c;
    return { x / d, y / -d };
  }
}

// if size is 0, then not crossed
vector<pt_t> cross_point(circle_t f, line_t l) {
  double d = dist(l, { f.x, f.y });
  if (!fge(f.r, d)) return {};
  line_t m = solve_line(l.b, -l.a, { f.x, f.y });
  pt_t p = cross_point(l, m);
  if (feq(d, f.r)) return { p };
  else {
    pt_t u = { l.b, -l.a };
    pt_t v = u * (sqrt(pow(f.r, 2) - pow(d, 2)) / norm(u));
    return { p + v, p - v };
  }
}

// if size is 0, then not crossed
vector<pt_t> cross_point(circle_t f, circle_t g) {
  line_t l = {
    -2 * f.x + 2 * g.x,
    -2 * f.y + 2 * g.y,
    (f.x * f.x + f.y * f.y - f.r * f.r) - (g.x * g.x + g.y * g.y - g.r * g.r)
  };
  return cross_point(f, l);
}

// tangent points of f through p
// if size is 0, then p is strictly contained in f
// if size is 1, then p is on f
// otherwise size is 2
vector<pt_t> tangent_point(circle_t f, pt_t p) {
  vector<pt_t> ret;
  double d2 = norm2(pt_t({ f.x, f.y }) - p);
  double r2 = d2 - f.r * f.r;
  if (fge(r2, 0)) {
    circle_t g = { p.x, p.y, fsqrt(r2) };
    ret = cross_point(f, g);
  }
  return ret;
}

// tangent lines of f through p
// if size is 0, then p is strictly contained in f
// if size is 1, then p is on f
// otherwise size is 2
vector<line_t> tangent_line(circle_t f, pt_t p) {
```

```cpp
  vector<pt_t> qs = tangent_point(f, p);
  vector<line_t> ret(qs.size());
  Loop(i, ret.size()) {
    ret[i] = solve_line(qs[i].x - f.x, qs[i].y - f.y, qs[i]);
  }
  return ret;
}

// tangent points on f through which there is a line tangent to g
// if size is 0, then one is strictly contained in the other
// if size is 1, then they are touched inside
// if size is 2, then they are crossed
// if size is 3, then they are touched outside
// otherwise size is 4
vector<pt_t> tangent_point(circle_t f, circle_t g) {
  vector<pt_t> ret;
  double d2 = norm2({ g.x - f.x, g.y - f.y });
  vector<double> r2(2);
  r2[0] = d2 - f.r * f.r + 2 * f.r * g.r;
  r2[1] = d2 - f.r * f.r - 2 * f.r * g.r;
  Loop(k, 2) {
    if (fge(r2[k], 0)) {
      circle_t g2 = { g.x, g.y, fsqrt(r2[k]) };
      vector<pt_t> buf = cross_point(f, g2);
      Loop(i, buf.size()) ret.push_back(buf[i]);
    }
  }
  return ret;
}

// common tangent lines between two circles
// if size is 0, then one is strictly contained in the other
// if size is 1, then they are touched inside
// if size is 2, then they are crossed
// if size is 3, then they are touched outside
// otherwise size is 4
vector<line_t> tangent_line(circle_t f, circle_t g) {
  vector<pt_t> qs = tangent_point(f, g);
  vector<line_t> ret(qs.size());
  Loop(i, ret.size()) {
    ret[i] = tangent_line(f, qs[i]).front();
  }
  return ret;
}

// suppose a.size() >= 3
double polygon_area(vector<pt_t> a) {
  double ret = 0;
  Loop(i, a.size()) {
    int j = (i + 1 < a.size() ? i + 1 : 0);
    ret += a[i].x * a[j].y - a[j].x * a[i].y;
  }
  ret = abs(ret) / 2;
  return ret;
}
```

```cpp
// include modll

namespace number_theoretic_transform {

  // when MOD - 1 = 2^m * a,
  // min_omega = root^a (try 3,5,7,... to get root)
  // min_omega_depth = m
  // mod_half = (MOD + 1) / 2

  modll min_omega;
  int min_omega_depth;
  modll mod_half;

  void make_base(int mode) {
    switch (mode) {
      /*
    case 0:
      MOD = 167772161;
      min_omega = 17;
      min_omega_depth = 25;
      mod_half = 83886081;
      break;
    case 1:
      MOD = 469762049;
      min_omega = 30;
      min_omega_depth = 26;
      mod_half = 234881025;
      break;
    case 2:
      MOD = 1224736769;
      min_omega = 149;
      min_omega_depth = 24;
      mod_half = 612368385;
      break;
      */
    default:
      MOD = 924844033;
      min_omega = 3597;
      min_omega_depth = 21;
      mod_half = 462422017;
    }
  }

  vector<modll> omegas, iomegas;

  inline int bit_reverse(int x, int digit) {
    int ret = digit ? x & 1 : 0;
    Loop(i, digit - 1) { ret <<= 1; x >>= 1; ret |= x & 1; }
    return ret;
  }

  inline void make_omegas(int n) {
    if (omegas.size() != n) {
      omegas.resize(n);
      modll omega = pow(min_omega, (1 << min_omega_depth) / n);
      Loop(i, n) {
        if (i == 0) omegas[i] = 1;
        else omegas[i] = omegas[i - 1] * omega;
      }
    }
  }

  inline void make_iomegas(int n) {
    if (iomegas.size() != n) {
      iomegas.resize(n);
      modll iomega = modll(1) / pow(min_omega, (1 << min_omega_depth) / n);
      Loop(i, n) {
        if (i == 0) iomegas[i] = 1;
        else iomegas[i] = iomegas[i - 1] * iomega;
      }
    }
```

```
72    }
73
74    // a.size() should be 2^digit
75    vector<modll> NTT(const vector<modll> a, int mode = 0) {
76      int n = int(a.size());
77      int digit = int(rndf(log2(n)));
78      vector<modll> ret = a;
79      make_omegas(n);
80      Loop(i, n) {
81        int j = bit_reverse(i, digit);
82        if (j > i) swap(ret[i], ret[j]);
83      }
84      Loop(i, digit) {
85        int j = 0, m = 1 << i, mw = (digit - i - 1);
86        Loop(group_id, n >> (i + 1)) {
87          Loop(k, m) {
88            modll x = ret[j] + omegas[k << mw] * ret[j + m];
89            modll y = ret[j] - omegas[k << mw] * ret[j + m];
90            ret[j] = x; ret[j + m] = y;
91            ++j;
92          }
93          j += m;
94        }
95      }
96      return ret;
97    }
98
99    // f.size() should be 2^digit
100   vector<modll> INTT(const vector<modll>& f, int mode = 0) {
101     int n = int(f.size());
102     int digit = int(rndf(log2(n)));
103     vector<modll> ret = f;
104     make_iomegas(n);
105     Loopr(i, digit) {
106       int j = 0, m = 1 << i, mw = (digit - i - 1);
107       Loop(group_id, n >> (i + 1)) {
108         Loop(k, m) {
109           modll q = (ret[j] + ret[j + m]) * mod_half;
110           modll r = (ret[j] - ret[j + m]) * mod_half * iomegas[k << mw];
111           ret[j] = q; ret[j + m] = r;
112           ++j;
113         }
114         j += m;
115       }
116     }
117     Loop(i, n) {
118       int j = bit_reverse(i, digit);
119       if (j > i) swap(ret[i], ret[j]);
120     }
121     return ret;
122   }
123
124   // a.size() = b.size() should be 2^digit
125   vector<modll> mul_convolution(const vector<modll> &a, const vector<modll> &b) {
126     int n = int(a.size());
127     vector<modll> ret;
128     make_base(0);
129     // Garner's algorithm is unsupported yet
130     vector<modll> g = NTT(a), h = NTT(b);
131     Loop(i, n) g[i] *= h[i];
132     ret = INTT(g);
133     return ret;
134   }
135
136   int legal_size_of(int n) {
137     int ret = 1 << (int)log2(n);
138     if (ret < n) ret <<= 1;
139     return ret;
140   }
141 }
142
```

```
143  using namespace number_theoretic_transform;
```

```cpp
ll powll(ll n, ll p) {
  if (p == 0) return 1;
  else if (p == 1) return n;
  else {
    ll ans = powll(n, p / 2);
    ans = ans * ans;
    if (p % 2 == 1) ans = ans * n;
    return ans;
  }
}

// n = 1.5e7 -> 80 ms
vll list_prime_until(ll n) {
  vll ret;
  vector<bool> a(n + 1, true); // is_prime
  if (a.size() > 0) a[0] = false;
  if (a.size() > 1) a[1] = false;
  Loop(i, n + 1) {
    if (a[i]) {
      ret.push_back(i);
      ll k = (ll)i * i;
      while (k < n + 1) {
        a[int(k)] = false;
        k += i;
      }
    }
  }
  return ret;
}

// primes has to be generated by list_prime_until(>=sqrt(n))
vector<Pll> prime_factorize(ll n, const vll &primes) {
  vector<Pll> ret;
  Loop(i, primes.size()) {
    if (n == 1) break;
    while (n % primes[i] == 0) {
      if (ret.size() == 0 || ret.back().fst != primes[i]) {
        ret.push_back({ primes[i], 0 });
      }
      ret.back().snd++;
      n /= primes[i];
    }
  }
  if (n != 1) ret.push_back({ n, 1 });
  return ret;
}

vll divisors(const vector<Pll> factors) {
  queue<ll> que;
  que.push(1);
  Loop(i, factors.size()) {
    ll x = factors[i].fst, d = factors[i].snd;
    vll a(d + 1, 1); Loop1(j, d) a[j] = a[j - 1] * x;
    int m = int(que.size());
    Loop(j, m) {
      ll y = que.front(); que.pop();
      Loop(k, d + 1) que.push(y * a[k]);
    }
  }
  int m = int(que.size());
  vll ret(m);
  Loop(i, m) {
    ret[i] = que.front(); que.pop();
  }
  sort(ret.begin(), ret.end());
  return ret;
}
```

```cpp
namespace Zeta_and_Mobius_transform {

  // f.size() should be 2^digit, ret will assemble value from subsets
  vll Zeta_trans(vll f) {
    int n = f.size();
    int digit = int(rndf(log2(n)));
    vll ret = f;
    Loop(i, digit) {
      int x = 1 << i;
      Loop(j, n) {
        if (j & x) ret[j] += ret[j ^ x];
      }
    }
    return ret;
  }

  // g.size() should be 2^digit, ret will disassemble value to subsets
  vll Mobius_trans(vll g) {
    int n = g.size();
    int digit = int(rndf(log2(n)));
    vll ret = g;
    Loop(i, digit) {
      int x = 1 << i;
      Loop(j, n) {
        if (j & x) ret[j] -= ret[j ^ x];
      }
    }
    return ret;
  }

