目次

1 all

1.1 BinaryTrie.cpp

```cpp
template <typename T = int, typename D = int>
struct BinaryTrie {
  struct Node {
    Node *par, *to[2];
    D cnt;
    Node() : par(nullptr), to{nullptr, nullptr}, cnt(D(0)) {}
  };
  int max_dep;
  Node *root;
  BinaryTrie(int _max_dep = 31) : max_dep(_max_dep), root(new
  ↪  Node()) {}
  constexpr void insert(T x, D num = 1, T xor_val = 0) noexcept {
    x ^= xor_val;
    Node *v = root;
    v->cnt += num;
    for (int i = max_dep - 1; i >= 0; --i) {
      int b = x >> i & 1;
      if (!(v->to[b])) {
        v->to[b] = new Node();
        v->to[b]->par = v;
      }
      v = v->to[b];
      v->cnt += num;
    }
  }
  constexpr void erase(const T &x, D num = 1, T xor_val = 0)
  ↪  noexcept {
    auto v = find(x ^ xor_val);
    if (!v) return;
    num = min(num, v->cnt);
    v->cnt -= num;
    for (int i = 0; i < max_dep; ++i) {
      int b = x >> i & 1;
      auto p = v->par;
      if (!(v->cnt)) p->to[b] = nullptr, v->par = nullptr;
      v = p, v->cnt -= num;
    }
  }
  constexpr D count(T x, T xor_val = 0) noexcept {
    auto v = find(x ^ xor_val);
    return v ? v->cnt : 0;
  }
  constexpr Node *find(T x) noexcept {
    Node *v = root;
    for (int i = max_dep - 1; i >= 0; --i) {
      int b = x >> i & 1;
      if (!(v->to[b])) return nullptr;
      v = v->to[b];
    }
    return v;
  }
  //   0-inxeded
  constexpr T kth_element(D k, T xor_val = 0) const noexcept {
    assert(k < root->cnt);
    Node *v = root;
    T res = 0;
    for (int i = max_dep - 1; i >= 0; --i) {
      int b = (xor_val >> i & 1);
      if (!(v->to[b]) || v->to[b]->cnt <= k) {
        if (v->to[b]) k -= v->to[b]->cnt;
        b ^= 1;
      }
      res <<= 1;
      res |= b;
      v = v->to[b];
    }
    assert(v->cnt > k);
    return res ^ xor_val;
  }
  constexpr T min_element(T xor_val = 0) const noexcept {
    return kth_element(0, xor_val);
  }
  constexpr T max_element(T xor_val = 0) const noexcept {
    return kth_element(root->cnt - 1, xor_val);
  }
};
```

1.2 BIT.cpp

```cpp
// 0-indexed
template <class T>
struct BIT {
  int treesize;
  vector<T> lst;
  // constructor
  BIT(int newn = 0) : treesize(newn), lst(newn + 1, 0) {}
  // a_place += num
  void add(int place, T num) {
    ++place;
```

```
    while (place <= treesize) {
      lst[place] += num;
      place += place & -place;
    }
  }
  // sum between [0,place)
  T sum(int place) {
    T res = 0;
    while (place > 0) {
      res += lst[place];
      place -= place & -place;
    }
    return res;
  }
  // sum [l,r)
  T sum(int left, int right) { return sum(right) - sum(left); }
};
```

1.3 centroids.cpp

```
vector<int> centroids(const vector<vector<int>> &g) {
  int n = g.size();
  vector<int> sz(n, 0), res;
  auto dfs = [&](int now, int par, auto &&dfs) -> void {
    sz[now] = 1;
    bool ch = true;
    for (auto to : g[now])
      if (to != par) {
        dfs(to, now, dfs);
        if (sz[to] > n / 2) ch = false;
        sz[now] += sz[to];
      }
    if (n - sz[now] > n / 2) ch = false;
    if (ch) res.push_back(now);
  };
  dfs(0, n, dfs);
  return res;
}
```

1.4 chinese_rem.cpp

```
// as + bt = GCD(a,b) a,b:const s,t:var(any)
// return GCD(a,b)
long long extGCD(long long a, long long b, long long& s, long long&
↪  t) {
  s = 1, t = 0;
  long long u = 0, v = 1;
  while (b) {
    long long tmp = a / b;
    a -= b * tmp;
    s -= u * tmp;
    t -= v * tmp;
    swap(s, u);
    swap(t, v);
    swap(a, b);
  }
  return a;
}
// (mod)x+ay=1, calculate y -> a^-1 (mod m) (a,m : coprime)
long long calcinv(long long a, long long m) {
  long long s, t;
  extGCD(a, m, s, t);
  return (s + m) % m;
}

// a * x = b_i(mod m_i) calc min x,lcm(m_i).
// if not exist, return (-1,-1)
pair<long long, long long> linear_congruence(const vector<long
↪  long>& a,
                                             const vector<long
                                             ↪  long>& b,
                                             const vector<long
                                             ↪  long>& m) {
  long long x = 0, l = 1;
  assert(b.size() == m.size() && a.size() == b.size());
  long long len = a.size();
  for (int i = 0; i < len; ++i) {
    long long p = a[i] * l, q = b[i] - a[i] * x, d = gcd(m[i], p);
    if (q % d != 0) return make_pair(-1, -1);
    long long t = q / d * calcinv(p / d, m[i] / d) % (m[i] / d);
    x += l * t;
    l *= m[i] / d;
  }
  return make_pair(x % l, l);
}

// x=b_i(mod m_i) calc min x,lcm(m_i).
// if not exist, return (-1,-1)
pair<long long, long long> chinese_rem(const vector<long long>& b,
                                       const vector<long long>& m) {
  long long r = 0, lcm = 1;
  assert(b.size() == m.size());
  long long bsize = b.size();
  for (int i = 0; i < bsize; ++i) {
    long long p, q, d, now;
    d = extGCD(lcm, m[i], p, q);
    if ((b[i] - r) % d != 0) return make_pair(-1, -1);
    now = (b[i] - r) / d * p % (m[i] / d);
    r += lcm * now;
    lcm *= m[i] / d;
  }
  return make_pair((r + lcm) % lcm, lcm);
}

// x=b_i(mod m_i) calc min x(mod nowMOD).
// if not exist, return -1
long long garner(vector<long long>& b, vector<long long>& m,
                 long long nowMOD = 9223372036854775807LL) {
  assert(b.size() == m.size());
  // prepair, O(N^2)
  // if m_i are coprime, don't have to do it
  long long bsize = b.size(), msize = m.size() + 1, dummy1, dummy2;
  for (int i = 0; i < bsize; ++i)
    for (int j = 0; j < i; ++j) {
      long long g = extGCD(m[i], m[j], dummy1, dummy2);
      if ((b[i] - b[j]) % g != 0) return -1;
      m[i] /= g;
      m[j] /= g;
      long long gi = extGCD(m[i], g, dummy1, dummy2), gj;
      gj = g / gi;
      do {
        g = extGCD(gi, gj, dummy1, dummy2);
        gi *= g, gj /= g;
      } while (g != 1);
      m[i] *= gi, m[j] *= gj;
      b[i] %= m[i], b[j] %= m[j];
    }
  // calc
  m.push_back(nowMOD);
  vector<long long> coeffs(msize, 1);
  vector<long long> constants(msize, 0);
  for (int k = 0; k < bsize; ++k) {
    long long tmp = (b[k] - constants[k]) % m[k];
    long long t = (tmp + m[k]) % m[k] * calcinv(coeffs[k], m[k]) %
    ↪  m[k];
    for (int i = k + 1; i < msize; ++i) {
      (constants[i] += t * coeffs[i] % m[i]) %= m[i];
      (coeffs[i] *= m[k]) %= m[i];
    }
  }
  return constants.back();
}
```

1.5 Dice.cpp

```
/*
 b
lurd
 f
*/
struct Dice {
  long long l, r, f, b, d, u, x, y;
  Dice(long long _l = 4, long long _r = 3, long long _f = 2, long
  ↪  long _b = 5,
       long long _d = 6, long long _u = 1, long long _x = 0, long
       ↪  long _y = 0)
      : l(_l), r(_r), f(_f), b(_b), d(_d), u(_u), x(_x), y(_y) {}

  void RollN() {
    long long buff = d;
    d = b;
    b = u;
    u = f;
    f = buff;
    ++y;
  }

  void RollS() {
    long long buff = d;
    d = f;
    f = u;
    u = b;
    b = buff;
    --y;
  }

  void RollE() {
    long long buff = d;
    d = r;
    r = u;
    u = l;
    l = buff;
    ++x;
  }

  void RollW() {
    long long buff = d;
```

```
    d = l;
    l = u;
    u = r;
    r = buff;
    --x;
  }

  void RollL() {
    long long buff = f;
    f = l;
    l = b;
    b = r;
    r = buff;
  }

  void RollR() {
    long long buff = f;
    f = r;
    r = b;
    b = l;
    l = buff;
  }

  vector<Dice> makeDice() {
    vector<Dice> ret;
    for (int i = 0; i < 6; i++) {
      Dice d(*this);
      if (i == 1) d.RollN();
      if (i == 2) d.RollS();
      if (i == 3) d.RollS(), d.RollS();
      if (i == 4) d.RollE();
      if (i == 5) d.RollW();
      for (int j = 0; j < 4; j++) {
        ret.emplace_back(d);
        d.RollL();
      }
    }
    return (ret);
  }

  bool operator==(const Dice &di) const {
    return l == di.l && r == di.r && f == di.f && b == di.b && d ==
    ↪  di.d &&
           u == di.u;
  }
};
```

1.6 Dinic.cpp

```
template <class T>
struct Dinic {
  struct edge {
    int to;
    T cap;
    int rev;
    edge(int t = 0, T c = 0, int r = -1) : to(t), cap(c), rev(r) {}
  };
  int n;
  T finf;
  vector<vector<edge>> g;
  vector<int> id;
  vector<int> d;
  Dinic(int _n = 1, T _finf = (int)(2e9))
      : n(_n), finf(_finf), g(n, vector<edge>()), id(n), d(n) {}

  int add(int start, int goal, T capacity) {
    g[start].emplace_back(goal, capacity, (int)g[goal].size());
    g[goal].emplace_back(start, 0, (int)g[start].size() - 1);
    return (int)g[start].size() - 1;
  }

  T change(int x, int idx, T newf, int s, int t) {
    int y = g[x][idx].to, ridx = g[x][idx].rev;
    newf -= g[x][idx].cap + g[y][ridx].cap;
    if (newf > 0) {
      g[x][idx].cap += newf;
      return solve(s, t);
    } else if (newf < 0) {
      g[x][idx].cap += newf;
      if (g[x][idx].cap >= 0) return T(0);
      swap(newf = 0, g[x][idx].cap);
      g[y][ridx].cap += newf;
      newf += solve(x, y, -newf);
      if (-newf) {
        solve(x, s, -newf);
        solve(t, y, -newf);
      }
      return newf;
    }
    return T(0);
  }

  void bfs(int st) {
    d.assign(n, -1);
    queue<int> qu;
    d[st] = 0;
    qu.push(st);
    while (qu.size()) {
      int now = qu.front();
      qu.pop();
      for (auto e : g[now])
        if (e.cap > 0 && d[e.to] < 0) {
          d[e.to] = d[now] + 1;
          qu.push(e.to);
        }
    }
  }
  T pathdfs(int now, int goal, T nf) {
    if (now == goal) return nf;
    int len = g[now].size();
    for (int& i = id[now]; i < len; ++i) {
      edge* e = &g[now][i];
      if (e->cap > 0 && d[now] < d[e->to]) {
        T res = pathdfs(e->to, goal, min(nf, e->cap));
        if (res > 0) {
          e->cap -= res;
          g[e->to][e->rev].cap += res;
          return res;
        }
      }
    }
    return 0;
  }
  T solve(int start, int goal, T flimit = 0) {
    if (!flimit) flimit = finf;
    T res = 0, nf = 0;
    while (flimit - res) {
      bfs(start);
      if (d[goal] < 0) return res;
      id.assign(n, 0);
      while ((nf = pathdfs(start, goal, flimit - res)) > 0) res +=
      ↪  nf;
    }
    return res;
  }
};
```

1.7 div_moebius.cpp

```
map<long long, long long> div_moebius(long long n) {
  map<long long, long long> res;
  vector<long long> primes;
  for (long long i = 2; i * i <= n; ++i)
    if (n % i == 0) {
      primes.push_back(i);
      while (n % i == 0) n /= i;
    }
  if (n > 1) primes.push_back(n);
  n = primes.size();
  for (long long i = 0; i < (1 << n); ++i) {
    long long mu = 1, d = 1;
    for (long long j = 0; j < n; ++j)
      if (i >> j & 1) {
        mu *= -1;
        d *= primes[j];
      }
    res[d] = mu;
  }
  return res;
}
```

1.8 EulerTour.cpp

```
struct EulerTour {
  int n, root;
  vector<vector<int>> g;
  vector<int> par, dep, in, out, lst;
  EulerTour(int n = 1, int _root = 0)
      : root(_root), g(n), par(n), dep(n), in(n), out(n) {}
  EulerTour(const vector<vector<int>> &_g, const int _root = 0)
      : root(_root),
        g(_g),
        par(_g.size()),
        dep(_g.size()),
        in(_g.size()),
        out(_g.size()) {
    build();
  }
  void add(int a, int b) {
    g[a].push_back(b);
    g[b].push_back(a);
  }
  void build() {
    lst.clear();
    dfs(root, -1, 0);
  }
  void dfs(int now, int bf, int d) {
    dep[now] = d;
```

```
    par[now] = bf;
    in[now] = lst.size();
    lst.push_back(now);
    for (auto &to : g[now])
      if (to != bf) {
        dfs(to, now, d + 1);
        lst.push_back(now);  // edge ver.
      }
    if (lst.back() != now) lst.push_back(now);  // edge ver.
    out[now] = lst.size();
  }
  int chil(int x, int y) { return dep[x] < dep[y] ? y : x; }
  template <typename T, typename F>
  void update(int node, T x, const F &f) {
    f(in[node], x);
    f(out[node], -x);
    // laze pattern
    // f(in[node],out[node],x);
  }
  // u-v(lca:r) edge path: [in[r] + 1,in[v] + 1) + [in[r] + 1, in[u]
  ↪  + 1)
};
```

1.9 euler_phi.cpp

```
long long euler_phi(long long x) {
  long long res = x;
  for (long long i = 2; i * i <= x; ++i)
    if (x % i == 0) {
      res -= res / i;
      while (x % i == 0) x /= i;
    }
  if (x > 1) res -= res / x;
  return res;
}
```

