Contents

1. Thi cử

1.1. Checklists

1. Wrong answer:
   - [ ] Clear data structure sau mỗi test case chưa ?
   - [ ] Thuật có đúng trong giới hạn input không ?
   - [ ] Đọc lại đề
   - [ ] Xét trường hợp biên chưa ?
   - [ ] Hiểu đúng đề chưa ?
   - [ ] Có biến nào chưa khởi tạo không ?
   - [ ] Tràn số ?
   - [ ] Nhầm biến (N với M, i với j) ?
   - [ ] Có chắc thuật đúng không ?
   - [ ] Có case nào không ngờ đến không ?
   - [ ] Nếu dùng STL, các hàm STL có hoạt động như ý muốn không ?
   - [ ] Debug bằng assert.
   - [ ] Trao đổi với teammate / 2 người cùng code.
   - [ ] Output format đúng chưa ?
   - [ ] Đọc lại checklist.
2. Runtime error:
   - [ ] Test trường hợp biên chưa ?
   - [ ] Biến chưa khởi tạo ?
   - [ ] Tràn mảng ?
   - [ ] Fail assert nào đó ?
   - [ ] Chia/mod cho 0 ?
   - [ ] Đệ quy vô hạn ?
   - [ ] Con trỏ hoặc iterator ?
   - [ ] Dùng quá nhiều bộ nhớ ?
   - [ ] Spam sub để debug (e.g. remapped signals, see Various).
3. Time limit exceeded:
   - [ ] Lặp vô hạn ?
   - [ ] Độ phức tạp có đúng không ?
   - [ ] Tối ưu mod ?
   - [ ] Copy biến quá nhiều ?
   - [ ] Thay vector, map thành array, unordered_map ? Thay int thành short ?
4. Memory limit exceeded:

- [ ] Tối đa cần bao nhiêu bộ nhớ ?
- [ ] Clear data structure sau mỗi test case chưa ?

1.2. commands

```sh
alias c='g++ -g --std=c++17 -O2 -Wall -Wconversion -Wfatal-errors -D_GLIBCXX_DEBUG -fsanitize=address -fsanitize=undefined'
```

1.3. Advices

- Nếu không sure, hãy thảo luận. Nếu kẹt, giải thích đề bài với teammate.
- Viết pseudocode trước khi code, điều này có thể tiết kiệm computer time. Không cần viết hết, mà chỉ cần những phần quan trong nhất.
- Đừng debug code trên máy. In code và debug output rồi debug trên giấy.
- Nếu kẹt, hãy đi dạo hoặc đi vệ sinh. Có thể nghĩ ra gì đó đấy.
- Nếu bị WA liên tục, để tạm đấy và xem bài khác rồi quay lại sau. Đừng ngại viết lại hết code, thường chỉ mất khoảng 15 phút thôi.
- Nếu có thể dễ sinh ra input lớn hoặc tricky test, hãy cố làm điều đó trước khi nộp.
- Làm xong bài nào thì ném mọi thứ liên quan đến nó xuống đất (đề bài, giấy nháp, ...).
- Xem bảng điểm liên tục. Nếu nhiều người giải được, nghĩa là bài đó dễ.
- Ghi lại xem ai đang làm bài nào.
- Cuối giờ, mọi người tập trung vào 1 bài thôi.