  // f.size() should be 2^digit, ret will assemble value from supersets
  vll Zeta_trans_rev(vll f) {
    int n = f.size();
    int digit = int(rndf(log2(n)));
    vll ret = f;
    Loop(i, digit) {
      int x = 1 << i;
      Loop(j, n) {
        if (!(j & x)) ret[j] += ret[j | x];
      }
    }
    return ret;
  }

  // g.size() should be 2^digit, ret will disassemble value to supersets
  vll Mobius_trans_rev(vll g) {
    int n = g.size();
    int digit = int(rndf(log2(n)));
    vll ret = g;
    Loop(i, digit) {
      int x = 1 << i;
      Loop(j, n) {
        if (!(j & x)) ret[j] -= ret[j | x];
      }
    }
    return ret;
  }

  int legal_size_of(int n) {
    int ret = 1 << (int)log2(n);
    if (ret < n) ret <<= 1;
    return ret;
  }
}

using namespace Zeta_and_Mobius_transform;
```

```cpp
namespace mod_op {

  const ll MOD = // (ll)1e9 + 7;

    class modll {
    private:
      ll val;
      ll modify(ll x) const { ll ret = x % MOD; if (ret < 0) ret += MOD; return ret; }
      ll inv(ll x) const {
        if (x == 0) return 1 / x;
        else if (x == 1) return 1;
        else return modify(inv(MOD % x) * modify(-MOD / x));
      }
    public:
      modll(ll init = 0) { val = modify(init); return; }
      modll(const modll& another) { val = another.val; return; }
      modll& operator=(const modll &another) { val = another.val; return *this; }
      modll operator+(const modll &x) const { return modify(val + x.val); }
      modll operator-(const modll &x) const { return modify(val - x.val); }
      modll operator*(const modll &x) const { return modify(val * x.val); }
      modll operator/(const modll &x) const { return modify(val * inv(x.val)); }
      modll& operator+=(const modll &x) { val = modify(val + x.val); return *this; }
      modll& operator-=(const modll &x) { val = modify(val - x.val); return *this; }
      modll& operator*=(const modll &x) { val = modify(val * x.val); return *this; }
      modll& operator/=(const modll &x) { val = modify(val * inv(x.val)); return *this; }
      bool operator==(const modll &x) { return val == x.val; }
      bool operator!=(const modll &x) { return val != x.val; }
      friend istream& operator >> (istream &is, modll& x) { is >> x.val; return is; }
      friend ostream& operator << (ostream &os, const modll& x) { os << x.val; return os; }
      ll get_val() { return val; }
  };

  modll pow(modll n, ll p) {
    modll ret;
    if (p == 0) ret = 1;
    else if (p == 1) ret = n;
    else {
      ret = pow(n, p / 2);
      ret *= ret;
      if (p % 2 == 1) ret *= n;
    }
    return ret;
  }

  vector<modll> facts;

  void make_facts(int n) {
    if (facts.empty()) facts.push_back(modll(1));
    for (int i = (int)facts.size(); i <= n; ++i) facts.push_back(modll(facts.back() * (ll)i));
    return;
  }

  vector<modll> ifacts;
  vector<modll> invs;

  void make_invs(int n) {
    if (invs.empty()) {
      invs.push_back(modll(0));
      invs.push_back(modll(1));
    }
    for (int i = (int)invs.size(); i <= n; ++i) {
      // because 0 = MOD = kq + r, 1/k = -q/r
      invs.push_back(invs[(int)MOD % i] * ((int)MOD - (int)MOD / i));
    }
    return;
  }

  void make_ifacts(int n) {
    make_invs(n);
    if (ifacts.empty()) ifacts.push_back(modll(1));
    for (int i = (int)ifacts.size(); i <= n; ++i) ifacts.push_back(modll(ifacts.back() * invs[i]));
```

```
72      return;
73    }
74
75    //nCr
76    modll combination(ll n, ll r) {
77      if (n >= r && r >= 0) {
78        modll ret;
79        make_facts((int)n);
80        make_ifacts((int)n);
81        ret = facts[(unsigned)n] * ifacts[(unsigned)r] * ifacts[(unsigned)(n - r)];
82        return ret;
83      }
84      else return 0;
85    }
86
87    modll get_fact(ll n) {
88      make_facts((int)n);
89      return facts[(int)n];
90    }
91
92    modll get_ifact(ll n) {
93      make_ifacts((int)n);
94      return ifacts[(int)n];
95    }
96
97    vector<vector<modll>> Stirling_nums2;
98    vector<vector<modll>> Stirling_nums2_sum;
99
100   void make_Stirling_nums2(int n) {
101     for (int i = (int)Stirling_nums2.size(); i <= n; ++i) {
102       Stirling_nums2.push_back(vector<modll>(i + 1));
103       Stirling_nums2_sum.push_back(vector<modll>(i + 1, 0));
104       Loop(j, i + 1) {
105         if (j == 0) Stirling_nums2[i][j] = 0;
106         else if (j == 1) Stirling_nums2[i][j] = 1;
107         else if (j == i) Stirling_nums2[i][j] = 1;
108         else Stirling_nums2[i][j] = Stirling_nums2[i - 1][j - 1] + Stirling_nums2[i - 1][j] * modll(j);
109         if (j > 0) Stirling_nums2_sum[i][j] = Stirling_nums2_sum[i][j - 1] + Stirling_nums2[i][j];
110       }
111     }
112   }
113
114   modll get_Stirling_num2(ll n, ll r) {
115     if (n >= r && r >= 0) {
116       make_Stirling_nums2((int)n);
117       return Stirling_nums2[(int)n][(int)r];
118     }
119     else return 0;
120   }
121
122   modll get_Stirling_num2_sum(ll n, ll r) {
123     if (n >= r && r >= 0) {
124       make_Stirling_nums2((int)n);
125       return Stirling_nums2_sum[(int)n][(int)r];
126     }
127     else return 0;
128   }
129
130   vector<vector<modll>> partition_nums;
131   vector<vector<modll>> partition_nums_sum;
132
133   void make_partition_nums(int n) {
134     for (int i = (int)partition_nums.size(); i <= n; ++i) {
135       partition_nums.push_back(vector<modll>(i + 1));
136       partition_nums_sum.push_back(vector<modll>(i + 1, 0));
137       Loop(j, i + 1) {
138         if (j == 0) partition_nums[i][j] = 0;
139         else if (j == 1) partition_nums[i][j] = 1;
140         else if (j == i) partition_nums[i][j] = 1;
141         else partition_nums[i][j] = partition_nums[i - 1][j - 1] + (i >= j * 2 ? partition_nums[i - j]
            [j] : 0);
```

```
142         if (j > 0) partition_nums_sum[i][j] = partition_nums_sum[i][j - 1] + partition_nums[i][j];
143       }
144     }
145   }
146   modll get_partition_num(ll n, ll r) {
147     if (n >= r && n >= 0) {
148       make_partition_nums((int)n);
149       return partition_nums[(int)n][(int)r];
150     }
151     else return 0;
152   }
153
154   modll get_partition_num_sum(ll n, ll r) {
155     if (n >= r && r >= 0) {
156       make_partition_nums((int)n);
157       return partition_nums_sum[(int)n][(int)r];
158     }
159     else return 0;
160   }
161
162   //log_a(b), if x does not exist, return -1
163   ll disc_log(modll a, modll b) {
164     ll ret = -1;
165     ll m = ceilsqrt(MOD);
166     unordered_map<ll, ll> mp;
167     modll x = 1;
168     Loop(i, m) {
169       mp[x.get_val()] = i;
170       x *= a;
171     }
172     x = modll(1) / pow(a, m);
173     modll k = b;
174     Loop(i, m) {
175       if (mp.find(k.get_val()) == mp.end()) k *= x;
176       else {
177         ret = i * m + mp[k.get_val()];
178         break;
179       }
180     }
181     return ret;
182   }
183 }
184
185 using namespace mod_op;
186 typedef vector<modll> vmodll;
187 typedef vector<vector<modll>> vvmodll;
188
189 // the number of methods of dividing n factors into r groups
190 // recommend to consider corner case (n == 0 or r == 0) irregularly
191 modll grouping(ll n, ll r, bool distinct_n, bool distinct_r, bool enable_empty_r) {
192   int mode = (distinct_n ? 0b100 : 0) + (distinct_r ? 0b010 : 0) + (enable_empty_r ? 0b001 : 0);
193   if (n < 0 || r < 0) return 0;
194   switch (mode) {
195   case 0b000:
196     return get_partition_num(n, r);
197   case 0b001:
198     return get_partition_num_sum(n, r);
199   case 0b010:
200     return combination(n - 1, r - 1);
201   case 0b011:
202     return combination(n + r - 1, r - 1);
203   case 0b100:
204     return get_Stirling_num2(n, r);
205   case 0b101:
206     return get_Stirling_num2_sum(n, r);
207   case 0b110:
208     return get_Stirling_num2(n, r) * get_fact(r);
209   case 0b111:
210     return pow(modll(r), n);
211   default:
212     return 0;
```

```
  }
}

using namespace mod_op;
using vmodll = vector<modll>;
using vvmodll = vector<vmodll>;
using vvvmodll = vector<vvmodll>;
```

```cpp
namespace strll_op {

  class strll {
  private:
    string val; // interior process is always reversed
    inline string ll_to_strll(ll x) {
      if (x == 0) return "0";
      bool neg_flag = false;
      if (x < 0) { neg_flag = true; x *= -1; }
      string ret = "";
      while (x > 0) {
        ret += '0' + (x % 10);
        x /= 10;
      }
      if (neg_flag) ret += '-';
      return ret;
    }
    inline string uadd_core(const string &s, const string &t) {
      int n = s.length();
      int m = t.length();
      string ret = "";
      int v_digits = max(n, m) + 1;
      vi v(v_digits, 0);
      Loop(i, v_digits - 1) {
        if (i < n) v[i] += s[i] - '0';
        if (i < m) v[i] += t[i] - '0';
        if (v[i] >= 10) {
          v[i] -= 10;
          v[i + 1] += 1;
        }
      }
      if (v[v_digits - 1] == 0) v_digits = max(1, v_digits - 1);
      Loop(i, v_digits) ret += '0' + v[i];
      return ret;
    }
    inline string usub_core(const string &s, const string &t) {
      int n = s.length();
      int m = t.length();
      string ret = "";
      int v_digits = 1;
      vi v(n, 0);
      Loop(i, n) {
        v[i] += s[i] - '0';
        if (i < m) v[i] -= (t[i] - '0');
        if (v[i] < 0) {
          v[i] += 10;
          v[i + 1] -= 1;
        }
        if (v[i] > 0) v_digits = i + 1;
      }
      Loop(i, v_digits) ret += '0' + v[i];
      return ret;
    }
    inline string umul_core(const string &s, const string &t) {
      int n = s.length();
      int m = t.length();
      string ret = "";
      vi v(n + m, 0);
      Loop(i, n) {
        Loop(j, m) {
          int z = (s[i] - '0') * (t[j] - '0');
          v[i + j] += z % 10;
          v[i + j + 1] += z / 10;
        }
      }
      int v_digits = 1;
      Loop(i, n + m - 1) {
        v[i + 1] += v[i] / 10;
        v[i] %= 10;
        if (v[i + 1] > 0) v_digits = i + 2;
      }
```