1.10 Factor.cpp

```
struct Factor {
  inline long long mul(long long x, long long y, long long z) {
    return (__int128_t)x * y % z;
  }
  inline long long f(long long x, long long n) {
    return ((__int128_t)x * x + 1) % n;
  }
  long long mpow(long long b, long long p, long long mod) {
    long long res = 1;
    while (p) {
      if (p & 1) res = mul(res, b, mod);
      b = mul(b, b, mod);
      p >>= 1;
    }
    return res;
  }
  bool millar(long long n) {
    if ((~n & 1) && n != 2) return 0;
    long long d = n - 1, x = n - 1;
    int s = __builtin_ctzll(d);
    d >>= s;
    vector<long long> a = {2, 7, 61};
    if (n >= 4759123141LL)
      a = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
    for (auto p : a) {
      if (p >= n) break;
      long long now = mpow(p, d, n);
      if (now == 1) continue;
      int i = s;
      while (now != x && i--) now = mul(now, now, n);
      if (now != x) return 0;
    }
    return 1;
  }
  long long rho(long long n) {
    long long x = 0, y = 0;
    int i = 1;
    while (1) {
      long long g = gcd(abs(x - y), n);
      if (g == n)
        y = f(x = ++i, n);
      else if (g == 1) {
        x = f(x, n);
        y = f(f(y, n), n);
      } else
        return g;
    }
    return -1;
  }
  map<long long, int> factor(long long n) {
    map<long long, int> res;
    if (n % 2 == 0) {
      res[2] = __builtin_ctzll(n);
      n >>= __builtin_ctzll(n);
    }
    queue<long long> qu;
    qu.push(n);
    while (qu.size()) {
      long long now = qu.front();
      qu.pop();
      if (now == 1) continue;
      if (millar(now)) {
        ++res[now];
        continue;
      }
      long long p = rho(now);
      qu.push(p);
      qu.push(now / p);
    }
    return res;
  }
};
```

1.11 fast_zeta_transform.cpp

```
// res[i] = sum j \in i v[j](if upper, swap(i, j))
template <class T>
vector<T> fast_zeta_transform(int n, vector<T> v, bool upper = 0) {
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < (1 << n); ++j)
      if ((j >> i & 1) ^ upper) v[j] += v[j ^ (1 << i)];
  return v;
}
```

1.12 FFT.cpp

```
// max(res[i]) <= 2^53 -> no error
struct FFT {
  struct CP {
    double x, y;

    CP() : x(0), y(0) {}

    CP(double x, double y) : x(x), y(y) {}

    inline CP operator+(const CP &c) const { return CP(x + c.x, y +
    ↪  c.y); }

    inline CP operator-(const CP &c) const { return CP(x - c.x, y -
    ↪  c.y); }

    inline CP operator*(const CP &c) const {
      return CP(x * c.x - y * c.y, x * c.y + y * c.x);
    }
    CP &operator*=(const CP &p) {
      *this = CP(*this) * p;
      return *this;
    }
    CP &operator/=(const double &p) {
      x /= p, y /= p;
      return *this;
    }

    inline CP conj() const { return CP(x, -y); }
  };
  const double PI = acos(-1);
  int base = 1;
  vector<CP> roots{CP(0, 0), CP(1, 0)};
  vector<int> rv{0, 1};
  FFT() {}
  void ensure_base(int nb) {
    if (nb <= base) return;
    rv.resize(1 << nb);
    roots.resize(1 << nb);
    for (int i = 0; i < (1 << nb); ++i)
      rv[i] = (rv[i >> 1] >> 1) + ((i & 1) << (nb - 1));
    while (base < nb) {
      double arg = PI * 2.0 / (1 << (base + 1));
      for (int i = 1 << (base - 1); i < (1 << base); ++i) {
        roots[i << 1] = roots[i];
        double narg = arg * (2 * i + 1 - (1 << base));
        roots[(i << 1) + 1] = CP(cos(narg), sin(narg));
      }
      ++base;
    }
  }
  void fft(vector<CP> &a, int n, bool sg = 0) {
    assert((n & (n - 1)) == 0);
    int dif = base - __builtin_ctz(n);
    for (int i = 0; i < n; ++i)
      if (i < (rv[i] >> dif)) swap(a[i], a[rv[i] >> dif]);
    for (int k = 1; k < n; k <<= 1)
      for (int i = 0; i < n; i += 2 * k)
        for (int j = 0; j < k; ++j) {
          CP z = a[i + j + k] * (sg ? roots[j + k] : roots[j +
          ↪  k].conj());
          a[i + j + k] = a[i + j] - z;
          a[i + j] = a[i + j] + z;
        }
```

```
    if (sg)
      for (int i = 0; i < n; ++i) a[i] /= n;
  }
  template <class T>
  vector<long long> multiply(const vector<T> &a, const vector<T> &b)
  ↪ {
    int need = a.size() + b.size() - 1;
    int nb = 1;
    while ((1 << nb) < need) ++nb;
    ensure_base(nb);
    int sz = 1 << nb;
    vector<CP> fa(sz), fb(sz);
    for (int i = 0; i < sz; ++i) {
      if (i < a.size()) fa[i] = CP(a[i], 0);
      if (i < b.size()) fb[i] = CP(b[i], 0);
    }
    fft(fa, sz);
    fft(fb, sz);
    for (int i = 0; i < sz; ++i) fa[i] *= fb[i];
    fft(fa, sz, 1);
    vector<long long> res(need);
    for (int i = 0; i < need; ++i) res[i] = llround(fa[i].x);
    return res;
  }
};
```

1.13 Fib.cpp

```
// 0-indexed
template <class T>
struct Fib {
  int n, l, r, len;
  vector<long long> fib, lens;
  Fib(int _n = 1000000) : n(_n), fib({1, 1}), lens({1, 2}) {
    while (lens.back() < n) {
      int len = fib.size();
      fib.push_back(fib[len - 2] + fib[len - 1]);
      lens.push_back(lens.back() + fib.back());
    }
  }
  void reset() { l = -1, r = lens.back(), len = fib.size(); }

  template <typename F, typename G>
  T calc(F& query, G& check, T unit) {
    reset();
    vector<int> called(r, 0);
    vector<T> memo(r, unit);
    for (int i = len - 1; i >= 0; --i) {
      int ml = l + fib[i], mr = r - fib[i];
      if (!called[ml] && ml < n) memo[ml] = query(ml), called[ml] =
      ↪ 1;
      if (!called[mr] && mr < n) memo[mr] = query(mr), called[mr] =
      ↪ 1;
      if (check(memo[ml], memo[mr]))
        l = ml;
      else
        r = mr;
    }
    return memo[r];
  }
};
```

1.14 floor_sum.cpp

```
// sum_{i = 0}^{n-1}floor((a * i + b)/m)
long long floor_sum(long long n, long long m, long long a, long long
↪ b) {
  long long res = b / m * n + n * (n - 1) / 2 * (a / m);
  b %= m;
  a %= m;
  if (a == 0 || n == 0) return res;
  long long p = (a * (n - 1) + b) / m;
  return res + floor_sum(p, a, m, a * (n - 1) - m * p + b + a);
}
```

1.15 FWHT_AND.cpp

```
template <class T>
struct FWHT_AND {
  vector<T> multiply(vector<T> a, vector<T> b) {
    int len = 1, n = max(a.size(), b.size());
    while (len < n) len <<= 1;
    a.resize(len), b.resize(len);
    fwht(a), fwht(b);
    for (int i = 0; i < len; ++i) a[i] *= b[i];
    ifwht(a);
    return a;
  }
  void fwht(vector<T> &v) {
    int n = v.size();
    for (int i = 1; i < n; i <<= 1)
      for (int j = 0; j < n; ++j)
        if (!(j & i)) v[j] += v[j | i];
  }
  void ifwht(vector<T> &v) {
    int n = v.size();
    for (int i = 1; i < n; i <<= 1)
      for (int j = 0; j < n; ++j)
        if (!(j & i)) v[j] -= v[j | i];
  }
};
```

1.16 FWHT_OR.cpp

```
template <class T>
struct FWHT_OR {
  vector<T> multiply(vector<T> a, vector<T> b) {
    int len = 1, n = max(a.size(), b.size());
    while (len < n) len <<= 1;
    a.resize(len), b.resize(len);
    fwht(a), fwht(b);
    for (int i = 0; i < len; ++i) a[i] *= b[i];
    ifwht(a);
    return a;
  }
  void fwht(vector<T> &v) {
    int n = v.size();
    for (int i = 1; i < n; i <<= 1)
      for (int j = 0; j < n; ++j)
        if (!(j & i)) v[j | i] += v[j];
  }
  void ifwht(vector<T> &v) {
    int n = v.size();
    for (int i = 1; i < n; i <<= 1)
      for (int j = 0; j < n; ++j)
        if (!(j & i)) v[j | i] -= v[j];
  }
};
```

1.17 FWHT_XOR.cpp

```
template <class T>
struct FWHT_XOR {
  vector<T> multiply(vector<T> a, vector<T> b) {
    int len = 1, n = max(a.size(), b.size());
    while (len < n) len <<= 1;
    a.resize(len), b.resize(len);
    fwht(a), fwht(b);
    for (int i = 0; i < len; ++i) a[i] *= b[i];
    ifwht(a);
    return a;
  }
  void fwht(vector<T> &v) {
    int n = v.size();
    for (int i = 1; i < n; i <<= 1)
      for (int j = 0; j < n; ++j)
        if (!(j & i)) {
          T x = v[j], y = v[j | i];
          v[j] = x + y, v[j | i] = x - y;
        }
  }
  void ifwht(vector<T> &v) {
    int n = v.size();
    for (int i = 1; i < n; i <<= 1)
      for (int j = 0; j < n; ++j)
        if (!(j & i)) {
          T x = v[j], y = v[j | i];
          // mint two = mint(2).inverse();
          v[j] = (x + y) / 2, v[j | i] = (x - y) / 2;
        }
  }
};
```

1.18 gauss_jordan.cpp

```
// use Matrix, ModInt
// MOD ver.
#define MOD (long long)(1e9 + 7)
int gauss_jordan(Matrix<ModInt<MOD>> &A, bool is_extended = false) {
  int m = A.height(), n = A.width(), rank = 0;
  for (int col = 0; col < n; ++col) {
    if (is_extended && col == n - 1) break;
    int piv = -1;
    for (int row = rank; row < m; ++row)
      if (A[row][col] != 0) {
        piv = row;
        break;
      }
    if (piv == -1) continue;
    swap(A[piv], A[rank]);
    ModInt<MOD> inv = A[rank][col].inverse();
    for (int col2 = 0; col2 < n; ++col2) A[rank][col2] *= inv;
    for (int row = 0; row < m; ++row)
      if (row != rank && A[row][col] != 0) {
        ModInt<MOD> fac = A[row][col];
        for (int col2 = 0; col2 < n; ++col2)
```

```
          A[row][col2] -= A[rank][col2] * fac;
      }
    ++rank;
  }
  return rank;
}

int linear_equation(Matrix<ModInt<MOD>> A, vector<ModInt<MOD>> b,
                    vector<ModInt<MOD>> &ans) {
  int m = A.height(), n = A.width();
  Matrix<ModInt<MOD>> M(m, n + 1);
  assert((int)b.size() == m);
  for (int i = 0; i < m; ++i) {
    for (int j = 0; j < n; ++j) M[i][j] = A[i][j];
    M[i][n] = b[i];
  }
  int rank = gauss_jordan(M, 1);
  ans.assign(n, 0);
  for (int i = 0; i < rank; ++i) {
    int id = -1;
    for (int j = 0; j < n; ++j)
      if (M[i][j] != 0) {
        id = j;
        break;
      }
    ans[id] = M[i][n];
  }
  // exist?
  for (int row = rank; row < m; ++row)
    if (M[row][n] != 0) return -1;
  return rank;
}
```

1.19 gcdlcm.cpp

```
// calculate |gcd|.
// if  ether num is 0, return 0
long long GCD(long long left, long long right) {
  if (left == 0 || right == 0) return 0;
  if (left < 0) left *= -1;
  if (right < 0) right *= -1;
  if (left < right) swap(left, right);
  long long nextnum, ansgcd = -1;
  while (ansgcd == -1) {
    nextnum = left % right;
    if (nextnum == 0) ansgcd = right;
    left = right;
    right = nextnum;
  }
  return ansgcd;
}

long long LCM(long long left, long long right) {
  return left / GCD(left, right) * right;
}
```