```
72        Loop(i, v_digits) ret += '0' + v[i];
73        return ret;
74      }
75      inline bool uge_core(const string &s, const string &t) const {
76        int n = s.length();
77        int m = t.length();
78        while (n > 1 && s[n - 1] == '0') n--;
79        while (m > 1 && t[m - 1] == '0') m--;
80        if (n > m) return true;
81        else if (n < m) return false;
82        else {
83          Loopr(i, n) {
84            if (s[i] > t[i]) return true;
85            if (s[i] < t[i]) return false;
86          }
87          return true;
88        }
89      }
90      inline string udiv_core(string s, const string &t, bool rem_flag) {
91        int n = s.length();
92        int m = t.length();
93        string ret = "0";
94        Loopr(i, n - m + 1) {
95          ret += '0';
96          string sbuf = s.substr(i, m + 1);
97          while (uge_core(sbuf, t)) {
98            sbuf = usub_core(sbuf, t);
99            ret.back()++;
100         }
101         Loop(j, min(m + 1, n)) {
102           s[i + j] = j < sbuf.size() ? sbuf[j] : '0';
103         }
104       }
105       reverse(ret.begin(), ret.end());
106       if (rem_flag) ret = s;
107       while (ret.size() > 1 && ret.back() == '0') ret.pop_back();
108       return ret;
109     }
110     inline string add(const string &s, const string &t) {
111       int n = int(s.length());
112       int m = int(t.length());
113       string ret = "";
114       int mode = (s[n - 1] == '-' ? 0b10 : 0) + (t[m - 1] == '-' ? 0b01 : 0);
115       switch (mode) {
116       case 0b00:
117         ret = uadd_core(s, t);
118         break;
119       case 0b01:
120         if (uge_core(s.substr(0, n), t.substr(0, m - 1))) ret = usub_core(s, t.substr(0, m - 1));
121         else ret = usub_core(t.substr(0, m - 1), s) + '-';
122         break;
123       case 0b10:
124         if (uge_core(s.substr(0, n - 1), t.substr(0, m))) ret = usub_core(s.substr(0, n - 1), t) + '-';
125         else ret = usub_core(t, s.substr(0, n - 1));
126         break;
127       case 0b11:
128         ret = uadd_core(s.substr(0, n - 1), t.substr(0, m - 1)) + '-';
129         break;
130       }
131       if (ret == "0-") ret.pop_back();
132       return ret;
133     }
134     inline string sub(const string &s, const string &t) {
135       string ret = "";
136       int n = s.length();
137       int m = t.length();
138       int mode = (s[n - 1] == '-' ? 0b10 : 0) + (t[m - 1] == '-' ? 0b01 : 0);
139       switch (mode) {
140       case 0b00:
141         if (uge_core(s.substr(0, n), t.substr(0, m))) ret = usub_core(s, t);
142         else ret = usub_core(t, s) + '-';
```

```
143         break;
144       case 0b01:
145         ret = uadd_core(s, t.substr(0, m - 1));
146         break;
147       case 0b10:
148         ret = uadd_core(s.substr(0, n - 1), t) + '-';
149         break;
150       case 0b11:
151         if (uge_core(s.substr(0, n - 1), t.substr(0, m - 1))) ret = usub_core(s.substr(0, n - 1), t.substr
            (0, m - 1)) + '-';
152         else ret = usub_core(t.substr(0, m - 1), s.substr(0, n - 1));
153         break;
154       }
155       if (ret == "0-") ret.pop_back();
156       return ret;
157     }
158     inline string mul(const string &s, const string &t) {
159       string ret;
160       int n = s.length();
161       int m = t.length();
162       int mode = (s[n - 1] == '-' ? 0b10 : 0) + (t[m - 1] == '-' ? 0b01 : 0);
163       switch (mode) {
164       case 0b00:
165         ret = umul_core(s, t);
166         break;
167       case 0b01:
168         ret = umul_core(s, t.substr(0, m - 1)) + '-';
169         break;
170       case 0b10:
171         ret = umul_core(s.substr(0, n - 1), t) + '-';
172         break;
173       case 0b11:
174         ret = umul_core(s.substr(0, n - 1), t.substr(0, m - 1));
175         break;
176       }
177       if (ret == "0-") ret.pop_back();
178       return ret;
179     }
180     inline bool ge(const string &s, const string &t) const {
181       bool ret;
182       int n = s.length();
183       int m = t.length();
184       int mode = (s[n - 1] == '-' ? 0b10 : 0) + (t[m - 1] == '-' ? 0b01 : 0);
185       switch (mode) {
186       case 0b00:
187         ret = uge_core(s, t);
188         break;
189       case 0b01:
190         ret = true;
191         break;
192       case 0b10:
193         ret = false;
194         break;
195       case 0b11:
196         if (s == t) ret = true;
197         else ret = !uge_core(s.substr(0, n - 1), t.substr(0, m - 1));
198         break;
199       }
200       return ret;
201     }
202     inline string div(const string &s, const string &t, bool rem_flag) {
203       string ret;
204       int n = s.length();
205       int m = t.length();
206       int mode = (s[n - 1] == '-' ? 0b10 : 0) + (t[m - 1] == '-' ? 0b01 : 0);
207       switch (mode) {
208       case 0b00:
209         ret = udiv_core(s, t, rem_flag);
210         break;
211       case 0b01:
212         ret = udiv_core(s, t.substr(0, m - 1), rem_flag);
```

```
213         if (!rem_flag) ret += '-';
214         break;
215       case 0b10:
216         ret = udiv_core(s.substr(0, n - 1), t, rem_flag) + '-';
217         break;
218       case 0b11:
219         ret = udiv_core(s.substr(0, n - 1), t.substr(0, m - 1), rem_flag);
220         if (rem_flag) ret += '-';
221         break;
222       }
223       if (ret == "0-") ret.pop_back();
224       return ret;
225     }
226   public:
227     strll(string init = "0") { reverse(init.begin(), init.end()); val = init; return; }
228     strll(ll init) { val = ll_to_strll(init); return; }
229     strll(const strll& another) { val = another.val; return; }
230     inline strll& operator=(const strll &another) { val = another.val; return *this; }
231     inline strll operator+(const strll &x) { strll ret; ret.val = add(val, x.val); return ret; }
232     inline strll operator-(const strll &x) { strll ret; ret.val = sub(val, x.val); return ret; }
233     inline strll operator*(const strll &x) { strll ret; ret.val = mul(val, x.val); return ret; }
234     inline strll operator/(const strll &x) { strll ret; ret.val = div(val, x.val, false); return ret; }
235     inline strll operator%(const strll &x) { strll ret; ret.val = div(val, x.val, true); return ret; }
236     inline strll& operator+=(const strll &x) { val = add(val, x.val); return *this; }
237     inline strll& operator-=(const strll &x) { val = sub(val, x.val); return *this; }
238     inline strll& operator*=(const strll &x) { val = mul(val, x.val); return *this; }
239     inline strll& operator/=(const strll &x) { val = div(val, x.val, false); return *this; }
240     inline strll& operator%=(const strll &x) { val = div(val, x.val, true); return *this; }
241     inline bool operator>=(const strll &x) { return ge(val, x.val); }
242     inline bool operator>(const strll &x) { return ge(val, x.val) && val != x.val; }
243     inline bool operator<=(const strll &x) { return ge(x.val, val); }
244     inline bool operator<(const strll &x) { return ge(x.val, val) && val != x.val; }
245     inline bool operator==(const strll &x) { return val == x.val; }
246     inline bool operator!=(const strll &x) { return val != x.val; }
247     inline bool operator<(const strll &x) const { return !ge(val, x.val); }
248     friend inline istream& operator >> (istream &is, strll& x) { is >> x.val; reverse(x.val.begin(),
        x.val.end()); return is; }
249     friend inline ostream& operator << (ostream &os, const strll& x) { os << x.get_val(); return os; }
250     string get_val() const { string ret = val; reverse(ret.begin(), ret.end()); return ret; }
251   };
252 }
253
254 using namespace strll_op;
255 typedef vector<strll> vstrll;
256 typedef vector<vector<strll>> vvstrll;
```

```cpp
class Ancestor {
private:
  int n;
  vvi lst;
  vvi table;
  vi from;
  vi visited, departed;
  deque<int> deq;
  void dfs(int a, int &t) {
    for (int i = 0; i < deq.size(); i = i * 2 + 1) {
      table[a].push_back(deq[i]);
    }
    visited[a] = t++;
    deq.push_front(a);
    Foreach(b, lst[a]) {
      if (from[b] == INT_MIN) {
        from[b] = a;
        dfs(b, t);
      }
    }
    deq.pop_front();
    departed[a] = t++;
  }
public:
  Ancestor(const vvi &lst, vi roots = { 0 }) {
    n = lst.size();
    this->lst = lst;
    table = vvi(n);
    from = vi(n, INT_MIN);
    visited.resize(n);
    departed.resize(n);
    int t = 0;
    Foreach(root, roots) {
      from[root] = -1;
      dfs(root, t);
    }
  }
  bool is_ancestor(int des, int anc) {
    return visited[anc] < visited[des]
      && departed[des] < departed[anc];
  }
  int lowest_common_ancestor(int x, int y) {
    if (x == y) return x;
    if (is_ancestor(x, y)) return y;
    if (is_ancestor(y, x)) return x;
    Loop1(i, table[x].size() - 1) {
      if (is_ancestor(y, table[x][i])) {
        return lowest_common_ancestor(table[x][i - 1], y);
      }
    }
    return lowest_common_ancestor(table[x].back(), y);
  }
  int get_ancestor(int des, int k) {
    if (k == 0) return des;
    int l = int(log2(k));
    if (l >= table[des].size()) return -1;
    else return get_ancestor(table[des][l], k - (1 << l));
  }
  // return first value causing "t" in evalfunc that returns descendant->[f,...,f,t,...,t]->root
  // NOTE: if [f,...,f] then return -1
  template<typename bsargv_t>
  int upper_bsearch(int des, const bsargv_t &v, bool(*evalfunc)(int, const bsargv_t&)) {
    if (evalfunc(des, v)) return des;
    if (table[des].size() == 0) return -1;
    Loop1(i, table[des].size() - 1) {
      if (evalfunc(table[des][i], v)) {
        return upper_bsearch(table[des][i - 1], v, evalfunc);
      }
    }
    return upper_bsearch(table[des].back(), v, evalfunc);
  }
```

```cpp
  // return last value causing "t" in evalfunc that returns descendant->[t,...,t,f,...,f]->root
  // NOTE: if [f,...,f] then return -1
  template<typename bsargv_t>
  int lower_bsearch(int des, const bsargv_t &v, bool(*evalfunc)(int, const bsargv_t&)) {
    if (!evalfunc(des, v)) return -1;
    if (table[des].size() == 0) return des;
    Loop(i, table[des].size()) {
      if (!evalfunc(table[des][i], v)) {
        if (i == 0) return des;
        else return lower_bsearch(table[des][i - 1], v, evalfunc);
      }
    }
    return lower_bsearch(table[des].back(), v, evalfunc);
  }
};

```