1.20 Geometry.cpp

```
// arg(x) : argment,[-PI,PI]
using CP = complex<long double>;
#define X real()
#define Y imag()
const long double PI = acos(-1.0L);
const long double EPS = 1e-10;
bool operator==(const CP &l, const CP &r) { return norm(l - r) <=
↪  EPS; }
struct Circle {
  CP o;
  long double r;
  Circle(long double _x = 0.0L, long double _y = 0.0L, long double
  ↪  _r = 0.0L)
      : o(CP(_x, _y)), r(_r) {}
  Circle(CP _o, long double _r = 0.0) : o(_o), r(_r) {}
  bool operator<(const Circle &cr) const { return r < cr.r; }
};

struct Line {
  CP s, t;
  Line(long double sx = 0.0L, long double sy = 0.0L, long double tx
  ↪  = 0.0L,
       long double ty = 0.0L)
      : s(CP(sx, sy)), t(CP(tx, ty)) {}
  Line(CP _s, CP _t) : s(_s), t(_t) {}
};

// cos a
long double costh(const long double &a, const long double &b,
                  const long double &c) {
  return (b * b - a * a + c * c) / (2.0L * b * c);
}

// dot(a,b) = |a||b|cos x
long double dot(const CP &a, const CP &b) { return (conj(a) * b).X; }

// cross(a,b) : area of parallelogram
// sign : a-> b ,counter clockwise? + : -
long double cross(const CP &a, const CP &b) { return (conj(a) *
↪  b).Y; }
long double corner(const CP &a, const CP &b) {
  //[0,PI]
  return acos(dot(a, b) / (abs(a) * abs(b)));
}

CP projectionLP(const CP &s, const CP &t, const CP &p) {
  if (s == t) return s;
  CP base = t - s;
  long double r = dot(p - s, base) / norm(base);
  return s + base * r;
}
CP projectionLP(const Line &l, const CP &p) {
  return projectionLP(l.s, l.t, p);
}

CP reflectionLP(const CP &s, const CP &t, const CP &p) {
  CP tmp = (projectionLP(s, t, p) - p);
  tmp *= 2;
  return p + tmp;
}
CP reflectionLP(const Line &l, const CP &p) {
  return reflectionLP(l.s, l.t, p);
}

int calc_clockwiseSP(const CP &s, CP t, CP p) {
  t -= s;
  p -= s;
  if (cross(t, p) > EPS) return 1;     // "COUNTER_CLOCKWISE"
  if (cross(t, p) < -EPS) return -1;  //"CLOCK_WISE"
  if (dot(t, p) < 0) return 2;         // "ONLINE_BACK"
  if (norm(t) < norm(p)) return -2;    // "ONLINE_FRONT"
  return 0;                            // "ON_SEGMENT"
}
int calc_clockwiseSP(const Line &l, const CP &p) {
  return calc_clockwiseSP(l.s, l.t, p);
}

int parallel_orthogonalLL(const CP &s, CP t, const CP &a, CP b) {
  t -= s;
  b -= a;
  if (abs(cross(t, b)) <= EPS) return 2;  // "parallel"
  if (abs(dot(t, b)) <= EPS) return 1;    // "orthogonal"
  return 0;
}
int parallel_orthogonalLL(const Line &ll, const Line &lr) {
  return parallel_orthogonalLL(ll.s, ll.t, lr.s, lr.t);
}

CP intersectionLL(const CP &a, const CP &b, const CP &c, const CP
↪  &d) {
  return a + (b - a) * (cross(d - c, c - a) / cross(d - c, b - a));
}
CP intersectionLL(const Line &ll, const Line &lr) {
  return intersectionLL(ll.s, ll.t, lr.s, lr.t);
}

bool on_segSP(const CP &s, const CP &t, const CP &p) {
  // if not use end point, dot(s - p, t - p) < 0
  return abs(cross(s - p, t - p)) <= EPS && dot(s - p, t - p) <= 0;
}
bool on_segSP(const Line &l, const CP &p) { return on_segSP(l.s,
↪  l.t, p); }

// crossing segments? (a,b) and (c,d)
bool iscrossSS(const CP &a, const CP &b, const CP &c, const CP &d) {
  // parallel
  return calc_clockwiseSP(a, b, c) * calc_clockwiseSP(a, b, d) <= 0
  ↪  &&
         calc_clockwiseSP(c, d, a) * calc_clockwiseSP(c, d, b) <= 0;
  // if (abs(cross(a - b, c - d)) <= EPS) {
  //   return on_segSP(a, b, c) || on_segSP(a, b, d) || on_segSP(c,
  ↪  d, a) ||
  //          on_segSP(c, d, b);
  // }
  // CP isp = intersectionLL(a, b, c, d);
  // return on_segSP(a, b, isp) && on_segSP(c, d, isp);
}

bool iscrossSS(const Line &ll, const Line &lr) {
  return iscrossSS(ll.s, ll.t, lr.s, lr.t);
}

long double distLP(const CP &s, const CP &t, const CP &p) {
  return abs(cross(t - s, p - s) / abs(t - s));
}
long double distLP(const Line &l, const CP &p) { return distLP(l.s,
↪  l.t, p); }

long double distSP(const CP &s, const CP &t, const CP &p) {
  if (dot(t - s, p - s) < 0) return abs(p - s);
```

```
  if (dot(s - t, p - t) < 0) return abs(p - t);
  return distLP(s, t, p);
}
long double distSP(const Line &l, const CP &p) { return distSP(l.s,
↪  l.t, p); }

long double distSS(const CP &a, const CP &b, const CP &c, const CP
↪  &d) {
  long double res = 1e18;
  if (iscrossSS(a, b, c, d)) return 0.0L;
  res = min(res, distSP(a, b, c));
  res = min(res, distSP(a, b, d));
  res = min(res, distSP(c, d, a));
  res = min(res, distSP(c, d, b));
  return res;
}
long double distSS(const Line &ll, const Line &lr) {
  return distSS(ll.s, ll.t, lr.s, lr.t);
}

// counter clockwise
bool is_convex(const vector<CP> &pol) {
  int n = pol.size();
  for (int i = 0; i < n; ++i)
    if (cross(pol[(i + 1) % n] - pol[i], pol[(i + 2) % n] - pol[(i +
    ↪  1) % n]) <
        -EPS)
      return 0;
  return 1;
}

vector<CP> convex_hull(vector<CP> &ps) {
  auto lmd = [&](const CP &l, const CP &r) {
    if (l.X != r.X) return l.X < r.X;
    return l.Y < r.Y;
  };
  vector<CP> res;
  int psize = ps.size();
  sort(ps.begin(), ps.end(), lmd);
  int k = 0;
  res.resize(psize * 2);
  for (int i = 0; i < psize; ++i) {
    while (k > 1 && cross(res[k - 1] - res[k - 2], ps[i] - res[k -
    ↪  1]) <= 0)
      --k;
    res[k++] = ps[i];
  }
  for (int i = psize - 2, t = k; i >= 0; --i) {
    while (k > t && cross(res[k - 1] - res[k - 2], ps[i] - res[k -
    ↪  1]) <= 0)
      --k;
    res[k++] = ps[i];
  }
  res.resize(k - 1);
  return res;
}

long double convex_diameter(const vector<CP> &pol) {
  vector<CP> ps = pol;
  ps = convex_hull(ps);
  int n = ps.size(), i = 0, j = 0;
  if (n < 2) return 0.0L;
  if (n == 2) return abs(ps[0] - ps[1]);
  for (int k = 0; k < n; ++k) {
    if (ps[k].X < ps[i].X) i = k;
    if (ps[k].X > ps[j].X) j = k;
  }
  long double res = 0;
  int si = i, sj = j;
  while (i != sj || j != si) {
    res = max(res, abs(ps[i] - ps[j]));
    if (cross(ps[(i + 1) % n] - ps[i], ps[(j + 1) % n] - ps[j]) < 0)
      (++i) %= n;
    else
      (++j) %= n;
  }
  return res;
}

vector<CP> convex_cut(const vector<CP> &pol, const CP &s, const CP
↪  &t) {
  vector<CP> res;
  int n = pol.size();
  for (int i = 0; i < n; ++i) {
    CP nows = pol[i], nowt = pol[(i + 1) % n];
    if (cross(t - s, nows - s) >= -EPS) res.push_back(nows);
    if (cross(t - s, nows - s) * cross(t - s, nowt - s) < 0)
      res.push_back(intersectionLL(s, t, nows, nowt));
  }
  return res;
}
vector<CP> convex_cut(const vector<CP> &pol, const Line &l) {
  return convex_cut(pol, l.s, l.t);
}
```

```
// number of tangents
int iscrossCC(Circle l, Circle r) {
  if (l.r < r.r) swap(l, r);
  long double distlr = abs(l.o - r.o);
  if (l.r + r.r < distlr)
    return 4;  // not touch
  else if (abs(distlr - l.r - r.r) <= EPS)
    return 3;  // circumscription
  else if (l.r - r.r < distlr)
    return 2;  // cross
  else if (abs(distlr - l.r + r.r) <= EPS)
    return 1;  // inscribed
  else         // contain
    return 0;
}

vector<CP> intersectionCC(const Circle &c1, const Circle &c2) {
  vector<CP> res;
  if (iscrossCC(c1, c2) == 4) return res;

  long double d = abs(c1.o - c2.o);
  long double a = acos(costh(c2.r, c1.r, d));
  long double t = atan2(c2.o.imag() - c1.o.imag(), c2.o.real() -
  ↪  c1.o.real());
  res.push_back(c1.o + CP(cos(t + a) * c1.r, sin(t + a) * c1.r));
  res.push_back(c1.o + CP(cos(t - a) * c1.r, sin(t - a) * c1.r));
  if (res[0].X > res[1].X || (res[0].X == res[1].X && res[0].Y >
  ↪  res[1].Y))
    swap(res[0], res[1]);
  return res;
}

vector<CP> intersectionCL(const Circle &ci, const CP &s, CP t) {
  vector<CP> res(2, projectionLP(s, t, ci.o));
  long double r = sqrt(ci.r * ci.r - norm(res[0] - ci.o));
  if (r <= EPS || t == s) return res;
  t -= s;
  t *= r / abs(t);
  res[0] += t;
  res[1] -= t;
  if (res[0].X > res[1].X || (res[0].X == res[1].X && res[0].Y >
  ↪  res[1].Y))
    swap(res[0], res[1]);
  return res;
}
vector<CP> intersectionCL(const Circle &ci, const Line &l) {
  return intersectionCL(ci, l.s, l.t);
}

vector<CP> contactCP(const Circle &ci, const CP &p) {
  vector<CP> res;
  long double d = abs(ci.o - p);
  if (abs(d - ci.r) <= EPS) {
    res.push_back(p);
    return res;
  } else if (d < ci.r)
    return res;
  long double arg = asin(ci.r / d);
  res.push_back((ci.o - p) * CP(cos(arg), sin(arg)));
  res[0] *= (d * cos(arg)) / abs(res[0]);
  res[0] += p;
  res.push_back(reflectionLP(p, ci.o, res[0]));
  if (res[0].X > res[1].X || (res[0].X == res[1].X && res[0].Y >
  ↪  res[1].Y))
    swap(res[0], res[1]);
  return res;
}

vector<Line> tangentCC(Circle cl, Circle cr) {
  vector<Line> res;
  if (cl.r < cr.r) swap(cl, cr);
  long double g = abs(cl.o - cr.o);
  if (abs(g - 0.0L) <= EPS) return res;
  CP hor = (cr.o - cl.o) / g, ver;
  ver = hor * (CP(cos(PI * 0.5L), sin(PI * 0.5L)));
  for (int s : {-1, 1}) {
    long double h = (cl.r + s * cr.r) / g;
    if (abs(1 - h * h) <= EPS) {
      res.emplace_back(cl.o + hor * cl.r, cl.o + (hor + ver) * cl.r);
    } else if (1 - h * h > 0) {
      CP nhor = hor * h, nver = ver * sqrtl(1 - h * h);
      res.emplace_back(cl.o + (nhor + nver) * cl.r,
                       cr.o - (nhor + nver) * (cr.r * s));
      res.emplace_back(cl.o + (nhor - nver) * cl.r,
                       cr.o - (nhor - nver) * (cr.r * s));
    }
  }
  return res;
}

long double areaPol(const vector<CP> &pol) {
  int n = pol.size();
  long double res = 0;
```

```
  for (int i = 0; i < n; ++i)
    res += (pol[(i - 1 + n) % n].X - pol[(i + 1) % n].X) * pol[i].Y;
  return res / 2.0L;
}

int containPolP(const vector<CP> &pol, CP p) {
  bool con = 0, onseg = 0;
  int n = pol.size();
  for (int i = 0; i < n; ++i) {
    onseg |= on_segSP(pol[i], pol[(i + 1) % n], p);
    CP s = pol[i] - p, t = pol[(i + 1) % n] - p;
    if (s.Y > t.Y) swap(s, t);
    if (s.Y * t.Y <= 0 && t.Y > 0 && cross(s, t) < 0) con = !con;
  }
  if (onseg) return 1;
  if (con) return 2;
  return 0;
}

long double closest_pair(vector<CP> &ps, int l = -1, int r = -1,
                         bool reqsqrt = 0) {
  if (l == r && l == -1) {
    l = 0;
    r = ps.size();
    reqsqrt = 1;
    auto lmd = [&](const CP &l, const CP &r) {
      if (l.X != r.X) return l.X < r.X;
      return l.Y < r.Y;
    };
    sort(ps.begin(), ps.end(), lmd);
  }
  if (r - l < 2) return 1e18;
  if (r - l == 2) {
    if (ps[l].Y > ps[l + 1].Y) swap(ps[l], ps[l + 1]);
    if (reqsqrt) return abs(ps[l] - ps[l + 1]);
    return norm(ps[l] - ps[l + 1]);
  }
  int mid = (l + r) / 2;
  long double x = ps[mid].X,
              res = min(closest_pair(ps, l, mid), closest_pair(ps,
              ↪  mid, r));
  auto f = [](CP pl, CP pr) { return pl.Y < pr.Y; };
  inplace_merge(ps.begin() + l, ps.begin() + mid, ps.begin() + r, f);
  vector<CP> tmp;
  for (int i = l; i < r; ++i) {
    long double dx = abs(ps[i].X - x);
    int tsize = tmp.size();
    if (dx * dx >= res) continue;
    for (int j = 0; j < tsize; ++j) {
      CP delta = ps[i] - tmp[tsize - 1 - j];
      if (delta.Y * delta.Y >= res) break;
      res = min(res, norm(delta));
    }
    tmp.push_back(ps[i]);
  }
  if (reqsqrt) res = sqrtl(res);
  return res;
}

Circle min_ball(vector<CP> ps) {
  int n = ps.size();
  if (n == 1) return Circle(ps[0], 0.0L);
  mt19937 mt(int(time(0)));
  shuffle(ps.begin(), ps.end(), mt);
  auto make3 = [](const CP &a, const CP &b, const CP &c) {
    long double A = norm(b - c), B = norm(c - a), C = norm(a - b),
                S = cross(b - a, c - a);
    CP p = (A * (B + C - A) * a + B * (C + A - B) * b + C * (A + B -
    ↪  C) * c) /
           (4 * S * S);
    long double nowr = norm(p - a);
    return Circle(p, nowr);
  };
  auto make2 = [](const CP &a, const CP &b) {
    CP c = (a + b) / 2.0L;
    long double nowr = norm(a - c);
    return Circle(c, nowr);
  };
  auto in_circle = [](const CP &a, const Circle &c) {
    return norm(a - c.o) <= c.r + EPS;
  };
  Circle res = make2(ps[0], ps[1]);
  for (int i = 2; i < n; ++i)
    if (!in_circle(ps[i], res)) {
      res = make2(ps[0], ps[i]);
      for (int j = 1; j < i; ++j)
        if (!in_circle(ps[j], res)) {
          res = make2(ps[i], ps[j]);
          for (int k = 0; k < j; ++k)
            if (!in_circle(ps[k], res)) res = make3(ps[i], ps[j],
            ↪  ps[k]);
        }
    }
  res.r = sqrtl(res.r);
  return res;
}

bool arg_comp(CP a, CP b) {
  int up_down_a = a.Y > 0 || (abs(a.Y) <= EPS && a.X >= 0);
  int up_down_b = b.Y > 0 || (abs(b.Y) <= EPS && b.X >= 0);
  if (up_down_a != up_down_b) return up_down_a < up_down_b;
  if (a.X * b.Y == a.Y * b.X) return norm(a) < norm(b);
  return calc_clockwiseSP(CP(0, 0), a, b) == 1;
}
bool operator<(const Line &l, const Line &r) {
  CP lp = l.t - l.s, rp = r.t - r.s;
  return arg_comp(lp, rp);
}
```

1.21 HeavyLightDecomposition.cpp

```
struct HeavyLightDecomposition {
  int root;
  vector<vector<int>> g;
  vector<int> sz, par, dep, in, out, head;
  HeavyLightDecomposition(int n = 1, int _root = 0)
      : root(_root), g(n), sz(n), par(n), dep(n), in(n), out(n),
      ↪  head(n) {}
  HeavyLightDecomposition(const vector<vector<int>> &_g, const int
  ↪  _root = 0)
      : root(_root),
        g(_g),
        sz(_g.size()),
        par(_g.size()),
        dep(_g.size()),
        in(_g.size()),
        out(_g.size()),
        head(_g.size()) {
    build();
  }
  void add(int a, int b) {
    g[a].push_back(b);
    g[b].push_back(a);
  }
  void build() {
    dfs_sz(root, -1, 0);
    int t = 0;
    dfs_hld(root, -1, t);
  }
  void dfs_sz(int now, int bf, int d) {
    dep[now] = d;
    par[now] = bf;
    sz[now] = 1;
    if (g[now].size() && g[now][0] == bf) swap(g[now][0],
    ↪  g[now].back());
    for (auto &to : g[now]) {
      if (to == bf) continue;
      dfs_sz(to, now, d + 1);
      sz[now] += sz[to];
      if (sz[g[now][0]] < sz[to]) swap(g[now][0], to);
    }
  }
  void dfs_hld(int now, int bf, int &t) {
    in[now] = t++;
    for (auto &to : g[now]) {
      if (to == bf) continue;
      head[to] = (g[now][0] == to ? head[now] : to);
      dfs_hld(to, now, t);
    }
    out[now] = t;
  }
  int lca(int x, int y) {
    for (;; y = par[head[y]]) {
      if (in[x] > in[y]) swap(x, y);
      if (head[x] == head[y]) return x;
    }
  }
  int chil(int x, int y) { return dep[x] < dep[y] ? y : x; }
  //[l,r)
  template <typename F>
  void for_each(int x, int y, const F &f) {
    for (;; y = par[head[y]]) {
      if (in[x] > in[y]) swap(x, y);
      f(max(in[head[y]], in[x]), in[y] + 1);
      if (head[x] == head[y]) return;
    }
  }
  //[l,r)
  template <typename F>
  void for_eachedge(int x, int y, const F &f) {
    while (1) {
      if (in[x] > in[y]) swap(x, y);
      if (head[x] != head[y]) {
        f(in[head[y]], in[y] + 1);
        y = par[head[y]];
      } else {
        if (x != y) f(in[x] + 1, in[y] + 1);
        break;
```

```
      }
    }
  }
  template <typename T, typename F>
  void update(int node, T x, const F &f) {
    f(in[node], x);
    f(out[node], -x);
    // laze pattern
    // f(in[node], out[node], x);
  }
};
```