```cpp
class BIT {
private:
  vll nodes;
  int n;
public:
  BIT(vll a) {
    n = a.size();
    nodes = vll(n, 0);
    Loop(i, a.size()) add(i, a[i]);
  }
  void add(int k, ll x) {
    ++k;
    for (int id = k; id <= n; id += id & -id) {
      nodes[id - 1] += x;
    }
  }
  // note: sum of [s, t)
  ll sum(int s, int t) {
    ll ret = 0;
    for (int id = t; id > 0; id -= id & -id) {
      ret += nodes[id - 1];
    }
    for (int id = s; id > 0; id -= id & -id) {
      ret -= nodes[id - 1];
    }
    return ret;
  }
};

// solve the number of pair(i, j) such that a[i] > a[j] (i < j)
ll solve_inversion_number(const vll &a) {
  int n = a.size();
  map<ll, int> mp;
  Loop(i, n) mp[a[i]] = 1;
  int cnt = 0;
  Loopitr(itr, mp) itr->second = cnt++;
  vi b(n);
  Loop(i, n) b[i] = mp[a[i]];
  BIT bit(vll(cnt, 0));
  ll ret = 0;
  Loop(i, n) {
    ret += bit.sum(b[i] + 1, cnt);
    bit.add(b[i], 1);
  }
  return ret;
}
```

```cpp
class Chuliu_Edmonds {
private:
  struct edge_t {
    int id;
    ll cost;
    stack<int> included_stk;
    bool operator<(const edge_t & another) const {
      return cost > another.cost;
    }
  };
  // edges are directed to the node itself
  struct node {
    int overnode; bool done; bool fin; priority_queue<edge_t> edges; edge_t from;
  };
  vector<node> nodes;
  int n, root;
  stack<int> stk;
  bool no_mca;
  int topnode(int k) {
    int a = k;
    while (nodes[a].overnode != -1) {
      a = nodes[a].overnode;
    }
    if (k != a) nodes[k].overnode = a;
    return a;
  }
  void contract(int s) {
    int a = s;
    priority_queue<edge_t> new_from_edges;
    int cnt = 0;
    do {
      a = topnode(a);
      while (nodes[a].edges.size()) {
        edge_t edge = nodes[a].edges.top();
        nodes[a].edges.pop();
        if (edge.id == nodes[a].from.id) continue;
        edge.cost -= nodes[a].from.cost;
        edge.included_stk.push(a);
        new_from_edges.push(edge);
      }
      nodes[a].overnode = nodes.size();
      a = nodes[a].from.id;
    } while (a != s);
    nodes.push_back({ -1, false, false, new_from_edges,{} });
  }
  void unfold() {
    while (stk.size()) {
      int a = stk.top(); stk.pop();
      if (a >= n) {
        int b = nodes[a].from.included_stk.top();
        ll d = nodes[b].from.cost;
        nodes[b].from = nodes[a].from;
        nodes[b].from.cost += d;
        nodes[b].from.included_stk.pop();
      }
      else nodes[a].fin = true;
    }
  }
public:
  Chuliu_Edmonds(const vvi &lst, const vvll &cst, int start) {
    n = lst.size();
    nodes.resize(n);
    Loop(i, n) nodes[i] = { -1, false, false, priority_queue<edge_t>(),{} };
    Loop(i, n) {
      Loop(j, lst[i].size()) {
        nodes[lst[i][j]].edges.push({ i, cst[i][j], stack<int>() });
      }
    }
    root = start;
    no_mca = false;
    nodes[root].fin = nodes[root].done = true;
```

```
72      Loop(i, n) {
73        if (!nodes[i].fin) {
74          int a = i;
75          nodes[a].done = true;
76          stk.push(a);
77          do {
78            int b;
79            do {
80              if (nodes[a].edges.empty()) { no_mca = true; return; }
81              nodes[a].from = nodes[a].edges.top(); nodes[a].edges.pop();
82              b = nodes[a].from.id;
83            } while (topnode(a) == topnode(b));
84            if (nodes[b].fin) unfold();
85            else if (nodes[b].done) {
86              contract(b);
87              stk.push(nodes.size() - 1);
88              a = nodes.size() - 1;
89            }
90            else {
91              nodes[b].done = true;
92              stk.push(b);
93              a = b;
94            }
95          } while (stk.size());
96        }
97      }
98      return;
99    }
100   vector<P> get_tree_idpair() {
101     if (no_mca) return{};
102     vector<P> ret;
103     Loop(i, n) {
104       if (i != root) ret.push_back({ nodes[i].from.id, i });
105     }
106     return ret;
107   }
108   ll get_weight() {
109     if (no_mca) return -1;
110     ll ret = 0;
111     Loop(i, n) {
112       if (i != root) ret += nodes[i].from.cost;
113     }
114     return ret;
115   }
116 };
```

```cpp
class Connected_Components {
private:
  int n;
  vi cc_gid;
  vvi ccs;
  void dfs(int a, int gid, const vvi &lst) {
    cc_gid[a] = gid;
    ccs.back().push_back(a);
    Foreach(b, lst[a]) {
      if (cc_gid[b] == -1) dfs(b, gid, lst);
    }
  }
public:
  Connected_Components(const vvi &lst) {
    n = lst.size();
    cc_gid = vi(n, -1);
    int gid = 0;
    Loop(i, n) {
      if (cc_gid[i] == -1) {
        ccs.push_back({});
        dfs(i, gid, lst);
        gid++;
      }
    }
  }
  vi get_cc_gid() {
    return cc_gid;
  }
  vvi get_ccs() {
    return ccs;
  }
};
```

```cpp
class Dijkstra {
private:
  struct node {
    int id; bool done; vi to; vll cst; int from; ll d;
  };
  struct pq_t {
    int id; ll d;
    bool operator<(const pq_t & another) const {
      return d != another.d ? d > another.d : id > another.id;
    }
  };
  vector<node> nodes;
  int n, m, source;
public:
  Dijkstra(const vvi &lst, const vvll &cst, int start) {
    n = lst.size();
    nodes.resize(n);
    Loop(i, n) nodes[i] = { i, false,{},{}, -1, LLONG_MAX };
    Loop(i, n) {
      Loop(j, lst[i].size()) {
        nodes[i].to.push_back(lst[i][j]);
        nodes[i].cst.push_back(cst[i][j]);
      }
    }
    source = start;
    nodes[source].d = 0;
    priority_queue<pq_t> pq;
    pq.push({ nodes[source].id, nodes[source].d });
    while (pq.size()) {
      int a = pq.top().id;
      pq.pop();
      if (nodes[a].done) continue;
      nodes[a].done = true;
      Loop(j, nodes[a].to.size()) {
        int b = nodes[a].to[j];
        if (nodes[b].done) continue;
        ll buf = nodes[a].d + nodes[a].cst[j];
        if (buf < nodes[b].d) {
          nodes[b].d = buf;
          nodes[b].from = a;
          pq.push({ b, nodes[b].d });
        }
      }
    }
    return;
  }
  vi get_path(int v) {
    stack<int> stk;
    stk.push(v);
    int a = v;
    while (nodes[a].from != -1) {
      stk.push(nodes[a].from);
      a = nodes[a].from;
    }
    if (a != source) return {};
    vi ret;
    while (stk.size()) {
      ret.push_back(stk.top());
      stk.pop();
    }
    return ret;
  }
  ll get_dist(int v) {
    return nodes[v].d;
  }
};
```

```cpp
// mx + ny = gcd(m, n), runtime error for (m, n) = (0, 0)
ll ex_euclid(ll m, ll n, ll &x, ll &y) {
  if (n == 0) { x = 1; y = 0; return m; }
  ll g = ex_euclid(n, m % n, y, x);
  y -= m / n * x;
  return g;
}

// In case when there is range restriction for (x, y)
class Extended_Euclid {
private:
  bool inrange(ll x, ll y, Pll x_rng, Pll y_rng) {
    if (x_rng.fst <= x && x <= x_rng.snd && y_rng.fst <= y && y <= y_rng.snd) return true;
    else return false;
  }
  bool subst_d(ll &x, ll &y, ll d, Pll x_rng, Pll y_rng) {
    ll xc = x, yc = y;
    Loop(k, 3) {
      x = xc + n / g * (d + k - 1);
      y = yc - m / g * (d + k - 1);
      if (inrange(x, y, x_rng, y_rng)) return true;
    }
    return false;
  }
  ll m, n, g, x, y;
public:
  Extended_Euclid(ll m, ll n) {
    this->m = m;
    this->n = n;
    vll q;
    g = gcd(m, n);
    ex_euclid(m, n, x, y);
  }
  bool solve(ll &x, ll &y, ll z, Pll x_rng = { LLONG_MIN, LLONG_MAX }, Pll y_rng = { LLONG_MIN, LLONG_MAX }) {
    if (z % g != 0) return false;
    else {
      ll q = z / g;
      x = this->x * q;
      y = this->y * q;
      if (inrange(x, y, x_rng, y_rng)) return true;
      if (x_rng.fst != LLONG_MIN) {
        ll d = (x_rng.fst - x) / (n / g);
        if (subst_d(x, y, d, x_rng, y_rng)) return true;
      }
      if (x_rng.snd != LLONG_MAX) {
        ll d = (x_rng.snd - x) / (n / g);
        if (subst_d(x, y, d, x_rng, y_rng)) return true;
      }
      if (y_rng.fst != LLONG_MIN) {
        ll d = (y_rng.fst - y) / (m / g);
        if (subst_d(x, y, -d, x_rng, y_rng)) return true;
      }
      if (x_rng.snd != LLONG_MAX) {
        ll d = (y_rng.snd - y) / (m / g);
        if (subst_d(x, y, -d, x_rng, y_rng)) return true;
      }
      return false;
    }
  }
};
```

```cpp
class Finding_Arts {
private:
  struct node {
    int id; bool done; vi to; int from; int pre; int low;
  };
  vector<node> nodes;
  int n;
  int ord;
  vi arts;
  void lowlink_dfs(int a, bool isroot) {
    nodes[a].done = true;
    nodes[a].pre = nodes[a].low = ord;
    ord++;
    int cnt = 0;
    Loop(i, nodes[a].to.size()) {
      int b = nodes[a].to[i];
      if (b == nodes[a].from) continue;
      if (!nodes[b].done) {
        nodes[b].from = a;
        lowlink_dfs(b, false);
        nodes[a].low = min(nodes[a].low, nodes[b].low);
        if (nodes[a].pre <= nodes[b].low) cnt++;
      }
      else {
        nodes[a].low = min(nodes[a].low, nodes[b].pre);
      }
    }
    if (cnt > (isroot ? 1 : 0)) arts.push_back(a);
    return;
  }
public:
  Finding_Arts(const vvi &lst) {
    n = lst.size();
    nodes.resize(n);
    Loop(i, n) nodes[i] = { i, false, {}, -1, -1, -1 };
    Loop(i, n) {
      Foreach(j, lst[i]) {
        nodes[i].to.push_back(j);
      }
    }
    ord = 0;
    Loop(i, nodes.size()) {
      if (!nodes[i].done) lowlink_dfs(i, true);
    }
    sort(arts.begin(), arts.end());
  }
  vi get_arts() {
    return arts;
  }
};
```