1.22 KMP.cpp

```
// A[i]: S[0,i-1] longest match(pre and suf)
template <class T>
struct KMP {
  vector<int> A;
  int n;
  T s;
  KMP() {}
  KMP(T _s) {
    s = _s;
    n = s.size();
    A.assign(n + 1, -1);
    for (int i = 0, j = -1; i < n; ++i) {
      while (j >= 0 && s[i] != s[j]) j = A[j];
      ++j;
      /* KMP
      if(i + 1 < n && s[i + 1] == s[j])
        A[i + 1] = A[j];
      else
        //*/
      A[i + 1] = j;
    }
  }
  inline const int &operator[](int k) const { return (A[k]); }
  vector<int> calccycle() {
    vector<int> res(n, 0);
    for (int i = 0; i < n; ++i) res[i] = i + 1 - A[i + 1];
    return res;
  }
  // search s in t
  vector<int> search(const T &t) {
    vector<int> res;
    int tsize = t.size();
    for (int i = 0, j = 0; i + j < tsize;) {
      if (s[j] == t[i + j]) {
        if (++j != n) continue;
        res.push_back(i);
      }
      i += j - A[j];
      j = max(A[j], 0);
    }
    return res;
  }
};
```

1.23 knapsack_with_limitations.cpp

```
// w:weight,v:value,m:limit, maximize v
template <class T>
vector<T> knapsack_with_limitations(const int lim, const vector<T>
↪  &w,
                                    const vector<T> &m, const
                                    ↪  vector<T> &v,
                                    const T inf = -1) {
  int n = w.size();
  assert(n == m.size() && n == v.size());
  vector<T> dp(lim + 1, inf), deqv(lim + 1);
  vector<int> deq(lim + 1);
  dp[0] = 0;
  for (int i = 0; i < n; ++i)
    for (int a = 0; a < w[i]; ++a) {
      int s = 0, t = 0;
      for (int j = 0; w[i] * j + a <= lim; ++j) {
        if (dp[w[i] * j + a] != inf) {
          auto val = dp[w[i] * j + a] - j * v[i];
          while (s < t && val > deqv[t - 1]) --t;
          deq[t] = j;
          deqv[t++] = val;
        }
        if (s < t) {
          dp[j * w[i] + a] = deqv[s] + j * v[i];
          if (deq[s] == j - m[i]) ++s;
        }
      }
    }
  return dp;
}
```

1.24 lagrange_interpolation.cpp

```
// O(n^2)  P(x_i) = y_i(i:[0,n])
//  calc c_i : P(x) = c_n x^n + c_(n-1) x^(n-1)...c_0
template <typename T>
vector<T> lagrange_interpolation(vector<T> &y, vector<T> &x) {
  assert(y.size() == x.size());
  long long n = y.size();
  vector<T> res(n, 0), Q(n), c[2];
  for (int i = 0; i < 2; ++i) c[i] = vector<T>(n, 0);
  c[0][0] = 1;
  for (int i = 0; i < n; ++i) {
    T inv = 1;
    for (int j = 0; j < n; ++j)
      if (j != i) inv *= x[i] - x[j];
    Q[i] = y[i] / inv;
    for (int j = 0; j < n; ++j) {
      c[(i + 1) % 2][j] = c[i % 2][j] * -x[i];
      if (j != 0) c[(i + 1) % 2][j] += c[i % 2][j - 1];
    }
  }
  for (int i = 0; i < n; ++i) {
    for (int j = n - 1; j >= 0; --j) {
      if (j == n - 1)
        c[(n + 1) % 2][j] = 1;
      else
        c[(n + 1) % 2][j] = c[n % 2][j + 1] + c[(n + 1) % 2][j + 1]
        ↪  * x[i];
      res[j] += c[(n + 1) % 2][j] * Q[i];
    }
  }
  return res;
}

// O(n log mod) calc f(t) x_i = a + i * d, f(y_i) = y_i
template <typename T>
T lagrange_interpolation(const vector<T> &y, const T &t, const T &a
↪  = 0,
                         const T &d = 1) {
  long long n = y.size();
  T res = 0, p = 1;
  for (int i = 1; i < n; ++i) {
    p *= t - (a + d * i);
    p /= -d * i;
  }
  for (int i = 0; i < n; ++i) {
    if (t == a + d * i) return y[i];
    res += y[i] * p;
    p *= t - (a + d * i);
    p /= t - d * (i + 1);
    p *= d * (i - (n - 1));
    p /= d * (i + 1);
  }
  return res;
}
```

1.25 largest_rectangle.cpp

```
template <class T>
T largest_rectangle(const vector<T>& v) {
  T res = 0;
  int n = v.size();
  vector<T> h, id;
  h.push_back(0);
  for (int i = 0; i <= n; ++i) {
    int nxt = i;
    T now = i == n ? 0 : v[i];
    while (now < h.back()) {
      nxt = id.back();
      res = max(res, h.back() * (i - nxt));
      h.pop_back();
      id.pop_back();
    }
    if (now > h.back()) {
      h.push_back(now);
      id.push_back(nxt);
    }
  }
  return res;
}
```

1.26 LCA.cpp

```
struct LCA {
  int n, root, h;
  vector<vector<int>> g, par;
  vector<int> dep;
  LCA(int _n = 1, int r = 0) : n(_n), root(r), g(_n), dep(_n) {
    h = 1;
    while ((1 << h) <= n) ++h;
    par.assign(h, vector<int>(n, -1));
  }
  LCA(const vector<vector<int>> &_g, const int r = 0)
      : n(_g.size()), root(r), g(_g), dep(_g.size()) {
```

```
    h = 1;
    while ((1 << h) <= n) ++h;
    par.assign(h, vector<int>(n, -1));
    build();
  }
  void add(int a, int b) {
    g[a].push_back(b);
    g[b].push_back(a);
  }
  void dfs(int now, int bf, int d) {
    par[0][now] = bf;
    dep[now] = d;
    for (int &to : g[now])
      if (to != bf) dfs(to, now, d + 1);
  }
  void build() {
    dfs(root, -1, 0);
    for (int i = 0; i + 1 < h; ++i)
      for (int j = 0; j < n; ++j) {
        if (par[i][j] < 0)
          par[i + 1][j] = -1;
        else
          par[i + 1][j] = par[i][par[i][j]];
      }
  }
  int calc(int x, int y) {
    if (dep[x] > dep[y]) swap(x, y);
    for (int i = 0; i < h; ++i)
      if ((dep[y] - dep[x]) >> i & 1) y = par[i][y];
    if (x == y) return x;
    for (int i = h - 1; i >= 0; --i)
      if (par[i][x] != par[i][y]) {
        x = par[i][x];
        y = par[i][y];
      }
    return par[0][y];
  }

  inline int dist(int x, int y) {
    return dep[x] + dep[y] - 2 * dep[calc(x, y)];
  }

  inline int operator[](const int &k) { return (dep.at(k)); }
};
```

1.27 LiChaoTree.cpp

```
// calc min(a*xs[i] + b)
template <class T>
struct LiChaoTree {
  struct Line {
    T a, b;
    Line(T a, T b) : a(a), b(b) {}
    inline T get(T x) const { return a * x + b; }
    inline bool over(const Line &b, const T &x) const {
      return get(x) < b.get(x);
    }
  };
  vector<T> xs;
  vector<Line> seg;
  int xsize;
  LiChaoTree() {}
  LiChaoTree(const vector<T> &x, T INF) : xs(x) {
    xsize = 1;
    while (xsize < xs.size()) xsize <<= 1;
    while (xs.size() < xsize) xs.push_back(xs.back() + 1);
    seg.assign((xsize << 1) - 1, Line(0, INF));
  }
  void update(Line &x, int k, int l, int r) {
    int mid = (l + r) >> 1;
    bool chl = x.over(seg[k], xs[l]), chr = x.over(seg[k], xs[mid]);
    if (chr) swap(seg[k], x);
    if (l + 1 >= r) return;
    if (chl != chr)
      update(x, 2 * k + 1, l, mid);
    else
      update(x, 2 * k + 2, mid, r);
  }
  void update(T a, T b) {
    Line nowl(a, b);
    update(nowl, 0, 0, xsize);
  }
  T query(int k) {
    const T x = xs[k];
    k += xsize - 1;
    T ans = seg[k].get(x);
    while (k > 0) {
      k = (k - 1) >> 1;
      ans = min(ans, seg[k].get(x));
    }
    return ans;
  }
};
```

1.28 manacher.cpp

```
template <class T>
vector<int> manacher(const T &s) {
  vector<int> res;
  int i = 0, j = 0, len = s.size();
  res.assign(len, 0);
  while (i < len) {
    while (i - j >= 0 && i + j < len && s[i - j] == s[i + j]) ++j;
    res[i] = j;
    int k = 1;
    while (i - k >= 0 && k + res[i - k] < j) res[i + k] = res[i -
    ↪  k], ++k;
    i += k, j -= k;
  }
  return res;
}
```

1.29 Matrix.cpp

```
template <class T>
struct Matrix {
  vector<vector<T>> A;
  Matrix() {}
  Matrix(size_t m, size_t n) : A(m, vector<T>(n, 0)) {}
  Matrix(size_t n) : A(n, vector<T>(n, 0)) {}
  size_t height() const { return (A.size()); }
  size_t width() const { return (A[0].size()); }
  inline const vector<T> &operator[](int k) const { return
  ↪  (A.at(k)); }
  inline vector<T> &operator[](int k) { return (A.at(k)); }
  static Matrix E(size_t n) {
    Matrix mat(n);
    for (int i = 0; i < n; ++i) mat[i][i] = 1;
    return (mat);
  }
  Matrix &operator+=(const Matrix &B) {
    size_t m = height(), n = width();
    assert(m == B.height() && n == B.width());
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j) (*this)[i][j] += B[i][j];
    return (*this);
  }
  Matrix &operator-=(const Matrix &B) {
    size_t m = height(), n = width();
    assert(m == B.height() && n == B.width());
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j) (*this)[i][j] -= B[i][j];
    return (*this);
  }
  Matrix &operator*=(const Matrix &B) {
    size_t m = height(), n = B.width(), p = width();
    assert(p == B.height());
    vector<vector<T>> C(m, vector<T>(n, 0));
    for (int i = 0; i < m; ++i)
      for (int k = 0; k < p; ++k) {
        T tmp = (*this)[i][k];
        for (int j = 0; j < n; ++j) C[i][j] += tmp * B[k][j];
      }
    A.swap(C);
    return (*this);
  }
  Matrix &operator^=(long long k) {
    Matrix B = Matrix::E(height());
    while (k) {
      if (k & 1) B *= *this;
      *this *= *this;
      k >>= 1;
    }
    A.swap(B.A);
    return (*this);
  }

  Matrix operator+(const Matrix &B) const { return (Matrix(*this) +=
  ↪  B); }
  Matrix operator-(const Matrix &B) const { return (Matrix(*this) -=
  ↪  B); }
  Matrix operator*(const Matrix &B) const { return (Matrix(*this) *=
  ↪  B); }
  Matrix operator^(const long long k) const { return (Matrix(*this)
  ↪  ^= k); }

  Matrix trans() {
    size_t m = height(), n = width();
    Matrix res(n, m);
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < m; ++j) res[i][j] = (*this)[j][i];
    return res;
  }

  Matrix inv() {
    assert(height() == width());
    size_t n = height();
    Matrix B(n, 2 * n);
```

```
    for (int i = 0; i < n; ++i) {
      B[i][i + n] = 1;
      for (int j = 0; j < n; ++j) B[i][j] = (*this)[i][j];
    }
    for (int i = 0; i < n; ++i) {
      int piv = i;
      for (int j = i; j < n; ++j)
        if (abs(B[j][i]) > abs(B[piv][i])) piv = j;
      // not exist or unique
      assert(abs(B[piv][i]) >= 0);
      swap(B[i], B[piv]);
      for (int j = i + 1; j < 2 * n; ++j) B[i][j] /= B[i][i];
      for (int j = 0; j < n; ++j)
        if (i != j)
          for (int k = i + 1; k < 2 * n; ++k) B[j][k] -= B[j][i] *
          ↪ B[i][k];
    }
    Matrix res(n);
    for (int i = 0; i < n; ++i)
      for (int j = 0; j < n; ++j) res[i][j] = B[i][j + n];
    return res;
  }

  T det() {
    int m = height(), n = width();
    assert(m == n);
    T res = 1;
    Matrix B(m);
    for (int i = 0; i < m; ++i)
      for (int j = 0; j < n; ++j) B[i][j] = (*this)[i][j];
    for (int i = 0; i < n; ++i) {
      int piv = i;
      for (int j = i + 1; j < m; ++j)
        if (B[j][i] != 0) {
          piv = j;
          break;
        }
      // if (abs(B[j][i]) > abs(B[piv][i])) piv = j;
      if (B[piv][i] == 0) return (T)0;
      // if (abs(B[piv][i]) < EPS) return (T)0;  // B[piv][i] < EPS
      if (piv != i) swap(B[i], B[piv]), res = -res;
      res *= B[i][i];
      // for (int j = i + 1; j < m; ++j)
      //   for (int k = n - 1; k >= i; --k) B[j][k] -= B[i][k] *
      ↪ B[j][i] /
      //   B[i][i];
      {
        const T d = (T)1 / B[i][i];
        for (int j = i + 1; j < n; ++j) B[i][j] *= d;
        for (int j = i + 1; j < m; ++j)
          for (int k = i + 1; k < n; ++k) B[j][k] -= B[i][k] *
          ↪ B[j][i];
      }
    }
    return res;
  }

  T cofactor(int r = -1, int c = -1) {
    int m = height(), n = width();
    if (r < 0) r = c = m - 1;
    assert(m == n && m > 1 && r < m && c < n);
    Matrix mat(m - 1, n - 1);
    for (int i = 0, rcnt = 0; i < m; ++i)
      if (i != r) {
        int ccnt = 0;
        for (int j = 0; j < n; ++j)
          if (j != c) mat[rcnt][ccnt++] = (*this)[i][j];
        ++rcnt;
      }
    T res = mat.det();
    if ((r ^ c) & 1) res *= -1;
    return res;
  }

  friend ostream &operator<<(ostream &os, Matrix &p) {
    size_t m = p.height(), n = p.width();
    for (int i = 0; i < m; i++) {
      os << "[";
      for (int j = 0; j < n; j++) {
        os << p[i][j] << (j + 1 == n ? "]\n" : ",");
      }
    }
    return (os);
  }
};
```

1.30 maximum_clique.cpp

```
using ull = unsigned long long;
inline int trail(ull s) { return (s ? __builtin_ctzll(s) : 64); }
// O(3^(n/3)) ? reference:
// https://sites.google.com/site/indy256/algo/bron_kerbosh
template <typename T>
T bron_kerbosh(const vector<ull> &g, ull cur, ull allowed, ull
↪ forbidden,
               const vector<T> &w) {
  if (allowed == 0 && forbidden == 0) {
    T res = 0;
    for (int u = trail(cur); u < g.size(); u += trail(cur >> (u +
    ↪ 1)) + 1)
      res += w[u];
    return res;
  }
  if (allowed == 0) return -1;
  T res = -1;
  int piv = trail(allowed | forbidden);
  ull z = allowed & ~g[piv];
  for (int u = trail(z); u < g.size(); u += trail(z >> (u + 1)) + 1)
  ↪ {
    res = max(res, bron_kerbosh(g, cur | (1ULL << u), allowed & g[u],
                                forbidden & g[u], w));
    allowed ^= 1ULL << u;
    forbidden |= 1ULL << u;
  }
  return res;
}

template <typename T>
T maximum_clique(const vector<vector<int>> G, const vector<T> &w) {
  int n = G.size();
  assert(n < 64);
  vector<ull> g(n, 0);
  for (int i = 0; i < n; ++i)
    for (int j : G[i]) g[i] ^= 1ULL << j;
  return bron_kerbosh<T>(g, 0, (1ULL << n) - 1, 0, w);
}

template <typename T>
T maximal_independent_set(const vector<vector<int>> &G, const
↪ vector<T> &w) {
  int n = G.size();
  assert(n < 64);
  vector<ull> g(n, (1ULL << n) - 1);
  for (int i = 0; i < n; ++i) {
    g[i] ^= 1ULL << i;
    for (int j : G[i]) g[i] ^= 1ULL << j;
  }
  return bron_kerbosh<T>(g, 0, (1ULL << n) - 1, 0, w);
}
```

1.31 ModInt.cpp

```
template <int mod = (int)(1e9 + 7)>
struct ModInt {
  int x;
  constexpr ModInt() : x(0) {}
  constexpr ModInt(int64_t y)
      : x(y >= 0 ? y % mod : (mod - (-y) % mod) % mod) {}
  constexpr ModInt &operator+=(const ModInt &p) noexcept {
    if ((x += p.x) >= mod) x -= mod;
    return *this;
  }
  constexpr ModInt &operator-=(const ModInt &p) noexcept {
    if ((x += mod - p.x) >= mod) x -= mod;
    return *this;
  }
  constexpr ModInt &operator*=(const ModInt &p) noexcept {
    x = (int)(1LL * x * p.x % mod);
    return *this;
  }
  constexpr ModInt &operator/=(const ModInt &p) noexcept {
    *this *= p.inverse();
    return *this;
  }
  constexpr ModInt operator-() const { return ModInt(-x); }
  constexpr ModInt operator+(const ModInt &p) const noexcept {
    return ModInt(*this) += p;
  }
  constexpr ModInt operator-(const ModInt &p) const noexcept {
    return ModInt(*this) -= p;
  }
  constexpr ModInt operator*(const ModInt &p) const noexcept {
    return ModInt(*this) *= p;
  }
  constexpr ModInt operator/(const ModInt &p) const noexcept {
    return ModInt(*this) /= p;
  }
  constexpr bool operator==(const ModInt &p) const noexcept { return
  ↪ x == p.x; }
  constexpr bool operator!=(const ModInt &p) const noexcept { return
  ↪ x != p.x; }
  constexpr ModInt inverse() const noexcept {
    int a = x, b = mod, u = 1, v = 0, t = 0;
    while (b > 0) {
      t = a / b;
      swap(a -= t * b, b);
      swap(u -= t * v, v);
```

```
    }
    return ModInt(u);
  }
  constexpr ModInt pow(int64_t n) const {
    ModInt res(1), mul(x);
    while (n) {
      if (n & 1) res *= mul;
      mul *= mul;
      n >>= 1;
    }
    return res;
  }
  friend constexpr ostream &operator<<(ostream &os, const ModInt &p)
  ↪ noexcept {
    return os << p.x;
  }
  friend constexpr istream &operator>>(istream &is, ModInt &a)
  ↪ noexcept {
    int64_t t = 0;
    is >> t;
    a = ModInt<mod>(t);
    return (is);
  }
  constexpr int get_mod() { return mod; }
};
using mint = ModInt<>;
```

1.32 ModintCombination.cpp

```
template <int mod = (int)(1e9 + 7)>
struct ModInt {
  int x;
  ModInt() : x(0) {}
  ModInt(int64_t y) : x(y >= 0 ? y % mod : (mod - (-y) % mod) % mod)
  ↪ {}
  ModInt &operator+=(const ModInt &p) {
    if ((x += p.x) >= mod) x -= mod;
    return *this;
  }
  ModInt &operator-=(const ModInt &p) {
    if ((x += mod - p.x) >= mod) x -= mod;
    return *this;
  }
  ModInt &operator*=(const ModInt &p) {
    x = (int)(1LL * x * p.x % mod);
    return *this;
  }
  ModInt &operator/=(const ModInt &p) {
    *this *= p.inverse();
    return *this;
  }
  ModInt operator-() const { return ModInt(-x); }
  ModInt operator+(const ModInt &p) const { return ModInt(*this) +=
  ↪ p; }
  ModInt operator-(const ModInt &p) const { return ModInt(*this) -=
  ↪ p; }
  ModInt operator*(const ModInt &p) const { return ModInt(*this) *=
  ↪ p; }
  ModInt operator/(const ModInt &p) const { return ModInt(*this) /=
  ↪ p; }
  bool operator==(const ModInt &p) const { return x == p.x; }
  bool operator!=(const ModInt &p) const { return x != p.x; }
  ModInt inverse() const {
    int a = x, b = mod, u = 1, v = 0, t;
    while (b > 0) {
      t = a / b;
      swap(a -= t * b, b);
      swap(u -= t * v, v);
    }
    return ModInt(u);
  }
  ModInt pow(int64_t n) const {
    ModInt res(1), mul(x);
    while (n) {
      if (n & 1) res *= mul;
      mul *= mul;
      n >>= 1;
    }
    return res;
  }
  friend ostream &operator<<(ostream &os, const ModInt &p) { return
  ↪ os << p.x; }
  friend istream &operator>>(istream &is, ModInt &a) {
    int64_t t;
    is >> t;
    a = ModInt<mod>(t);
    return (is);
  }
  static int get_mod() { return mod; }
};

struct Combination {
  vector<ModInt<>> _fact, _rfact, _inv;
  Combination(long long nsize = 5000000)
      : _fact(nsize + 1), _rfact(nsize + 1), _inv(nsize + 1) {
    _fact[0] = _rfact[nsize] = _inv[0] = 1;
    for (int i = 1; i <= nsize; i++) _fact[i] = _fact[i - 1] * i;
    _rfact[nsize] /= _fact[nsize];
    for (int i = nsize - 1; i >= 0; i--) _rfact[i] = _rfact[i + 1] *
    ↪ (i + 1);
    for (int i = 1; i <= nsize; i++) _inv[i] = _rfact[i] * _fact[i -
    ↪ 1];
  }
  inline ModInt<> fact(int k) const { return _fact[k]; }

  inline ModInt<> rfact(int k) const { return _rfact[k]; }

  inline ModInt<> inv(int k) const { return _inv[k]; }

  ModInt<> P(int n, int r) const {
    if (r < 0 || n < r) return 0;
    return fact(n) * rfact(n - r);
  }

  ModInt<> C(int p, int q) const {
    if (q < 0 || p < q) return 0;
    return fact(p) * rfact(q) * rfact(p - q);
  }

  ModInt<> largeC(long long p, long long q) const {
    if (q < 0 || p < q) return 0;
    if (q >= (long long)_fact.size()) q = p - q;
    // if (q >= (long long)5000) q = p - q;
    ModInt<> res = rfact(q);
    for (int i = 0; i < q; ++i) res *= p - i;
    return res;
  }

  // n types,choose r
  ModInt<> H(int n, int r) const {
    if (n < 0 || r < 0) return (0);
    return r == 0 ? 1 : C(n + r - 1, r);
  }

  ModInt<> largeH(long long n, long long r) const {
    if (n < 0 || r < 0) return (0);
    return r == 0 ? 1 : largeC(n + r - 1, r);
  }

  ModInt<> Catalan(int n) {
    // C(2n,n) / (n + 1)
    return fact(2 * n) * rfact(n + 1) * rfact(n);
  }
};
using mint = ModInt<>;
```

1.33 ModIntR.cpp

```
// ModIntR::set_mod()
struct ModIntR {
  int x;
  ModIntR() : x(0) {}
  ModIntR(int64_t y) : x(y >= 0 ? y % mod() : (mod() - (-y) % mod())
  ↪ % mod()) {}
  static int &mod() {
    static int mod = 0;
    return mod;
  }
  static void set_mod(int md) { mod() = md; }
  ModIntR &operator+=(const ModIntR &p) {
    if ((x += p.x) >= mod()) x -= mod();
    return *this;
  }
  ModIntR &operator-=(const ModIntR &p) {
    if ((x += mod() - p.x) >= mod()) x -= mod();
    return *this;
  }

  ModIntR &operator*=(const ModIntR &p) {
    long long m = mod();
    x = (int)(1LL * x * p.x % m);
    return *this;
  }

  ModIntR &operator/=(const ModIntR &p) {
    *this *= p.inverse();
    return *this;
  }

  ModIntR operator-() const { return ModIntR(-x); }

  ModIntR operator+(const ModIntR &p) const { return ModIntR(*this)
  ↪ += p; }

  ModIntR operator-(const ModIntR &p) const { return ModIntR(*this)
  ↪ -= p; }
```

```
  ModIntR operator*(const ModIntR &p) const { return ModIntR(*this)
  ↪  *= p; }

  ModIntR operator/(const ModIntR &p) const { return ModIntR(*this)
  ↪  /= p; }

  bool operator==(const ModIntR &p) const { return x == p.x; }

  bool operator!=(const ModIntR &p) const { return x != p.x; }

  ModIntR inverse() const {
    int a = x, b = mod(), u = 1, v = 0, t;
    while (b > 0) {
      t = a / b;
      swap(a -= t * b, b);
      swap(u -= t * v, v);
    }
    return ModIntR(u);
  }

  ModIntR pow(int64_t n) const {
    ModIntR res(1), mul(x);
    while (n) {
      if (n & 1) res *= mul;
      mul *= mul;
      n >>= 1;
    }
    return res;
  }

  friend ostream &operator<<(ostream &os, const ModIntR &p) {
    return os << p.x;
  }

  friend istream &operator>>(istream &is, ModIntR &a) {
    int64_t t;
    is >> t;
    a = ModIntR(t);
    return (is);
  }
};
```

1.34 mod_log.cpp

```
// as + bt = GCD(a,b) a,b:const s,t:var(any)
// return GCD(a,b)
long long extGCD(long long a, long long b, long long &s, long long
↪  &t) {
  s = 1, t = 0;
  long long u = 0, v = 1;
  while (b) {
    long long tmp = a / b;
    a -= b * tmp;
    s -= u * tmp;
    t -= v * tmp;
    swap(s, u);
    swap(t, v);
    swap(a, b);
  }
  return a;
}
// (mod)x+ay=1, calculate y -> a^-1 (mod m) (a,m : coprime)
long long calcinv(long long a, long long m) {
  long long s, t;
  extGCD(a, m, s, t);
  return (s + m) % m;
}

// calc x s.t. a**x = b(mod p)
long long mod_log(long long a, long long b, long long p) {
  assert(p > 0);
  long long sqp = sqrt(p) + 2;
  unordered_map<long long, long long> bs;
  long long base = 1;                       // a if deny 0
  {                                         // baby-step
    for (long long i = 0; i < sqp; ++i) {   // i = 1 if deny 0
      if (!bs.count(base)) bs[base] = i;
      (base *= a) %= p;
    }
  }
  {  // giant-step
    long long rev = calcinv(base, p);
    for (long long i = 0; i <= sqp; ++i) {
      if (bs.count(b)) return bs[b] + i * sqp;
      (b *= rev) %= p;
    }
  }
  return -1;
}
```

1.35 NTT.cpp

```
template <int mod = 998244353>
struct NTT {
  int base, maxb, root;
  vector<int> rv, roots, invr;
  NTT() : base(1), rv({0, 1}), roots({0, 1}), invr({0, 1}) {
    assert(mod >= 3 && mod & 1);
    int tmp = mod - 1;
    maxb = 0;
    while (!(tmp & 1)) tmp >>= 1, ++maxb;
    root = 2;
    while (mpow(root, (mod - 1) >> 1) == 1) ++root;
    assert(mpow(root, mod - 1) == 1);
    root = mpow(root, (mod - 1) >> maxb);
  }

  inline int mpow(int x, int n) {
    int res = 1;
    while (n) {
      if (n & 1) res = mul(res, x);
      x = mul(x, x);
      n >>= 1;
    }
    return res;
  }

  inline int inv(int x) {
    assert(x != 0);
    return mpow(x, mod - 2);
  }

  inline int add(int x, int y) {
    if ((x += y) >= mod) x -= mod;
    return x;
  }

  inline int mul(int x, int y) { return (int)(1LL * x * y % mod); }

  void ensure_base(int nb) {
    if (nb <= base) return;
    rv.resize(1 << nb);
    roots.resize(1 << nb);
    invr.resize(1 << nb);
    for (int i = 0; i < (1 << nb); ++i)
      rv[i] = (rv[i >> 1] >> 1) + ((i & 1) << (nb - 1));
    assert(nb <= maxb);
    while (base < nb) {
      int z = mpow(root, 1 << (maxb - 1 - base)), invz = inv(z);
      for (int i = 1 << (base - 1); i < (1 << base); ++i) {
        roots[i << 1] = roots[i];
        roots[(i << 1) + 1] = mul(roots[i], z);
        invr[i << 1] = invr[i];
        invr[(i << 1) + 1] = mul(invr[i], invz);
      }
      ++base;
    }
  }
  void ntt(vector<int> &a, int n, bool sg = 0) {
    assert((n & (n - 1)) == 0);
    int dif = base - __builtin_ctz(n);
    for (int i = 0; i < n; ++i)
      if (i < (rv[i] >> dif)) swap(a[i], a[rv[i] >> dif]);
    for (int k = 1; k < n; k <<= 1)
      for (int i = 0; i < n; i += 2 * k)
        for (int j = 0; j < k; ++j) {
          int z = mul(a[i + j + k], (sg ? roots[j + k] : invr[j +
          ↪  k]));
          a[i + j + k] = add(a[i + j], mod - z);
          a[i + j] = add(a[i + j], z);
        }
    int invn = inv(n);
    if (sg)
      for (int i = 0; i < n; ++i) a[i] = mul(a[i], invn);
  }
  template <class T>
  vector<T> multiply(const vector<T> &a, const vector<T> &b) {
    int need = a.size() + b.size() - 1;
    int nb = 1;
    while ((1 << nb) < need) ++nb;
    ensure_base(nb);
    int sz = 1 << nb;
    vector<int> fa(sz, 0), fb(sz, 0);
    for (int i = 0; i < sz; ++i) {
      if (i < a.size()) fa[i] = a[i];
      if (i < b.size()) fb[i] = b[i];
    }
    ntt(fa, sz);
    ntt(fb, sz);
    for (int i = 0; i < sz; ++i) fa[i] = mul(fa[i], fb[i]);
    ntt(fa, sz, 1);
    vector<T> res(need);
    for (int i = 0; i < need; ++i) res[i] = fa[i];
    return res;
  }
```

```
};
```