```cpp
class Finding_Bridges {
private:
  struct node {
    int id; bool done; vi to; int from; int pre; int low;
  };
  vector<node> nodes;
  int n, m;
  int ord;
  vector<P> result;
  void lowlink_dfs(int a) {
    nodes[a].done = true;
    nodes[a].pre = nodes[a].low = ord;
    ord++;
    Loop(i, nodes[a].to.size()) {
      int b = nodes[a].to[i];
      if (b == nodes[a].from) continue;
      if (!nodes[b].done) {
        nodes[b].from = a;
        lowlink_dfs(b);
        nodes[a].low = min(nodes[a].low, nodes[b].low);
        if (nodes[a].pre < nodes[b].low) {
          if (a < b) result.push_back({ a,b });
          else result.push_back({ b,a });
        }
      }
      else {
        nodes[a].low = min(nodes[a].low, nodes[b].pre);
      }
    }
    return;
  }
public:
  Finding_Bridges(const vvi &lst) {
    n = lst.size();
    nodes.resize(n);
    Loop(i, n) nodes[i] = { i, false,{}, -1, -1, -1 };
    Loop(i, n) {
      Foreach(j, lst[i]) {
        nodes[i].to.push_back(j);
      }
    }
    ord = 0;
    Loop(i, nodes.size()) {
      if (!nodes[i].done) lowlink_dfs(i);
    }
    sort(result.begin(), result.end());
  }
  vector<P> get_bridges() {
    return result;
  }
};
```

```cpp
class LIS {
  vll result;
  vll id_result;
  int n;
public:
  LIS(vll a, bool strict_flag) {
    int n = a.size();
    vll record;
    vi id_record, parents(n, -1);
    Loop(i, n) {
      auto itr = strict_flag ? lower_bound(record.begin(), record.end(), a[i])
        : upper_bound(record.begin(), record.end(), a[i]);
      if (itr == record.end()) {
        record.push_back(a[i]);
        id_record.push_back(i);
        itr = record.end();
        itr--;
      }
      else {
        *itr = a[i];
        id_record[distance(record.begin(), itr)] = i;
      }
      if (itr != record.begin()) {
        parents[i] = id_record[distance(record.begin(), itr) - 1];
      }
    }
    result = {};
    id_result = {};
    int focus = id_record.back();
    do {
      id_result.push_back(focus);
      result.push_back(a[focus]);
      focus = parents[focus];
    } while (focus != -1);
    reverse(result.begin(), result.end());
    reverse(id_result.begin(), id_result.end());
  }
  vll get_lis() {
    return result;
  }
  vll get_lisid() {
    return id_result;
  }
  int get_lisn() {
    return result.size();
  }
};


```

```cpp
class Max_Clique {
private:
  static int max_clique_rec(const vvi &mx, unordered_map<ll, int> &mp, ll mask) {
    if (mask != 0 && mp[mask] == 0) {
      ll x = mask & -mask;
      int id = int(log2(x));
      int r0 = max_clique_rec(mx, mp, mask ^ x);
      ll y = 0;
      for (int j = id + 1; j < mx[id].size(); ++j) {
        if (mask & (ll(mx[id][j]) << j)) y |= (1LL << j);
      }
      int r1 = max_clique_rec(mx, mp, y) + 1;
      mp[mask] = max(r0, r1);
    }
    return mp[mask];
  }
public:
  // O(n*2^(n/2))
  static int max_clique(const vvi &mx) {
    int n = int(mx.size());
    unordered_map<ll, int> mp;
    return max_clique_rec(mx, mp, (1LL << n) - 1);
  }
};
```

```cpp
class Max_Queue {
private:
  stack<PII> stk0, stk1;
public:
  void push(ll x) {
    ll y = stk1.size() ? stk1.top().second : LLONG_MIN;
    stk1.push({ x, max(x, y) });
  }
  void pop() {
    if (!stk0.size()) {
      while (stk1.size()) {
        ll x = stk1.top().first;
        ll y = stk0.size() ? stk0.top().second : LLONG_MIN;
        stk0.push({ x, max(x, y) });
        stk1.pop();
      }
    }
    stk0.pop();
  }
  size_t size() {
    return stk0.size() + stk1.size();
  }
  void clear() {
    while (stk0.size()) stk0.pop();
    while (stk1.size()) stk1.pop();
  }
  ll get_max() {
    ll x = LLONG_MIN, y = LLONG_MIN;
    if (stk0.size()) x = stk0.top().second;
    if (stk1.size()) y = stk1.top().second;
    return max(x, y);
  }
};
```

```cpp
class Maxflow {
private:
  struct edge_t {
    int cap;
  };
  int n, source, sink;
  int result;
  vector<bool> done;
  vector<unordered_map<int, edge_t>> lst;
  int dfs(int a, int t) {
    if (a == t) return 1;
    done[a] = true;
    Loopitr(itr, lst[a]) {
      int b = itr->fst;
      int cap = itr->snd.cap;
      if (!done[b] && cap > 0) {
        if (dfs(b, t)) {
          lst[a][b].cap--;
          lst[b][a].cap++;
          return 1;
        }
      }
    }
    return 0;
  }
  int run_flow(int s, int t, int f) {
    int ret = 0;
    Loop(i, f) {
      done = vector<bool>(n, false);
      if (dfs(s, t)) ret++;
      else break;
    }
    return ret;
  }
public:
  Maxflow(const vvi &lst, const vvi &cap, int s, int t) {
    n = lst.size();
    this->lst.resize(n);
    Loop(i, n) {
      Loop(j, lst[i].size()) {
        this->lst[i][lst[i][j]].cap += cap[i][j];
        this->lst[lst[i][j]][i].cap += 0;
      }
    }
    source = s;
    sink = t;
    result = 0;
    update();
  }
  void add_cap(int s, int t, int dcap, bool update_flag = true) {
    lst[s][t].cap += dcap;
    // program not be ensured when cap. becomes negative
    if (lst[s][t].cap < 0) {
      int df = -lst[s][t].cap;
      run_flow(s, source, df);
      run_flow(sink, t, df);
      lst[s][t].cap += df;
      lst[t][s].cap -= df;
      result -= df;
    }
    if (update_flag) update();
  }
  void update() {
    result += run_flow(source, sink, INT_MAX);
  }
  int get_maxflow() {
    return result;
  }
};
```

```cpp
class Mincostflow {
private:
  struct edge {
    int eid, from, to;
    ll cap, cost;
  };
  struct node {
    int id; ll d; int from_eid; vector<int> to_eids;
  };
  struct pq_t {
    int id; ll d;
    bool operator<(const pq_t & another) const {
      return d != another.d ? d > another.d : id > another.id;
    }
  };
  int dual_eid(int eid) {
    if (eid < m) return eid + m;
    else return eid - m;
  }
  vector<node> nodes;
  vector<edge> edges;
  int n, m;
  int source, sink;
  bool overflow;
public:
  Mincostflow(const vvi &lst, const vvll &cap, const vvll &cst, int s, int t) {
    n = lst.size();
    nodes.resize(n);
    Loop(i, n) nodes[i] = { i, LLONG_MAX, -1, {} };
    int eid = 0;
    Loop(i, n) {
      Loop(j, lst[i].size()) {
        nodes[i].to_eids.push_back(eid);
        edges.push_back({ eid, i, lst[i][j], cap[i][j], cst[i][j] });
        eid++;
      }
    }
    m = eid;
    Loop(i, n) {
      Loop(j, lst[i].size()) {
        nodes[lst[i][j]].to_eids.push_back(eid);
        edges.push_back({ eid, lst[i][j], i, 0, -cst[i][j] });
        eid++;
      }
    }
    source = s;
    sink = t;
    overflow = false;
  }
  bool add_flow(ll f) {
    if (overflow) return false;
    while (f > 0) {
      Loop(i, n) {
        nodes[i].d = LLONG_MAX;
        nodes[i].from_eid = -1;
      }
      // Bellmanford
      nodes[source].d = 0;
      Loop(k, n) {
        Loop(i, n) {
          int a = i;
          if (nodes[a].d == LLONG_MAX) continue;
          Foreach(eid, nodes[a].to_eids) {
            if (edges[eid].cap == 0) continue;
            int b = edges[eid].to;
            if (nodes[a].d + edges[eid].cost < nodes[b].d) {
              nodes[b].d = nodes[a].d + edges[eid].cost;
              nodes[b].from_eid = eid;
              if (k == n - 1) {
                return false;
              }
```

```
72              }
73            }
74          }
75        }
76        if (nodes[sink].d == LLONG_MAX) return false;
77        int a = sink;
78        ll df = f;
79        while (a != source) {
80          df = min(df, edges[nodes[a].from_eid].cap);
81          a = edges[nodes[a].from_eid].from;
82        }
83        a = sink;
84        while (a != source) {
85          edges[nodes[a].from_eid].cap -= df;
86          edges[dual_eid(nodes[a].from_eid)].cap += df;
87          a = edges[nodes[a].from_eid].from;
88        }
89        f -= df;
90      }
91      return true;
92    }
93    vll get_eid_flow() {
94      vll ret(m, -1);
95      if (overflow) return ret;
96      Loop(i, m) {
97        ret[i] = edges[i + m].cap;
98      }
99      return ret;
100   }
101   ll get_flow() {
102     if (overflow) return -1;
103     ll ret = 0;
104     Foreach(eid, nodes[sink].to_eids) {
105       if (eid >= m) ret += edges[eid].cap;
106     }
107     return ret;
108   }
109   ll get_cost() {
110     if (overflow) return -1;
111     ll ret = 0;
112     Loop(i, m) {
113       ret += edges[i].cost * edges[i + m].cap;
114     }
115     return ret;
116   }
117 };
```

```cpp
class Nim {
private:
  bool result;
public:
  Nim(vll a) {
    ll x = 0;
    Loop(i, a.size()) x ^= a[i];
    if (x != 0) result = true;
    else result = false;
  }
  bool get_result() {
    return result;
  }
  string get_winner(string player1, string player2) {
    if (result) return player1;
    else return player2;
  }
};