1.36 NTT_ModInt.cpp

```
struct NTT_ModInt {
  int base, maxb, mod;
  mint root;
  vector<int> rv;
  vector<mint> roots, invr;
  NTT_ModInt()
      : base(1),
        mod(mint().get_mod()),
        rv({0, 1}),
        roots({0, 1}),
        invr({0, 1}) {
    int tmp = mod - 1;
    maxb = 0;
    while (!(tmp & 1)) tmp >>= 1, ++maxb;
    root = 2;
    while (root.pow((mod - 1) >> 1) == 1) root += 1;
    assert(root.pow(mod - 1) == 1);
    root = root.pow((mod - 1) >> maxb);
  }

  void ensure_base(int nb) {
    if (nb <= base) return;
    rv.resize(1 << nb);
    roots.resize(1 << nb);
    invr.resize(1 << nb);
    for (int i = 0; i < (1 << nb); ++i)
      rv[i] = (rv[i >> 1] >> 1) + ((i & 1) << (nb - 1));
    assert(nb <= maxb);
    while (base < nb) {
      mint z = root.pow(1 << (maxb - 1 - base)), invz = z.inverse();
      for (int i = 1 << (base - 1); i < (1 << base); ++i) {
        roots[i << 1] = roots[i];
        roots[(i << 1) + 1] = roots[i] * z;
        invr[i << 1] = invr[i];
        invr[(i << 1) + 1] = invr[i] * invz;
      }
      ++base;
    }
  }
  void ntt(vector<mint> &a, int n, bool sg = 0) {
    assert((n & (n - 1)) == 0);
    int dif = base - __builtin_ctz(n);
    for (int i = 0; i < n; ++i)
      if (i < (rv[i] >> dif)) swap(a[i], a[rv[i] >> dif]);
    for (int k = 1; k < n; k <<= 1)
      for (int i = 0; i < n; i += 2 * k)
        for (int j = 0; j < k; ++j) {
          mint z = a[i + j + k] * (sg ? roots[j + k] : invr[j + k]);
          a[i + j + k] = a[i + j] - z;
          a[i + j] += z;
        }
    mint invn = mint(n).inverse();
    if (sg)
      for (int i = 0; i < n; ++i) a[i] *= invn;
  }
  vector<mint> multiply(vector<mint> a, vector<mint> b) {
    int need = a.size() + b.size() - 1;
    int nb = 1;
    while ((1 << nb) < need) ++nb;
    ensure_base(nb);
    int sz = 1 << nb;
    a.resize(sz);
    b.resize(sz);
    ntt(a, sz);
    ntt(b, sz);
    for (int i = 0; i < sz; ++i) a[i] *= b[i];
    ntt(a, sz, 1);
    a.resize(need);
    return a;
  }
};
```

1.37 popcount.cpp

```
unsigned long long popcount(unsigned long long x) {
  x = ((x & 0xaaaaaaaaaaaaaaaaUL) >> 1) + (x & 0x5555555555555555UL);
  x = ((x & 0xccccccccccccccccUL) >> 2) + (x & 0x3333333333333333UL);
  x = ((x & 0xf0f0f0f0f0f0f0f0UL) >> 4) + (x & 0x0f0f0f0f0f0f0f0fUL);
  x = ((x & 0xff00ff00ff00ff00UL) >> 8) + (x & 0x00ff00ff00ff00ffUL);
  x = ((x & 0xffff0000ffff0000UL) >> 16) + (x &
  ↪  0x0000ffff0000ffffUL);
  x = ((x & 0xffffffff00000000UL) >> 32) + (x &
  ↪  0x00000000ffffffffUL);
  return x;
}
// 1000 -> 3
inline int trail(unsigned long long s) { return (s ?
↪  __builtin_ctzll(s) : 64); }
// 111 -> 61
inline int lead(unsigned long long s) { return (s ?
↪  __builtin_clzll(s) : 64); }
```

1.38 PrimalDual.cpp

```
template <class T>
struct Primal_Dual {
  using Pa = pair<T, int>;
  int infinity = (int)(1e9);
  struct edge {
    int to;
    T cap, cost;
    int rev;
  };
  int v;
  vector<vector<edge>> edges;
  vector<T> h;
  vector<T> dist;
  vector<int> prevv, preve;
  Primal_Dual(int vsize = 1) {
    v = vsize;
    edges.resize(v);
    h.resize(v);
    dist.resize(v);
    prevv.resize(v);
    preve.resize(v);
  }
  bool add(int from, int to, T cap, T cost) {
    edges[from].push_back((edge){to, cap, cost,
    ↪  (int)edges[to].size()});
    edges[to].push_back((edge){from, 0, -cost,
    ↪  (int)edges[from].size() - 1});
    return 1;
  }
  T solve(int s, int t, T f) {
    T ans = 0;
    h.assign(v, 0);
    while (f > 0) {
      priority_queue<Pa, vector<Pa>, greater<Pa>> qu;
      dist.assign(v, infinity);
      dist[s] = 0;
      qu.push({0, s});
      while (!qu.empty()) {
        Pa now = qu.top();
        qu.pop();
        int nowv = now.second;
        if (dist[nowv] < now.first) continue;
        for (int i = 0; i < (int)edges[nowv].size(); ++i) {
          edge &e = edges[nowv][i];
          if (e.cap > 0 &&
              dist[e.to] > dist[nowv] + e.cost + h[nowv] - h[e.to]) {
            dist[e.to] = dist[nowv] + e.cost + h[nowv] - h[e.to];
            prevv[e.to] = nowv;
            preve[e.to] = i;
            qu.push({dist[e.to], e.to});
          }
        }
      }
      if (dist[t] == infinity) return -1;
      for (int i = 0; i < v; ++i) h[i] += dist[i];
      T d = f;
      for (int i = t; i != s; i = prevv[i])
        d = min(d, edges[prevv[i]][preve[i]].cap);
      f -= d;
      ans += d * h[t];
      for (int i = t; i != s; i = prevv[i]) {
        edge &e = edges[prevv[i]][preve[i]];
        e.cap -= d;
        edges[i][e.rev].cap += d;
      }
    }
    return ans;
  }
};
```

1.39 prime.cpp

```
struct Prime {
  long long n = 0;
  vector<bool> ch;
  vector<long long> ary;
  Prime(long long N = 2000000) {
    ch.resize(N + 1);
    ch[0] = ch[1] = 1;
    for (int i = 2; i <= N; ++i)
      if (!ch[i]) {
        ary.push_back(i);
        for (int j = i; 1LL * i * j <= N; ++j) ch[i * j] = 1;
      }
    n = ary.size();
  }
  inline const bool isprime(int n) { return !ch[n]; }
};
```

1.40 quotient_range.cpp

```cpp
// return (q, l, r): n / i == q(l <= i < r)
template <class T>
vector<tuple<T, T, T>> quotient_range(T n) {
  vector<tuple<T, T, T>> res;
  T l = T(1);
  while (l <= n) {
    T r = n / (n / l) + 1;
    res.emplace_back(n / l, l, r);
    l = r;
  }
  return res;
}
```

1.41 RollingHash.cpp

```cpp
struct RollingHash {
  int Mod, Base;
  vector<int> pow;
  vector<vector<int>> hash;
  // mod : 1e9 + 9,base : 1007
  RollingHash(const int len = 3000000, const int mod = (int)(1e9 +
  ↪ 7),
              const int base = 1009) {
    Mod = mod;
    Base = base;
    pow.assign(len + 1, 0);
    pow[0] = 1;
    for (int i = 1; i <= len; ++i) pow[i] = 1LL * pow[i - 1] * Base
    ↪ % Mod;
  }
  template <class T>
  int add(const T &s) {
    int id = hash.size();
    hash.push_back(vector<int>());
    sethash(id, s);
    return id;
  }
  template <class T>
  void sethash(const int id, const T &s) {
    assert(id < (int)hash.size());
    int len = s.size();
    hash[id].resize(len + 1, 0);
    for (int i = 0; i < len; ++i) {
      hash[id][i + 1] = 1LL * hash[id][i] * Base % Mod;
      if ((hash[id][i + 1] += s[i] + 1) >= Mod) hash[id][i + 1] -=
      ↪ Mod;
    }
  }
  // [l,r),0-indexed
  inline int calchash(const int &id, const int &l, const int &r)
  ↪ const {
    assert(r >= l);
    int res = hash[id][r];
    res += Mod - 1LL * hash[id][l] * pow[r - l] % Mod;
    if (res >= Mod) res -= Mod;
    return res;
  }
  inline bool issame(const int &idl, const int &ll, const int &lr,
                     const int &idr, const int &rl, const int &rr)
                     ↪ const {
    return calchash(idl, ll, lr) == calchash(idr, rl, rr);
  }
};
```

1.42 run_enumerate.cpp

```cpp
// use z_algorithm
template <class T>
void run_enumerate(int l, int r, const T& s,
                   vector<vector<pair<int, int>>>& res) {
  if (r - l <= 1) return;
  int m = (l + r) >> 1;
  run_enumerate(l, m, s, res);
  run_enumerate(m, r, s, res);

  auto func = [&](bool rev = 0) {
    T t, tl, tr;
    copy(s.begin() + l, s.begin() + r, back_inserter(t));
    if (rev) {
      reverse(t.begin(), t.end());
      m = l + r - m;
    }
    int len = r - l, mlen = m - l;
    copy(t.begin(), t.begin() + mlen, back_inserter(tl));
    reverse(tl.begin(), tl.end());
    copy(t.begin() + mlen, t.end(), back_inserter(tr));
    copy(t.begin(), t.end(), back_inserter(tr));
    auto zl = z_algorithm(tl), zr = z_algorithm(tr);
    zl.push_back(0);
    for (int k = 1; k <= mlen; ++k) {
      int li = m - k - zl[k], ri = m + min(r - m, zr[len - k]);
      if (rev) {
        swap(li, ri);
        li = l + r - li;
        ri = l + r - ri;
      }
      if (ri - li < 2 * k) continue;
      if (li > 0 && s[li - 1] == s[li - 1 + k]) continue;
      if (ri < s.size() && s[ri] == s[ri - k]) continue;
      res[li].emplace_back(ri, k);
    }
  };
  func();
  func(1);
}

template <class T>
vector<vector<pair<int, int>>> run_enumerate(const T& s) {
  int len = s.size();
  vector<vector<pair<int, int>>> run(len), res(len);
  run_enumerate(0, len, s, run);
  for (int i = 0; i < len; ++i) {
    int rlen = run[i].size();
    sort(run[i].begin(), run[i].end());
    for (int j = 0; j < rlen; ++j)
      if (j == 0 || run[i][j].first != run[i][j - 1].first)
        res[i].push_back(run[i][j]);
  }
  return res;
}
```

1.43 SAnnealing.cpp

```cpp
#define LIMTIME 1930
const double STTEMP = 250, ENDTEMP = 5;
const long long SAINF = 1e18;

unsigned int xorshift() {
  static unsigned int nowx = 123456789, nowy = 362436069, nowz =
  ↪ 521288629,
                      noww = 88675123;
  unsigned int temp;
  temp = (nowx ^ (nowx << 11));
  nowx = nowy;
  nowy = nowz;
  nowz = noww;
  return (noww = (noww ^ (noww >> 19)) ^ (temp ^ (temp >> 8)));
}
void SA() {
  auto start = chrono::system_clock::now();
  auto nowt = chrono::system_clock::now();
  auto dur = nowt - start;
  auto msec =
  ↪ chrono::duration_cast<chrono::milliseconds>(dur).count();
  while (msec < LIMTIME) {
    long long BFSCORE = 0, NOWSCORE = 0;
    /* SA pattern
    double TEMPERATURE = STTEMP + (ENDTEMP - STTEMP) * msec /
↪   LIMTIME;
    double PROBABLE = exp((NOWSCORE - BFSCORE) / TEMPERATURE);
    if (PROBABLE <= (xorshift() % SAINF) / (double)SAINF) {
      // return to before state
    }
    //*/
    //* climbing pattern
    if (NOWSCORE < BFSCORE) {
      // return to before score
    }
    //*/
    nowt = chrono::system_clock::now();
    dur = nowt - start;
    msec = chrono::duration_cast<chrono::milliseconds>(dur).count();
  }
}
```

1.44 SegmentSet.cpp

```cpp
// [l,r]
template <class T, T inf = numeric_limits<T>::max()>
struct SegmentSet {
  using Pt = pair<T, T>;
  set<Pt> st;
  SegmentSet() {
    st.clear();
    st.emplace(-inf, -inf);
    st.emplace(inf, inf);
  }

  bool covered(T l, T r) {
    auto [x, y] = *prev(st.upper_bound(Pt(l, inf)));
    return x <= l && r <= y;
  }
  bool exist(T x) { return covered(x, x); }
```

```
  T insert(int l, int r) {
    auto it = prev(st.upper_bound(Pt(l, inf)));
    // cover
    if (it->first <= l && r <= it->second) return T(0);
    T er = T(0);
    if (l - 1 <= it->second) {
      l = it->first;
      er += it->second - it->first + 1;
      it = st.erase(it);
    } else
      it = next(it);
    while (it->first <= r + 1) {
      r = max(r, it->second);
      er += it->second - it->first + 1;
      it = st.erase(it);
    }
    st.emplace(l, r);
    return r - l + 1 - er;
  }
  T insert(int x) { return insert(x, x); }

  T erase(int l, int r) {
    auto it = prev(st.upper_bound(Pt(l, inf)));
    // cover
    if (it->first <= l && r <= it->second) {
      if (it->first < l) st.emplace(it->first, l - 1);
      if (r < it->second) st.emplace(r + 1, it->second);
      st.erase(it);
      return r - l + 1;
    }
    T res = T(0);
    if (l <= it->second) {
      res += it->second - l + 1;
      if (it->first < l) st.emplace(it->first, l - 1);
      it = st.erase(it);
    } else
      it = next(it);
    while (it->first <= r) {
      res += min(r, it->second) - it->first + 1;
      if (r < it->second) st.emplace(r + 1, it->second);
      it = st.erase(it);
    }
    return res;
  }
  T erase(int x) { return erase(x, x); }

  // [x,inf)
  T mex(T x = T(0)) {
    auto [l, r] = *prev(st.upper_bound(Pt(x, inf)));
    // cover
    if (x <= r) return r + 1;
    return x;
  }

  friend ostream &operator<<(ostream &os, SegmentSet &p) {
    for (auto &[l, r] : p.st)
      if (abs(l) != inf) os << "[" << l << ", " << r << "] ";
    return (os);
  }
};
```