// Grundy number pseudo code
class Grundy {
private:
  bool result;
  vi grundies;
  vi diff;
public:
  void make_grundies(int k) {
    // memoization
    if (grundies[k] != -1) return;
    else {
      set<int> s;
      Loop(j, diff.size()) {
        // transition rule
        int index = k - diff[j];
        if (index >= 0) {
          if (grundies[index] == -1) make_grundies(index);
          s.insert(grundies[index]);
        }
      }
      int c = 0;
      while (s.find(c) != s.end()) c++;
      grundies[k] = c;
      return;
    }
  }
  Grundy(vi states, vi diff) {
    Grundy::diff = diff;
    // calculate all possible grundy numbers
    int grundy_size = 1000;
    grundies = vi(grundy_size, -1);
    Loop(i, grundy_size) make_grundies(i);
    // decide the grundy number in each states
    vll x(states.size());
    Loop(i, states.size()) x[i] = grundies[states[i]];
    // return to Nim
    Nim *nim = new Nim(x);
    result = nim->get_result();
  }
  bool get_result() {
    return result;
  }
  string get_winner(string player1, string player2) {
    if (result) return player1;
    else return player2;
  }
};

```

```cpp
template<typename val_t>
class Partial_Combination {
private:
  int n;
  vector<vector<val_t>> result;
  vvi combs; // iCj
  void core_func(const vector<val_t> &a, int n, int r, int start) {
    if (r == 0 || n < r) return;
    Loop(i, combs[n - 1][r - 1]) {
      result[start + i].push_back(a[Partial_Combination::n - n]);
    }
    if (n > 1) {
      core_func(a, n - 1, r - 1, start);
      core_func(a, n - 1, r, start + combs[n - 1][r - 1]);
    }
  }
  void make_combs(int n) {
    combs = vvi(n + 1, vi(n + 1));
    Loop(i, n + 1) {
      combs[i][0] = 1;
      Loop1(j, i) {
        combs[i][j] = combs[i - 1][j - 1] + combs[i - 1][j];
      }
    }
  }
public:
  vector<vector<val_t>> get_partial_combination(const vector<val_t> &a, int r) {
    n = int(a.size());
    if (n < r) return {};
    make_combs(n);
    result = vector<vector<val_t>>(combs[n][r]);
    core_func(a, n, r, 0);
    return result;
  }
};

class Partial_Combination_Bitmask {
private:
  int n;
  vll result;
  vvi combs; // iCj
  void core_func(const ll &a, int n, int r, int start) {
    if (r == 0 || n < r) return;
    ll x = a & -a;
    Loop(i, combs[n - 1][r - 1]) {
      result[start + i] += x;
    }
    if (n > 1) {
      core_func(a - x, n - 1, r - 1, start);
      core_func(a - x, n - 1, r, start + combs[n - 1][r - 1]);
    }
  }
  void make_combs(int n) {
    combs = vvi(n + 1, vi(n + 1));
    Loop(i, n + 1) {
      combs[i][0] = 1;
      Loop1(j, i) {
        combs[i][j] = combs[i - 1][j - 1] + combs[i - 1][j];
      }
    }
  }
public:
  vll get_partial_combination(int n, int r) {
    this->n = n;
    if (n < r) return {};
    make_combs(n);
    result = vll(combs[n][r]);
    ll a = (1LL << n) - 1;
    core_func(a, n, r, 0);
    return result;
  }
```

```
72  };
```

```cpp
template<typename val_t>
class Partial_Permutation {
private:
  int n;
  vector<bool> used;
  vector<vector<val_t>> result;
  vvi facts; // iPj
  void core_func(const vector<val_t> &a, int n, int r, int start) {
    if (r == 0 || n < r) return;
    int m = facts[n - 1][r - 1];
    int cnt = 0;
    Loop(i, Partial_Permutation::n) {
      if (!used[i]) {
        Loop(j, m) {
          result[start + m * cnt + j].push_back(a[i]);
        }
        used[i] = true;
        core_func(a, n - 1, r - 1, start + m * cnt);
        used[i] = false;
        cnt++;
      }
    }
  }
  void make_facts(int n) {
    facts = vvi(n + 1, vi(n + 1));
    Loop(i, n + 1) {
      facts[i][0] = 1;
      Loop(j, i) {
        facts[i][j + 1] = facts[i][j] * (i - j);
      }
    }
  }
public:
  vector<vector<val_t>> get_partial_permutation(const vector<val_t> &a, int r) {
    n = int(a.size());
    if (n < r) return {};
    used = vector<bool>(n, false);
    make_facts(n);
    result = vector<vector<val_t>>(facts[n][r]);
    core_func(a, n, r, 0);
    return result;
  }
};


class Partial_Permutation_String {
private:
  int n;
  string a;
  vector<bool> used;
  vector<string> result;
  vvi facts; // iPj
  void core_func(const string &a, int n, int r, int start) {
    if (r == 0 || n < r) return;
    int m = facts[n - 1][r - 1];
    int cnt = 0;
    Loop(i, Partial_Permutation_String::n) {
      if (!used[i]) {
        Loop(j, m) {
          result[start + m * cnt + j] += a[i];
        }
        used[i] = true;
        core_func(a, n - 1, r - 1, start + m * cnt);
        used[i] = false;
        cnt++;
      }
    }
  }
  void make_facts(int n) {
    facts = vvi(n + 1, vi(n + 1));
    Loop(i, n + 1) {
```

```
72        facts[i][0] = 1;
73        Loop(j, i) {
74          facts[i][j + 1] = facts[i][j] * (i - j);
75        }
76      }
77    }
78  public:
79    vector<string> get_partial_permutation(const string &a, int r) {
80      n = int(a.size());
81      if (n < r) return {};
82      used = vector<bool>(n, false);
83      make_facts(n);
84      result = vector<string>(facts[n][r]);
85      core_func(a, n, r, 0);
86      return result;
87    }
88  };
```

```cpp
class Prim {
private:
  struct node {
    int id; bool done; vi to; vll cst; int from; ll d;
  };
  struct pq_t {
    int id; ll d;
    bool operator<(const pq_t & another) const {
      return d != another.d ? d > another.d : id > another.id;
    }
  };
  vector<node> nodes;
  int n, m;
public:
  Prim(const vvi &lst, const vvll &cst) {
    n = lst.size();
    nodes.resize(n);
    Loop(i, n) nodes[i] = { i, false,{},{}, -1, LLONG_MAX };
    Loop(i, n) {
      Loop(j, lst[i].size()) {
        nodes[i].to.push_back(lst[i][j]);
        nodes[i].cst.push_back(cst[i][j]);
      }
    }
    nodes[0].d = 0;
    priority_queue<pq_t> pq;
    pq.push({ nodes[0].id, nodes[0].d });
    while (pq.size()) {
      int a = pq.top().id;
      pq.pop();
      if (nodes[a].done) continue;
      nodes[a].done = true;
      Loop(j, nodes[a].to.size()) {
        int b = nodes[a].to[j];
        if (nodes[b].done) continue;
        ll buf = nodes[a].cst[j];
        if (buf < nodes[b].d) {
          nodes[b].d = buf;
          nodes[b].from = a;
          pq.push({ b, nodes[b].d });
        }
      }
    }
    return;
  }
  vector<P> get_result() {
    vector<P> ret;
    Loop1(i, n - 1) {
      int a = i;
      int b = nodes[i].from;
      if (a > b) swap(a, b);
      ret.push_back({ a, b });
    }
  }
  ll get_weight() {
    ll ret = 0;
    Loop(i, n) {
      ret += nodes[i].d;
    }
    return ret;
  }
};
```

```cpp
class Random {
private:
  mt19937 *mt;
  uniform_int_distribution<> *distr;
public:
  // uniform int distribution of [0, m)
  Random(int m) {
    mt = new mt19937(unsigned(time(NULL)));
    distr = new uniform_int_distribution<>(0, m - 1);
  }
  int get() {
    return (*distr)(*mt);
  }
};
```

```cpp
// include strongly connected components

struct cnf2_t {
  int n; // size of variables
  struct literal_t {
    int index;
    bool neg;
  };
  struct clause_t {
    literal_t x, y;
  };
  vector<clause_t> L;
};

class SAT2 {
private:
  int n;
  bool fail_flag;
  vvi sccs;
  vi scc_gid;
  vector<bool> result;
  int inv(int id) {
    return (id + n) % (n * 2);
  }
public:
  SAT2(cnf2_t CNF) {
    vvi lst(n * 2);
    Loop(i, CNF.L.size()) {
      lst[CNF.L[i].x.index + (CNF.L[i].x.neg ? 0 : n)].push_back(CNF.L[i].y.index + (CNF.L[i].y.neg ? n : 0));
      lst[CNF.L[i].y.index + (CNF.L[i].y.neg ? 0 : n)].push_back(CNF.L[i].x.index + (CNF.L[i].x.neg ? n : 0));
    }
    Strongly_Connected_Components *scc = new Strongly_Connected_Components(lst);
    sccs = scc->get_sccs();
    scc_gid = scc->get_scc_gid();
    fail_flag = false;
    result.resize(n);
    Loop(i, n) {
      if (scc_gid[i] > scc_gid[inv(i)]) result[i] = true;
      else if (scc_gid[i] < scc_gid[inv(i)]) result[i] = false;
      else {
        result.clear();
        fail_flag = true;
        return;
      }
    }
    return;
  }
  bool is_satisfiable() {
    return !fail_flag;
  }
  vector<bool> get_result() {
    return result;
  }
};
```

```cpp
class SegTree {
private:
  struct val_t {
    bool enable;
    ll upd, add, min, max, sum;
  };
  int n, N; // n is the original size, while N is the extended size
  int base;
  vector<val_t> nodes;
  int left_of(int id) {
    if (id >= base) return -1;
    else return id * 2 + 1;
  }
  int right_of(int id) {
    if (id >= base) return -1;
    else return id * 2 + 2;
  }
  int parent_of(int id) {
    if (id == 0) return -1;
    else return (id - 1) >> 1;
  }
  void merge(int id, int id_l, int id_r) {
    nodes[id].min = min(nodes[id_l].min + nodes[id_l].add, nodes[id_r].min + nodes[id_r].add);
    nodes[id].max = max(nodes[id_l].max + nodes[id_l].add, nodes[id_r].max + nodes[id_r].add);
    nodes[id].sum = nodes[id_l].sum + nodes[id_l].add * cover_size(id_l)
      + nodes[id_r].sum + nodes[id_r].add * cover_size(id_r);
  }
  void lazy(int id) {
    if (id >= base) return;
    int id_l = left_of(id);
    int id_r = right_of(id);
    if (nodes[id].enable) {
      ll upd = nodes[id].upd + nodes[id].add;
      nodes[id_l] = { true, upd, 0, upd, upd, upd * cover_size(id_l) };
      nodes[id_r] = { true, upd, 0, upd, upd, upd * cover_size(id_r) };
      nodes[id] = { false, 0, 0, upd, upd, upd * cover_size(id) };
    }
    else {
      nodes[id_l].add += nodes[id].add;
      nodes[id_r].add += nodes[id].add;
      nodes[id].add = 0;
      merge(id, id_l, id_r);
    }
  }
  enum change_t {
    UPD, ADD
  };
  void change_rec(int s, int t, int l, int r, int id, ll x, change_t op) {
    if (s == l && t == r) {
      if (op == UPD) nodes[id] = { true, x, 0, x, x, x * cover_size(id) };
      else if (op == ADD) nodes[id].add += x;
    }
    else {
      lazy(id);
      int m = (l + r) / 2;
      int id_l = left_of(id);
      int id_r = right_of(id);
      if (s < m && m < t) {
        change_rec(s, m, l, m, id_l, x, op);
        change_rec(m, t, m, r, id_r, x, op);
      }
      else if (s < m) {
        change_rec(s, t, l, m, id_l, x, op);
      }
      else if (m < t) {
        change_rec(s, t, m, r, id_r, x, op);
      }
      merge(id, id_l, id_r);
    }
  }
  enum solve_t {
```

```cpp
    MIN, MAX, SUM
  };
  ll solve_rec(int s, int t, int l, int r, int id, solve_t op) {
    ll v = 0;
    if (s == l && t == r) {
      if (op == MIN) v = nodes[id].min;
      else if (op == MAX) v = nodes[id].max;
      else if (op == SUM) v = nodes[id].sum;
    }
    else {
      lazy(id);
      int m = (l + r) / 2;
      int id_l = left_of(id);
      int id_r = right_of(id);
      if (s < m && m < t) {
        ll v0 = solve_rec(s, m, l, m, id_l, op);
        ll v1 = solve_rec(m, t, m, r, id_r, op);
        if (op == MIN) v = min(v0, v1);
        else if (op == MAX) v = max(v0, v1);
        else if (op == SUM) v = v0 + v1;
      }
      else if (s < m) {
        v = solve_rec(s, t, l, m, id_l, op);
      }
      else if (m < t) {
        v = solve_rec(s, t, m, r, id_r, op);
      }
    }
    if (op == MIN) v += nodes[id].add;
    else if (op == MAX) v += nodes[id].add;
    else if (op == SUM) v += nodes[id].add * (t - s);
    return v;
  }
public:
  SegTree(int n, ll init) {
    this->n = n;
    N = (int)pow(2, ceil(log2(n)));
    base = N - 1;
    nodes = vector<val_t>(base + N, { false, 0, 0, LLONG_MAX, LLONG_MIN, 0 });
    upd(0, n, init);
  }
  int cover_size(int id) {
    int cnt = 1;
    while (left_of(id) != -1) {
      id = left_of(id);
      cnt *= 2;
    }
    int l = id - base;
    int r = min(l + cnt, n);
    return max(0, r - l);
  }
  void upd(int s, int t, ll x) {
    change_rec(s, t, 0, N, 0, x, UPD);
  }
  void add(int s, int t, ll x) {
    change_rec(s, t, 0, N, 0, x, ADD);
  }
  ll minof(int s, int t) {
    return solve_rec(s, t, 0, N, 0, MIN);
  }
  ll maxof(int s, int t) {
    return solve_rec(s, t, 0, N, 0, MAX);
  }
  ll sumof(int s, int t) {
    return solve_rec(s, t, 0, N, 0, SUM);
  }
};
```

```cpp
class Strongly_Connected_Components {
private:
  struct node {
    int id; bool done; vi to; int from;
  };
  vector<node> nodes[2];
  int n;
  stack<int> stk;
  vvi sccs;
  vi scc_gid;
  // u means the direction
  void scc_dfs(int a, int u) {
    nodes[u][a].done = true;
    Loop(i, nodes[u][a].to.size()) {
      int b = nodes[u][a].to[i];
      if (b == nodes[u][a].from) continue;
      if (!nodes[u][b].done) {
        nodes[u][b].from = a;
        scc_dfs(b, u);
      }
    }
    if (u == 0) stk.push(a);
    else {
      sccs.back().push_back(a);
    }
    return;
  }
public:
  Strongly_Connected_Components(const vvi &lst) {
    n = lst.size();
    Loop(i, 2) nodes[i].resize(n);
    Loop(i, 2) {
      Loop(j, n) {
        nodes[i][j] = { i, false,{}, -1 };
      }
    }
    Loop(i, n) {
      Foreach(j, lst[i]) {
        nodes[0][i].to.push_back(j);
        nodes[1][j].to.push_back(i);
      }
    }
    Loop(i, n) {
      if (!nodes[0][i].done) scc_dfs(i, 0);
    }
    while (stk.size()) {
      int a = stk.top(); stk.pop();
      if (!nodes[1][a].done) {
        sccs.push_back({});
        scc_dfs(a, 1);
        sort(sccs.back().begin(), sccs.back().end());
      }
    }
    return;
  }
  // already in topological order
  vvi get_sccs() {
    return sccs;
  }
  vi get_scc_gid() {
    if (scc_gid.empty()) {
      scc_gid.resize(n);
      Loop(i, sccs.size()) {
        Loop(j, sccs[i].size()) {
          scc_gid[sccs[i][j]] = i;
        }
      }
    }
    return scc_gid;
  }
};
```

```cpp
class Suffix_Array {
private:
  struct sa_t {
    int r0, r1;
    int p;
    bool operator<(const sa_t &another) const {
      return r0 != another.r0 ? r0 < another.r0 : r1 < another.r1;
    }
  };
public:
  // excluding empty substring
  static vi suffix_array(const string &s) {
    int n = s.length();
    vi ret(n);
    vector<sa_t> a(n); // fst = current rank, snd add rank
    Loop(k, ceillog2(n)) {
      if (k == 0) {
        Loop(i, n) a[i] = { s[i], 0, i };
      }
      else {
        int d = 1 << (k - 1);
        Loop(i, n) {
          if (inrange(a[i].p + d, n)) a[i].r1 = ret[a[i].p + d];
          else a[i].r1 = -1;
        }
      }
      sort(a.begin(), a.end());
      sa_t pre;
      Loop(i, n) {
        if (i > 0 && a[i].r0 == pre.r0 && a[i].r1 == pre.r1) {
          a[i] = { a[i - 1].r0, 0, a[i].p };
        }
        else {
          pre = a[i];
          a[i] = { i, 0, a[i].p };
        }
        ret[a[i].p] = a[i].r0;
      }
    }
    return ret;
  }
};
```

```cpp
class Trie {
private:
  struct node {
    char val; map<char, node*> childs_mp; ll deg; node *parent; int cnt;
  };
  bool erase_leaf(node *ptr) {
    if (ptr->val != '¥0') {
      do {
        char v = ptr->val;
        ptr->cnt--;
        ptr = ptr->parent;
        if (ptr->childs_mp[v]->cnt == 0) {
          delete(ptr->childs_mp[v]);
          ptr->childs_mp.erase(v);
        }
      } while (ptr != root);
      ptr->cnt--;
      return true;
    }
    else return false;
  }
  node *root;
  bool multi_flag;
public:
  Trie(bool multi_flag) {
    root = new node{ '¥0',{}, 0, nullptr, 0 };
    Trie::multi_flag = multi_flag;
  }
  void add(string s) {
    node *a = root;
    Loop(i, s.length()) {
      char c = s[i];
      if (a->childs_mp.find(c) == a->childs_mp.end()) {
        node *node_buf = new node{ c,{},a->deg + 1, a, 0 };
        a->childs_mp[c] = node_buf;
      }
      a->cnt++;
      a = a->childs_mp[c];
    }
    if (a->childs_mp.find('¥0') == a->childs_mp.end()) {
      node *nil = new node{ '¥0',{}, a->deg + 1, a, 0 };
      a->childs_mp['¥0'] = nil;
    }
    a->cnt++;
    a = a->childs_mp['¥0'];
    a->cnt++;
    if (!multi_flag && a->cnt >= 2) erase_leaf(a);
  }
  bool find(string s) {
    node *a = root;
    Loop(i, s.length()) {
      char c = s[i];
      if (a->childs_mp.find(c) == a->childs_mp.end()) return false;
      else a = a->childs_mp[c];
    }
    if (a->childs_mp.find('¥0') != a->childs_mp.end()) return true;
    else return false;
  }
  bool erase(string s) {
    node *a = root;
    Loop(i, s.length()) {
      char c = s[i];
      if (a->childs_mp.find(c) == a->childs_mp.end()) return false;
      else a = a->childs_mp[c];
    }
    if (a->childs_mp.find('¥0') != a->childs_mp.end()) {
      if (erase_leaf(a)) return true;
      else return false;
    }
    else return false;
  }
```

```
72  };
```

```cpp
class Trie2 {
private:
  struct node {
    int val; node* childs[2]; int deg; node *parent; int cnt;
  };
  int height;
  node *root;
  bool multi_flag;
  bool erase_leaf(node *ptr) {
    if (ptr->deg != height) return false;
    while (ptr != root) {
      bool v = ptr->val;
      ptr->cnt--;
      ptr = ptr->parent;
      if (ptr->childs[v]->cnt == 0) {
        delete(ptr->childs[v]);
        ptr->childs[v] = nullptr;
      }
    }
    ptr->cnt--;
    return true;
  }
  int lower_bit(ll x, int bitp) {
    return (x >> bitp) & 1LL;
  }
  int upper_bit(ll x, int bitp) {
    return (x >> (height - 1 - bitp)) & 1LL;
  }
public:
  Trie2(bool multi_flag, int height = 63) {
    Trie2::height = height;
    root = new node{ 0,{ nullptr, nullptr }, 0, nullptr, 0 };
    Trie2::multi_flag = multi_flag;
  }
  void add(ll x) {
    node *a = root;
    Loop(i, height) {
      int v = upper_bit(x, i);
      if (a->childs[v] == nullptr) {
        node *node_buf = new node{ v,{ nullptr, nullptr }, a->deg + 1, a, 0 };
        a->childs[v] = node_buf;
      }
      a->cnt++;
      a = a->childs[v];
    }
    a->cnt++;
    if (!multi_flag && a->cnt >= 2) erase_leaf(a);
    return;
  }
  bool find(ll x) {
    node *a = root;
    Loop(i, height) {
      int v = upper_bit(x, i);
      if (a->childs[v] == nullptr) return false;
      else a = a->childs[v];
    }
    return true;
  }
  bool erase(ll x) {
    node *a = root;
    Loop(i, height) {
      bool v = upper_bit(x, i);
      if (a->childs[v] == nullptr) return false;
      else a = a->childs[v];
    }
    return erase_leaf(a);
  }
  ll prior_find(ll x) {
    node *a = root;
    if (a->cnt == 0) return -1;
    ll ret = 0;
```

```
72      Loop(i, height) {
73        int v = upper_bit(x, i);
74        if (a->childs[v] == nullptr) v ^= 1;
75        ret += ((ll)v << (height - 1 - i));
76        a = a->childs[v];
77      }
78      return ret;
79    }
80  };
```