1.45 SegmentTree.cpp

```
// 0-indexed
template <class T>
struct SegmentTree {
  // a,b,c: T, e:T(unit)
  // abc = (ab)c = a(bc)
  // ae = ea = a
  typedef function<T(T, T)> F;
  int n;
  F f;
  T unit;
  vector<T> dat;
  SegmentTree(){};
  SegmentTree(int _n, F f, T t) : f(f), unit(t) { init(_n); }
  SegmentTree(const vector<T> &v, F f, T t) : f(f), unit(t) {
    int _n = v.size();
    init(_n);
    for (int i = 0; i < _n; ++i) dat[n + i] = v[i];
    for (int i = n - 1; i; --i) dat[i] = f(dat[i << 1], dat[(i << 1)
    ↪  | 1]);
  }
  void init(int _n) {
    n = 1;
    while (n < _n) n <<= 1;
    dat.assign(n << 1, unit);
  }

  // "go up" process
  void update(int k, T newdata) {
    dat[k += n] = newdata;
    while (k >>= 1) dat[k] = f(dat[k << 1], dat[(k << 1) | 1]);
  }

  // [a,b)
  T query(int a, int b) {
    T vl = unit, vr = unit;
    for (int l = a + n, r = b + n; l < r; l >>= 1, r >>= 1) {
      if (l & 1) vl = f(vl, dat[l++]);
      if (r & 1) vr = f(dat[--r], vr);
    }
    return f(vl, vr);
  }
  // require: func(unit) == false
  // min left: st <= res && func(seg.query(st,res + 1))
  template <typename C>
  int find_left(int st, C &func, T &acc, int k, int l, int r) {
    if (l + 1 == r) {
      acc = f(acc, dat[k]);
      return func(acc) ? l : -1;
    }
    int mid = (l + r) >> 1;
    if (mid <= st) return find_left(st, func, acc, (k << 1) | 1,
    ↪  mid, r);
    if (st <= l && !func(f(acc, dat[k]))) {
      acc = f(acc, dat[k]);
      return -1;
    }
    int nres = find_left(st, func, acc, (k << 1), l, mid);
    if (~nres) return nres;
    return find_left(st, func, acc, (k << 1) | 1, mid, r);
  }
  template <typename C>
  int find_left(int st, C &func) {
    T acc = unit;
    return find_left(st, func, acc, 1, 0, n);
  }

  // max right: res <= st && func(seg.query(res - 1,st))
  template <typename C>
  int find_right(int st, C &func, T &acc, int k, int l, int r) {
    if (l + 1 == r) {
      acc = f(dat[k], acc);
      return func(acc) ? r : -1;
    }
    int mid = (l + r) >> 1;
    if (st <= mid) return find_right(st, func, acc, k << 1, l, mid);
    if (r <= st && !func(f(dat[k], acc))) {
      acc = f(dat[k], acc);
      return -1;
    }
    int nres = find_right(st, func, acc, (k << 1) | 1, mid, r);
    if (~nres) return nres;
    return find_right(st, func, acc, k << 1, l, mid);
  }
  template <typename C>
  int find_right(int st, C &func) {
    T acc = unit;
    return find_right(st, func, acc, 1, 0, n);
  }
};
```

1.46 SegmentTree2d.cpp

```
// 0-indexed
template <class T>
struct SegmentTree2d {
  // a,b,c: T, e:T(unit)
  // abc = (ab)c = a(bc)
  // ae = ea = a
  typedef function<T(T, T)> F;
  int h, w;
  F f;
  T unit;
  vector<vector<T>> dat;
  SegmentTree2d(){};
  SegmentTree2d(int _h, int _w, F f, T t) : f(f), unit(t) { init(_h,
  ↪  _w); }
  SegmentTree2d(const vector<vector<T>> &v, F f, T t) : f(f),
  ↪  unit(t) {
    int _h = v.size(), _w = _h ? v[0].size() : 0;
    init(_h, _w);
    for (int i = 0; i < _h; ++i) {
      for (int j = 0; j < _w; ++j) dat[h + i][w + j] = v[i][j];
      for (int j = w - 1; j; --j)
        dat[h + i][j] = f(dat[h + i][j << 1], dat[h + i][(j << 1) |
        ↪  1]);
    }
    for (int i = h - 1; i; --i)
      for (int j = (w << 1) - 1; j; --j)
        dat[i][j] = f(dat[i << 1][j], dat[(i << 1) | 1][j]);
  }
  void init(int _h, int _w) {
    h = 1, w = 1;
    while (h < _h) h <<= 1;
    while (w < _w) w <<= 1;
    dat.assign(h << 1, vector<T>(w << 1, unit));
  }
```

```
  // "go up" process
  void update(int r, int c, T newdata) {
    dat[r += h][c += w] = newdata;
    while (r) {
      int k = c;
      while (k >>= 1) dat[r][k] = f(dat[r][k << 1], dat[r][(k << 1)
      ↪  | 1]);
      r >>= 1, dat[r][c] = f(dat[r << 1][c], dat[(r << 1) | 1][c]);
    }
  }
  // [u,d) && [l,r)
  T query(int u, int d, int l, int r) {
    T vl = unit, vr = unit;
    for (int a = u + h, b = d + h; a < b; a >>= 1, b >>= 1) {
      if (a & 1) vl = f(vl, query_in(a++, l, r));
      if (b & 1) vr = f(query_in(--b, l, r), vr);
    }
    return f(vl, vr);
  }
  // [a,b)
  T query_in(int id, int a, int b) {
    T vl = unit, vr = unit;
    for (int l = a + w, r = b + w; l < r; l >>= 1, r >>= 1) {
      if (l & 1) vl = f(vl, dat[id][l++]);
      if (r & 1) vr = f(dat[id][--r], vr);
    }
    return f(vl, vr);
  }
};
```

1.47 SegmentTreeLaze.cpp

```
// 0-indexed
template <class T, class E>
struct SegmentTreeLaze {
  // a,b:T c,d:E e:E(unit)
  // g(f(a,b),c) = f(g(a,c),g(b,c))
  // g(g(a,c),d) = g(a,h(c,d))
  // g(a,e) = a
  typedef function<T(T, T)> F;
  typedef function<T(T, E)> G;
  typedef function<E(E, E)> H;
  int n, height;
  F f;
  G g;
  H h;
  T tunit;
  E eunit;
  vector<T> dat;
  vector<E> laz;
  SegmentTreeLaze(){};
  SegmentTreeLaze(int newn, F f, G g, H h, T nt, E ne)
      : f(f), g(g), h(h), tunit(nt), eunit(ne) {
    init(newn);
  }
  SegmentTreeLaze(const vector<T> &v, F f, G g, H h, T nt, E ne)
      : f(f), g(g), h(h), tunit(nt), eunit(ne) {
    int _n = v.size();
    init(v.size());
    for (int i = 0; i < _n; ++i) dat[n + i] = v[i];
    for (int i = n - 1; i; --i) dat[i] = f(dat[i << 1], dat[(i << 1)
    ↪  | 1]);
  }
  void init(int newn) {
    n = 1, height = 0;
    while (n < newn) n <<= 1, ++height;
    dat.assign(n << 1, tunit);
    laz.assign(n << 1, eunit);
  }

  inline T reflect(int k) {
    return laz[k] == eunit ? dat[k] : g(dat[k], laz[k]);
  }

  inline void eval(int k) {
    if (laz[k] == eunit) return;
    laz[k << 1] = h(laz[k << 1], laz[k]);
    laz[(k << 1) | 1] = h(laz[(k << 1) | 1], laz[k]);
    dat[k] = reflect(k);
    laz[k] = eunit;
  }

  inline void thrust(int k) {
    for (int i = height; i; --i) eval(k >> i);
    // reset query
    // dat[k] = reflect(k);
    // laz[k] = eunit;
  }

  void recalc(int k) {
    while (k >>= 1) dat[k] = f(reflect(k << 1), reflect((k << 1) |
    ↪  1));
  }

  // [a,b)
  void update(int a, int b, E newdata) {
    thrust(a += n);
    thrust(b += n - 1);
    for (int l = a, r = b + 1; l < r; l >>= 1, r >>= 1) {
      if (l & 1) laz[l] = h(laz[l], newdata), l++;
      if (r & 1) --r, laz[r] = h(laz[r], newdata);
    }
    recalc(a);
    recalc(b);
  }

  void set_val(int k, T newdata) {
    thrust(k += n);
    dat[k] = newdata;
    laz[k] = eunit;
    recalc(k);
  }

  // [a,b)
  T query(int a, int b) {
    thrust(a += n);
    thrust(b += n - 1);
    T vl = tunit, vr = tunit;
    for (int l = a, r = b + 1; l < r; l >>= 1, r >>= 1) {
      if (l & 1) vl = f(vl, reflect(l++));
      if (r & 1) vr = f(reflect(--r), vr);
    }
    return f(vl, vr);
  }

  // require: func(unit) == false
  // min left: st <= res && func(seg.query(st,res + 1))
  template <typename C>
  int find_left(int st, C &func, T &acc, int k, int l, int r) {
    if (l + 1 == r) {
      acc = f(acc, reflect(k));
      return func(acc) ? l : -1;
    }
    eval(k);
    int mid = (l + r) >> 1;
    if (mid <= st) return find_left(st, func, acc, (k << 1) | 1,
    ↪  mid, r);
    if (st <= l && !func(f(acc, dat[k]))) {
      acc = f(acc, dat[k]);
      return -1;
    }
    int nres = find_left(st, func, acc, (k << 1), l, mid);
    if (~nres) return nres;
    return find_left(st, func, acc, (k << 1) | 1, mid, r);
  }
  template <typename C>
  int find_left(int st, C &func) {
    T acc = tunit;
    return find_left(st, func, acc, 1, 0, n);
  }

  // max right: res <= st && func(seg.query(res - 1,st))
  template <typename C>
  int find_right(int st, C &func, T &acc, int k, int l, int r) {
    if (l + 1 == r) {
      acc = f(reflect(k), acc);
      return func(acc) ? r : -1;
    }
    eval(k);
    int mid = (l + r) >> 1;
    if (st <= mid) return find_right(st, func, acc, k << 1, l, mid);
    if (r <= st && !func(f(dat[k], acc))) {
      acc = f(dat[k], acc);
      return -1;
    }
    int nres = find_right(st, func, acc, (k << 1) | 1, mid, r);
    if (~nres) return nres;
    return find_right(st, func, acc, k << 1, l, mid);
  }
  template <typename C>
  int find_right(int st, C &func) {
    T acc = tunit;
    return find_right(st, func, acc, 1, 0, n);
  }
};
```

1.48 SlopeTrick.cpp

```
template <typename T = int>
struct SlopeTrick {
  T minf, inf, addl, addr;
  priority_queue<T> pql;
  priority_queue<T, vector<T>, greater<T>> pqr;
  SlopeTrick(T _inf = 1e9) : minf(0), inf(_inf), addl(0), addr(0) {
    pql.push(-inf);
    pqr.push(inf);
  }
```

```
  void pushL(const T &a) { pql.push(a - addl); }
  T topL() const { return pql.top() + addl; }
  T popL() {
    T res = topL();
    pql.pop();
    return res;
  }
  void pushR(const T &a) { pqr.push(a - addr); }
  T topR() const { return pqr.top() + addr; }
  T popR() {
    T res = topR();
    pqr.pop();
    return res;
  }
  size_t size() { return pql.size() + pqr.size(); }

  // min f(x)
  T get_minf() { return minf; }
  T get_minx() { return pql.top(); }
  T get_maxx() { return pqr.top(); }
  T get_fx(const T &x) {
    T res = minf;
    while (pql.size()) res += max(T(0), popL() - x);
    while (pqr.size()) res += max(T(0), x - popR());
    return res;
  }
  // f(x) += a
  void add_all(const T &a) { minf += a; }
  // add f(x) = max(0, x-a), _/
  void add_xa(const T &a) {
    minf += max(T(0), topL() - a);
    pushL(a);
    pushR(popL());
  }
  // add f(x) = max(0, a-x), \_
  void add_ax(const T &a) {
    minf += max(T(0), a - topR());
    pushR(a);
    pushL(popR());
  }
  // add f(x) = abs(x-a) = max(0, x-a) + max(0, a-x), \/
  void add_abs(const T &a) { add_xa(a), add_ax(a); }
  // f(x) := min f(y) (y >= x), \/ -> _/
  void clear_left() {
    priority_queue<T> newpq;
    swap(pql, newpq);
  }
  // f(x) := min f(y) (y <= x), \/ -> \_
  void clear_right() {
    priority_queue<T, vector<T>, greater<T>> newpq;
    swap(pqr, newpq);
  }
  // f(x) = min f(y) (x-b <= y <= x-a), \/ -> \_/
  void shift(const T &a, const T &b) {
    assert(a <= b);
    addl += a, addr += b;
  }
  // f(x) = f(x-a), \/. -> .\/
  void shift(const T &a) { shift(a, a); }
  // f(x) = f(x) + g(x)
  void merge(SlopeTrick &st) {
    if (st.size() > size()) {
      swap(st.pql, pql), swap(st.pqr, pqr);
      swap(st.addl, addl), swap(st.addr, addr), swap(st.minf, minf);
    }
    while (st.pqr.size()) add_xa(st.popR());
    while (st.pql.size()) add_ax(st.popL());
    minf += st.minf;
  }
};
```

1.49 StronglyConnectedComponent.cpp

```
struct StronglyConnectedComponent {
  int n, k;
  vector<vector<int>> G, reverseG;
  vector<int> vs, cmp;
  vector<bool> used;
  StronglyConnectedComponent(int newv = 1) {
    n = newv;
    G.resize(n);
    reverseG.resize(n);
    used.resize(n, 0);
    cmp.resize(n);
  }

  bool add(int from, int to) {
    G[from].push_back(to);
    reverseG[to].push_back(from);
    return 1;
  }
  void dfs(int v) {
    used[v] = true;
    for (auto to : G[v])
      if (!used[to]) dfs(to);
    vs.push_back(v);
  }
  void rdfs(int v) {
    used[v] = true;
    cmp[v] = k;
    for (auto to : reverseG[v])
      if (!used[to]) rdfs(to);
  }
  int solve() {
    used.assign(n, 0);
    vs.clear();
    for (int v = 0; v < n; ++v)
      if (!used[v]) dfs(v);
    used.assign(n, 0);
    k = 0;
    for (int i = (int)vs.size() - 1; i >= 0; --i)
      if (!used[vs[i]]) {
        rdfs(vs[i]);
        ++k;
      }
    return k;
  }
  vector<vector<int>> make_graph(bool dosolve = 1) {
    if (dosolve) solve();
    vector<vector<int>> res(k);
    for (int i = 0; i < n; ++i)
      for (int to : G[i])
        if (!issame(i, to)) res[cmp[i]].push_back(cmp[to]);
    for (auto& v : res) {
      sort(v.begin(), v.end());
      v.erase(unique(v.begin(), v.end()), v.end());
    }
    return res;
  }
  vector<vector<int>> make_list(bool dosolve = 1) {
    if (dosolve) solve();
    vector<vector<int>> res(k);
    for (int i = 0; i < n; ++i) res[cmp[i]].push_back(i);
    return res;
  }
  bool issame(int l, int r) { return cmp[l] == cmp[r]; }
};
```

1.50 SuffixArray.cpp

```
template <class T>
struct SuffixArray {
  T s;
  int ssize, nowlen;
  vector<int> rank, tmp, sa, lcp;
  SuffixArray() {}
  SuffixArray(T news, bool reqlcp = 0) {
    s = news;
    ssize = s.size();
    nowlen = 0;
    rank.resize(ssize + 1);
    tmp.resize(ssize + 1);
    sa.resize(ssize + 1);
    constuct_sa();
    if (reqlcp) constuct_lcp();
  }

  void constuct_sa() {
    auto cmp = [&](int l, int r) {
      if (rank[l] != rank[r]) return rank[l] < rank[r];
      int rx = l + nowlen <= ssize ? rank[l + nowlen] : -1;
      int ly = r + nowlen <= ssize ? rank[r + nowlen] : -1;
      return rx < ly;
    };

    for (int i = 0; i <= ssize; ++i) {
      sa[i] = i;
      rank[i] = i < ssize ? s[i] : -1;
    }
    for (nowlen = 1; nowlen <= ssize; nowlen *= 2) {
      sort(sa.begin(), sa.end(), cmp);
      tmp[sa[0]] = 0;
      for (int i = 1; i <= ssize; ++i)
        tmp[sa[i]] = tmp[sa[i - 1]] + (cmp(sa[i - 1], sa[i]) ? 1 :
        ↪ 0);
      for (int i = 0; i <= ssize; ++i) rank[i] = tmp[i];
    }
  }
  // Longest Common Prefix Array
  void constuct_lcp() {
    lcp.resize(ssize + 1);
    for (int i = 0; i <= ssize; ++i) rank[sa[i]] = i;
    int h = 0;
    lcp[0] = 0;
    for (int i = 0; i < ssize; ++i) {
      int j = sa[rank[i] - 1];
      if (h > 0) --h;
      for (; j + h < ssize && i + h < ssize; ++h)
```

```
      if (s[j + h] != s[i + h]) break;
    lcp[rank[i] - 1] = h;
  }
 }

 bool contain(T& nowt) {
  int lef = 0, righ = strlen, tsize = nowt.size();
  while (righ - lef > 1) {
   int mid = (lef + righ) / 2;
   if (s.compare(sa[mid], tsize, nowt) < 0)
    lef = mid;
   else
    righ = mid;
  }
  return s.compare(sa[righ], tsize, nowt) == 0;
 }
};
```

1.51 Sum_2d.cpp

```
// 0-indexed
template <class T>
struct Sum_2d {
 int h, w;
 vector<vector<T>> v;
 Sum_2d() {}
 Sum_2d(const vector<vector<T>> &_v) : v(_v) {
  h = v.size(), w = h ? v[0].size() : 0;
  for (int i = 0; i < h; ++i)
   for (int j = 0; j < w; ++j) {
    if (i) v[i][j] += v[i - 1][j];
    if (j) v[i][j] += v[i][j - 1];
    if (i && j) v[i][j] -= v[i - 1][j - 1];
   }
 }
 // sum v[i][j] (u <= i <= d, l <= j <= r)
 T calc(int u, int d, int l, int r) {
  T res = v[d][r];
  if (l) res -= v[d][l - 1];
  if (u) res -= v[u - 1][r];
  if (l && u) res += v[u - 1][l - 1];
  return res;
 }
};
```

1.52 TwoEdgeConnectedComponents.cpp

```
// TwoEdgeConnectedComponents + LowLink
struct TwoEdgeConnectedComponents {
 int n;
 vector<vector<int>> g;
 vector<int> ord, low, articulation, id;
 vector<bool> used;
 using P = pair<int, int>;
 vector<P> bridge;
 TwoEdgeConnectedComponents(int _n = 1) : n(_n), g(n) {}
 TwoEdgeConnectedComponents(vector<vector<int>> &_g) :
 ↪ n(_g.size()), g(_g) {
  lowlinkbuild();
 }

 bool add(int from, int to) {
  g[from].push_back(to);
  g[to].push_back(from);
  return 1;
 }

 void lowlinkbuild() {
  ord.assign(n, -1);
  low.assign(n, -1);
  int k = 0;
  for (int i = 0; i < n; ++i)
   if (ord[i] < 0) lowlinkdfs(i, -1, k);
  sort(articulation.begin(), articulation.end());
 }
 void lowlinkdfs(int now, int par, int &k) {
  ord[now] = low[now] = k++;
  bool ch = 0;  // articulation
  int cnt = 0;
  for (auto &to : g[now])
   if (ord[to] < 0) {
    ++cnt;
    lowlinkdfs(to, now, k);
    low[now] = min(low[now], low[to]);
    ch |= ~par && low[to] >= ord[now];  // articulation
    if (ord[now] < low[to])
     bridge.emplace_back(min(now, to), max(now, to));  // bridge
   } else if (to != par)
    low[now] = min(low[now], ord[to]);
  ch |= par == -1 && cnt > 1;           // articulation
  if (ch) articulation.push_back(now);  // articulation
 }

 vector<vector<int>> build() {
  id.assign(n, -1);
  int k = 0;
  for (int i = 0; i < n; ++i)
   if (id[i] < 0) dfs(i, -1, k);
  vector<vector<int>> res(k);
  for (auto &e : bridge) {
   int x = id[e.first], y = id[e.second];
   res[x].push_back(y);
   res[y].push_back(x);
  }
  return res;
 }

 void dfs(int now, int par, int &k) {
  if (~par && ord[par] >= low[now])
   id[now] = id[par];
  else
   id[now] = k++;
  for (auto &to : g[now])
   if (id[to] < 0) dfs(to, now, k);
 }

 inline int operator[](const int &k) { return (id.at(k)); }
};
```

1.53 TwoSAT.cpp

```
// use SCC
struct TwoSAT {
 int n;
 StronglyConnectedComponent scc;
 TwoSAT(int _n = 1) : n(_n), scc(2 * _n) {}
 // a v c : !a -> c && !c -> a
 void add(int a, bool b, int c, bool d) {
  scc.add(a + b * n, c + (!d) * n);
  scc.add(c + d * n, a + (!b) * n);
 }
 bool solve() {
  scc.solve();
  for (int i = 0; i < n; ++i)
   if (scc.cmp[i] == scc.cmp[i + n]) return 0;
  return 1;
 }
 vector<bool> make_vec() {
  vector<bool> res;
  if (!solve()) return res;
  res.assign(n, 0);
  for (int i = 0; i < n; ++i) res[i] = scc.cmp[i] > scc.cmp[i + n];
  return res;
 }
};
```

1.54 unionfind.cpp

```
struct Unionfind {
 // tree number
 vector<int> par;
 // constructor
 Unionfind(int n = 1) : par(n, -1) {}
 // search root
 int root(int x) {
  if (par[x] < 0) return x;
  return par[x] = root(par[x]);
 }
 // is same?
 bool issame(int x, int y) { return root(x) == root(y); }

 // add
 // already added, return 0
 bool uni(int x, int y) {
  x = root(x);
  y = root(y);
  if (x == y) return 0;
  if (par[x] > par[y]) swap(x, y);
  par[x] += par[y];
  par[y] = x;
  return 1;
 }
 int size(int x) { return -par[root(x)]; }
};
```

1.55 unionfindp.cpp

```
struct UnionfindP {
 using P = pair<int, int>;
 vector<int> par, last;
 vector<vector<P>> hist;
 UnionfindP(int n = 1) : par(n, -1), last(n, -1), hist(n) {
  for (auto& v : hist) v.emplace_back(-1, -1);
 }
 int root(int t, int x) {
  if (last[x] == -1 || t < last[x]) return x;
  return root(t, par[x]);
```

```
  }
  bool issame(int t, int x, int y) { return root(t, x) == root(t,
  ↪  y); }
  bool uni(int t, int x, int y) {
    x = root(t, x), y = root(t, y);
    if (x == y) return 0;
    if (par[x] > par[y]) swap(x, y);
    par[x] += par[y];
    par[y] = x;
    last[y] = t;
    hist[x].emplace_back(t, par[x]);
    return 1;
  }
  int size(int t, int x) {
    x = root(t, x);
    return -prev(lower_bound(hist[x].begin(), hist[x].end(), P(t,
    ↪  0)))->second;
  }
};
```

1.56 unionfindw.cpp

```
template <class T>
struct UnionfindW {
  // tree number
  vector<int> par;
  // tree rank
  vector<int> treerank;
  // edge weight
  vector<T> weight;
  const int inf = 2147483647;
  // constructor
  UnionfindW(int n = 1, T initialnum = 0) { stree(n + 1,
  ↪  initialnum); }
  // make and initialization
  void stree(int n = 1, T initialnum = 0) {
    par.assign(n, -1);
    treerank.assign(n, 0);
    weight.assign(n, initialnum);
  }
  // search root
  int root(int x) {
    if (par[x] < 0) return x;
    int rx = root(par[x]);
    weight[x] += weight[par[x]];
    return par[x] = rx;
  }
  // is same?
  bool issame(int x, int y) { return root(x) == root(y); }
  // calculate weight
  T calcw(int x) {
    root(x);
    return weight[x];
  }
  // add
  // x+w = y
  // already added, return 0
  bool uni(int x, int y, T w = 0) {
    w += calcw(x);
    w -= calcw(y);
    x = root(x);
    y = root(y);
    if (x == y) return 0;
    if (treerank[x] > treerank[y]) swap(x, y);
    if (treerank[x] == treerank[y]) ++treerank[x];
    par[y] -= size(x);
    par[x] = y;
    weight[x] = -w;
    return 1;
  }

  int size(int x) { return -par[root(x)]; }

  // calculate difference between x and y (y-x)
  // if not same tree, return 1
  T calcdiff(int x, int y) {
    if (!issame(x, y)) return inf;
    return calcw(y) - calcw(x);
  }
};
```

1.57 WaveletMatrix.cpp

```
// 0-indexed
struct SuccinctIndexableDictionary {
  using ui = unsigned int;
  int len, blen;
  vector<ui> bit, sum;
  SuccinctIndexableDictionary(int _len = 0)
      : len(_len), blen((_len + 63) >> 5) {
    bit.assign(blen, 0), sum.assign(blen, 0);
  }
  SuccinctIndexableDictionary(const vector<int> &v) {
    len = v.size(), blen = ((len + 63) >> 5);
    bit.assign(blen, 0), sum.assign(blen, 0);
    for (int i = 0; i < len; ++i)
      if (v[i]) set(i);
    build();
  }
  // v[k] = 1
  void set(int k) { bit[k >> 5] |= 1U << (k & 31); }
  inline int popcount(ui x) { return __builtin_popcount(x); }
  void build() {
    sum[0] = 0U;
    for (int i = 1; i < blen; ++i) sum[i] = sum[i - 1] +
    ↪  popcount(bit[i - 1]);
  }
  int rank(int k) {
    return (sum[k >> 5] + popcount(bit[k >> 5] & ((1U << (k & 31)) -
    ↪  1)));
  }
  // count b in [0,k)
  int rank(int b, int k) { return (b ? rank(k) : k - rank(k)); }
  // search k-th b (0-indexed)
  int select(int b, int k) {
    if (rank(b, len - 1) < k) return -1;
    int l = -1, r = len - 1;
    while (r - l > 1) {
      int mid = (l + r) >> 1;
      if (rank(b, mid) >= k)
        r = mid;
      else
        l = mid;
    }
    return l;
  }
  // v[k]
  bool operator[](int k) { return ((bit[k >> 5] >> (k & 31)) & 1); }
};

template <class T, int high = 31>
struct WaveletMatrix {
  int len;
  vector<SuccinctIndexableDictionary> mat;
  vector<int> mid;
  WaveletMatrix() {}
  WaveletMatrix(vector<T> v) : len(v.size()) {
    vector<T> lv(len), rv(len);
    mat.assign(high, SuccinctIndexableDictionary(len));
    mid.resize(high);
    for (int t = high - 1; t >= 0; --t) {
      int l = 0, r = 0;
      for (int i = 0; i < len; ++i)
        if (v[i] >> t & 1)
          mat[t].set(i), rv[r++] = v[i];
        else
          lv[l++] = v[i];
      mid[t] = l, mat[t].build(), v.swap(lv);
      for (int i = 0; i < r; ++i) v[l + i] = rv[i];
    }
  }
  // v[k]
  T access(int k) {
    T res = 0;
    for (int t = high - 1; t >= 0; --t) {
      bool b = mat[t][k];
      if (b) res |= T(1) << t;
      k = mat[t].rank(b, k) + mid[t] * b;
    }
    return res;
  }
  T operator[](const int &k) { return access(k); }

  pair<int, int> succ(bool b, int l, int r, int t) {
    return pair<int, int>(mat[t].rank(b, l) + mid[t] * b,
                          mat[t].rank(b, r) + mid[t] * b);
  }

  // count x in [0,r)
  int rank(const T &x, int r) {
    int l = 0;
    for (int t = high - 1; t >= 0; --t) tie(l, r) = succ(x >> t & 1,
    ↪  l, r, t);
    return r - l;
  }
  //   WIP  search k-th x (0-indexed)
  int select(const T &x, int k) { return -1; }
  // k-th(0-indexed) smallest number in [l,r)
  T kth_smallest(int l, int r, int k) {
    assert(0 <= k && k < r - l);
    T res = 0;
    for (int t = high - 1; t >= 0; --t) {
      int cnt = mat[t].rank(0, r) - mat[t].rank(0, l);
      bool ch = cnt <= k;
      if (ch) res |= T(1) << t, k -= cnt;
      tie(l, r) = succ(ch, l, r, t);
    }
```

```
    return res;
  }
  // k-th(0-indexed) largest number in [l,r)
  T kth_largest(int l, int r, int k) {
    return kth_smallest(l, r, r - l - k - 1);
  }
  // count x < u in [l,r)
  int range_freq(int l, int r, int u) {
    int res = 0;
    for (int t = high - 1; t >= 0; --t) {
      bool b = u >> t & 1;
      if (b) res += mat[t].rank(0, r) - mat[t].rank(0, l);
      tie(l, r) = succ(b, l, r, t);
    }
    return res;
  }
  // count d <= x < u in [l,r)
  int range_freq(int l, int r, int d, int u) {
    return range_freq(l, r, u) - range_freq(l, r, d);
  }
  // max y < x in[l,r)
  T prev_value(int l, int r, T x) {
    int cnt = range_freq(l, r, x);
    return !cnt ? T(-1) : kth_smallest(l, r, cnt - 1);
  }
  // min y >= x in [l,r)
  T next_value(int l, int r, T x) {
    int cnt = range_freq(l, r, x);
    return cnt == r - l ? T(-1) : kth_smallest(l, r, cnt);
  }
};
```

1.58 xorshift.cpp

```
unsigned int xorshift() {
  static unsigned int nowx = 123456789, nowy = 362436069, nowz =
  ↪  521288629,
                      noww = 88675123;
  unsigned int temp;
  temp = (nowx ^ (nowx << 11));
  nowx = nowy;
  nowy = nowz;
  nowz = noww;
  return (noww = (noww ^ (noww >> 19)) ^ (temp ^ (temp >> 8)));
}
```

1.59 z_algorithm.cpp

```
template <class T>
vector<int> z_algorithm(const T &s) {
  vector<int> res;
  int i = 1, j = 0, len = s.size();
  res.resize(s.size());
  res[0] = len;
  while (i < len) {
    while (i + j < len && s[j] == s[i + j]) ++j;
    res[i] = j;
    if (j == 0) {
      ++i;
      continue;
    }
    int k = 1;
    while (i + k < len && k + res[k] < j) res[i + k] = res[k], ++k;
    i += k, j -= k;
  }
  return res;
}
```