```cpp
class Union_Find {
private:
  vi p, r, c; // parent, rank, the number of connected components
public:
  Union_Find(int N) {
    p.resize(N);
    r.resize(N);
    c.resize(N);
    Loop(i, N) {
      p[i] = i;
      r[i] = 0;
      c[i] = 1;
    }
  }
  int find(int x) {
    if (p[x] == x) return x;
    else return p[x] = find(p[x]);
  }
  void unite(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y) return;
    if (r[x] == r[y]) r[x]++;
    if (r[x] < r[y]) {
      p[x] = y;
      c[y] += c[x];
    }
    else {
      p[y] = x;
      c[x] += c[y];
    }
  }
  bool is_same(int x, int y) {
    return find(x) == find(y);
  }
  int get_cnt(int x) {
    return c[find(x)];
  }
};
```

```cpp
class Warshallfloyd {
private:
  int n;
  bool negative_cycle;
  vvll table;
public:
  Warshallfloyd(const vvi &lst, const vvll &cst) {
    n = lst.size();
    table = vvll(n, vll(n, LLONG_MAX));
    Loop(i, n) {
      Loop(j, lst[i].size()) {
        table[i][lst[i][j]] = cst[i][j];
      }
    }
    Loop(i, n) table[i][i] = 0;
    Loop(k, n) {
      Loop(i, n) {
        Loop(j, n) {
          if (table[i][k] == LLONG_MAX || table[k][j] == LLONG_MAX) continue;
          table[i][j] = min(table[i][j], table[i][k] + table[k][j]);
        }
      }
    }
    Loop(i, n) {
      if (table[i][i] < 0) {
        negative_cycle = true;
        return;
      }
    }
    negative_cycle = false;
    return;
  }
  vvll get_table() {
    return table;
  }
  bool is_negative_cycle() {
    return negative_cycle;
  }
};
```

```cpp
class Bellmanford {
private:
  struct node {
    int id; bool done; vi to; vll cst; int from; ll d;
  };
  vector<node> nodes;
  int n, m, source;
  bool negative_cycle;
public:
  Bellmanford(const vvi &lst, const vvll &cst, int start) {
    n = lst.size();
    nodes.resize(n);
    Loop(i, n) nodes[i] = { i, false,{},{}, -1, LLONG_MAX };
    Loop(i, n) {
      Loop(j, lst[i].size()) {
        nodes[i].to.push_back(lst[i][j]);
        nodes[i].cst.push_back(cst[i][j]);
      }
    }
    source = start;
    nodes[source].d = 0;
    Loop(k, n) {
      Loop(i, n) {
        int a = i;
        if (nodes[a].d == LLONG_MAX) continue;
        Loop(j, nodes[a].to.size()) {
          int b = nodes[a].to[j];
          if (nodes[a].d + nodes[a].cst[j] < nodes[b].d) {
            nodes[b].d = nodes[a].d + nodes[a].cst[j];
            nodes[b].from = nodes[a].id;
            if (k == n - 1) {
              negative_cycle = true;
              return;
            }
          }
        }
      }
    }
    negative_cycle = false;
    return;
  }
  vi get_path(int v) {
    stack<int> stk;
    stk.push(v);
    int a = v;
    while (nodes[a].from != -1) {
      stk.push(nodes[a].from);
      a = nodes[a].from;
    }
    if (a != source) return{ -1 };
    vi ret;
    while (stk.size()) {
      ret.push_back(stk.top());
      stk.pop();
    }
    return ret;
  }
  ll get_dist(int v) {
    return nodes[v].d;
  }
  bool is_negative_cycle() {
    return negative_cycle;
  }
};
```

```cpp
template <class val_t>
class kdTree {
private:
  using vval_t = vector<val_t>;
  struct node {
    int id;
    int deg;
    vval_t val;
    node *parent;
    node *child_l, *child_r;
    vval_t range_l, range_r;
  };
  int dimension; // dimension
  int n; // the number of nodes
  node *root; // the root of the tree
  node *nil; // the node for leaves of the tree
  struct idval_t {
    int id;
    vval_t val;
  };
  vector<idval_t> ary;
  inline void update_cover_range(node *focus, node* target) {
    if (target == nil) return;
    else {
      Loop(i, dimension) {
        focus->range_l[i] = min(focus->range_l[i], target->range_l[i]);
        focus->range_r[i] = max(focus->range_r[i], target->range_r[i]);
      }
      return;
    }
  }
  node* build_kdTree_rec(node *parent, int l, int r, int depth) {
    if (r - l == 0) return nil;
    node *ret = new node;
    int axis = depth % dimension;
    int mid = (l + r) / 2;
    nth_element(ary.begin() + l, ary.begin() + mid, ary.begin() + r, [=](const idval_t& l, const idval_t& r) { return l.val[axis] < r.val[axis]; });
    *ret = { ary[mid].id, depth, ary[mid].val, nil, nil, nil, ary[mid].val, ary[mid].val };
    ret->child_l = build_kdTree_rec(ret, l, mid, depth + 1);
    update_cover_range(ret, ret->child_l);
    ret->child_r = build_kdTree_rec(ret, mid + 1, r, depth + 1);
    update_cover_range(ret, ret->child_r);
    return ret;
  }
  inline bool check_crossed_find_range(node *focus, pair<vval_t, vval_t> &range) {
    if (focus == nil) return false;
    Loop(i, dimension) {
      if (range.first[i] <= focus->range_r[i] && focus->range_l[i] <= range.second[i]) continue;
      else return false;
    }
    return true;
  }
  inline bool check_in_range(node *focus, pair<vval_t, vval_t> &range) {
    if (focus == nil) return false;
    Loop(i, dimension) {
      if (range.first[i] <= focus->val[i] && focus->val[i] <= range.second[i]) continue;
      else return false;
    }
    return true;
  }
  void find_in_range_rec(node *focus, pair<vval_t, vval_t> &range, int depth, vi &in_range_list) {
    if (focus == nil) return;
    else {
      int axis = depth % dimension;
      if (check_in_range(focus, range)) in_range_list.push_back(focus->id);
      if (check_crossed_find_range(focus->child_l, range)) {
        find_in_range_rec(focus->child_l, range, depth + 1, in_range_list);
      }
      if (check_crossed_find_range(focus->child_r, range)) {
        find_in_range_rec(focus->child_r, range, depth + 1, in_range_list);
```

```
      }
    }
  }
public:
  kdTree(const vector<vval_t> &A, int dimension) {
    n = (int)A.size();
    this->dimension = dimension;
    ary.resize(n);
    Loop(i, n) ary[i] = { i, A[i] };
    nil = new node;
    root = build_kdTree_rec(nil, 0, n, 0);
    return;
  }
  // return id of vals in [range.first, range.second]
  vi find_in_range(pair<vval_t, vval_t> range) {
    vi ret;
    find_in_range_rec(root, range, 0, ret);
    sort(ret.begin(), ret.end());
    return ret;
  }
};
```

```cpp
vi topological_sort(const vvi &lst) {
  vi ret = {};
  int n = lst.size();
  vi cnt(n);
  Loop(a, n) {
    Foreach(b, lst[a]) cnt[b]++;
  }
  set<int> st;
  Loop(a, n) {
    if (cnt[a] == 0) st.insert(a);
  }
  while (st.size()) {
    auto itr = st.begin(); st.erase(itr);
    int a = *itr;
    ret.push_back(a);
    Foreach(b, lst[a]) {
      cnt[b]--;
      if (cnt[b] == 0) st.insert(b);
    }
  }
  if (ret.size() != n) return {};
  else return ret;
}
```

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

using vi = vector<int>; using vvi = vector<vi>; using vvvi = vector<vvi>;
using ll = long long int;
using vll = vector<ll>; using vvll = vector<vll>; using vvvll = vector<vvll>;
using vd = vector<double>; using vvd = vector<vd>; using vvvd = vector<vvd>;
using P = pair<int, int>;
using Pll = pair<ll, ll>;
using cdouble = complex<double>;

const double eps = 1e-7;
#define Loop(i, n) for(int i = 0; i < int(n); i++)
#define Loopll(i, n) for(ll i = 0; i < ll(n); i++)
#define Loop1(i, n) for(int i = 1; i <= int(n); i++)
#define Loopll1(i, n) for(ll i = 1; i <= ll(n); i++)
#define Loopr(i, n) for(int i = int(n) - 1; i >= 0; i--)
#define Looprll(i, n) for(ll i = ll(n) - 1; i >= 0; i--)
#define Loopr1(i, n) for(int i = int(n); i >= 1; i--)
#define Looprll1(i, n) for(ll i = ll(n); i >= 1; i--)
#define Foreach(buf, container) for(auto buf : container)
#define Loopdiag(i, j, h, w, sum) for(int i = ((sum) >= (h) ? (h) - 1 : (sum)), j = (sum) - i; i >= 0 && j < (w); i--, j++)
#define Loopdiagr(i, j, h, w, sum) for(int j = ((sum) >= (w) ? (w) - 1 : (sum)), i = (sum) - j; j >= 0 && i < (h); j--, i++)
#define Loopdiagsym(i, j, h, w, gap) for (int i = ((gap) >= 0 ? (gap) : 0), j = i - (gap); i < (h) && j < (w); i++, j++)
#define Loopdiagsymr(i, j, h, w, gap) for (int i = ((gap) > (h) - (w) - 1 ? (h) - 1 : (w) - 1 + (gap)), j = i - (gap); i >= 0 && j >= 0; i--, j--)
#define Loopitr(itr, container) for(auto itr = container.begin(); itr != container.end(); itr++)
#define printv(vector) Loop(ex_i, vector.size()) { cout << vector[ex_i] << " "; } cout << endl;
#define printmx(matrix) Loop(ex_i, matrix.size()) { Loop(ex_j, matrix[ex_i].size()) { cout << matrix[ex_i][ex_j] << " "; } cout << endl; }
#define quickio() ios::sync_with_stdio(false); cin.tie(0);
#define bitmanip(m,val) static_cast<bitset<(int)m>>(val)
#define Comp(type_t) bool operator<(const type_t &another) const
#define fst first
#define snd second
#define INF INFINITY
bool feq(double x, double y) { return abs(x - y) <= eps; }
bool inrange(ll x, ll t) { return x >= 0 && x < t; }
bool inrange(vll xs, ll t) { Foreach(x, xs) if (!(x >= 0 && x < t)) return false; return true; }
int ceillog2(ll x) { int ret = 0; x--; while (x > 0) { ret++; x >>= 1; } return ret; }
ll rndf(double x) { return (ll)(x + (x >= 0 ? 0.5 : -0.5)); }
ll floorsqrt(ll x) { ll m = (ll)sqrt((double)x); return m + (m * m <= x ? 0 : -1); }
ll ceilsqrt(ll x) { ll m = (ll)sqrt((double)x); return m + (x <= m * m ? 0 : 1); }
ll rnddiv(ll a, ll b) { return (a / b + (a % b * 2 >= b ? 1 : 0)); }
ll ceildiv(ll a, ll b) { return (a / b + (a % b == 0 ? 0 : 1)); }
ll gcd(ll m, ll n) { if (n == 0) return m; else return gcd(n, m % n); }
ll lcm(ll m, ll n) { return m * n / gcd(m, n); }

/*******************************************************/


```