2. Toán

2.1. MillerRabin

Description: Kiểm tra số nguyên tố nhanh, **chắc chắn** đúng trong `unsigned long long`.

```h
inline uint64_t mod_mult64(uint64_t a, uint64_t b, uint64_t m) {
  return __int128_t(a) * b % m;
}
uint64_t mod_pow64(uint64_t a, uint64_t b, uint64_t m) {
  uint64_t ret = (m > 1);
  for (;;) {
    if (b & 1) ret = mod_mult64(ret, a, m);
    if (!(b >>= 1)) return ret;
    a = mod_mult64(a, a, m);
  }
}

bool is_prime(uint64_t n) {
  if (n <= 3) return (n >= 2);
```

```
  static const uint64_t small[] = {
      2,   3,   5,   7,   11,  13,  17,  19,  23,  29,  31,  37,
      41,  43,  47,  53,  59,  61,  67,  71,  73,  79,  83,  89,
      97,  101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151,
      157, 163, 167, 173, 179, 181, 191, 193, 197, 199,
  };
  for (size_t i = 0; i < sizeof(small) / sizeof(uint64_t); ++i) {
    if (n % small[i] == 0) return n == small[i];
  }
  static const uint64_t millerrabin[] = {
      2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
  };
  static const uint64_t A014233[] = {
      // From OEIS.
      2047LL,
      1373653LL,
      25326001LL,
      3215031751LL,
      2152302898747LL,
      3474749660383LL,
      341550071728321LL,
      341550071728321LL,
      3825123056546413051LL,
      3825123056546413051LL,
      3825123056546413051LL,
      0,
  };
  uint64_t s = n - 1, r = 0;
  while (s % 2 == 0) {
    s /= 2;
    r++;
  }
  for (size_t i = 0, j; i < sizeof(millerrabin) / sizeof(uint64_t); i++) {
    uint64_t md = mod_pow64(millerrabin[i], s, n);
    if (md != 1) {
      for (j = 1; j < r; j++) {
        if (md == n - 1) break;
        md = mod_mult64(md, md, n);
      }
      if (md != n - 1) return false;
    }
    if (n < A014233[i]) return true;
  }
  return true;
}
// }}}
```

2.2. ModLog

Description: Tìm $x > 0$ nhỏ nhất sao cho $a^x = b \bmod m$, hoặc -1. `modLog(a,1,m)` trả về order của a trong $\mathbb{Z}_m^*$. Độ phức tạp $O(\sqrt{m})$.

```
ll modLog(ll a, ll b, ll m) {
  ll n = (ll)sqrt(m) + 1, e = 1, f = 1, j = 1;
  unordered_map<ll, ll> A;
  while (j <= n && (e = f = e * a % m) != b % m) A[e * b % m] = j++;
  if (e == b % m) return j;
  if (gcd(m, e) == gcd(m, b))
    rep(i, 2, n + 2) if (A.count(e = e * f % m)) return n * i - A[e];
  return -1;
}
```

2.3. ModSQRT

Description: Tìm căn bậc hai modulo p trong trung bình $O(\log p)$.

```
ll modsqrt(ll a, ll p) {
  a %= p;
  if (a < 0) a += p;
  if (a == 0) return 0;

  if (modpow(a, (p - 1) / 2, p) != 1) return -1;
  if (p % 4 == 3) return modpow(a, (p + 1) / 4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0) ++r, s /= 2;
```

```
  /// find a non-square mod p
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m) t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

2.4. Factor

Description: Tìm một ước của n nhanh trong $O(\sqrt[4]{n}\log n)$

```
ull pollard(ull n) {
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  auto f = [&](ull x) { return modmul(x, x, n) + i; };
  while (t++ % 40 || __gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x, y) - min(x, y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

2.5. CRT

Description: Duy trì các phương trình đồng dư và nghiệm thoả mãn.

```
template <typename T>
struct CRT {
  T res;
  CRT() { res = 0, prd = 1; }
  // Add condition: res % p == r
  void add(T p, T r) {
    res += mul(r - res % p + p, euclid(prd, p).first + p, p) * prd;
    prd *= p;
    if (res >= prd) res -= prd;
  }

 private:
  T prd;
  T mul(T a, T b, T p) {
    a %= p, b %= p;
    T q = (T)((long double)a * b / p);
    T r = a * b - q * p;
    while (r < 0) r += p;
    while (r >= p) r -= p;
    return r;
  }
  pair<T, T> euclid(T a, T b) {
    if (!b) return make_pair(1, 0);
    pair<T, T> r = euclid(b, a % b);
    return make_pair(r.second, r.first - a / b * r.second);
  }
};
```

2.6. DivModSum

Description: Tính $\sum_{i=0}^{n-1} \frac{a+i\times d}{m}$ and $\sum_{i=0}^{n-1} (a+i\times d) \bmod m$

```
ll sumsq(ll to) { return to / 2 * ((to - 1) | 1); }
/// ^ written in a weird way to deal with overflows correctly
```

```
// sum( (a + d*i) / m ) for i in [0, n-1]
ll divsum(ll a, ll d, ll m, ll n) {
  ll res = d / m * sumsq(n) + a / m * n;
  d %= m, a %= m;
  if (!d) return res;
  ll to = (n * d + a) / m;
  return res + (n - 1) * to - divsum(m - 1 - a, m, d, to);
}
// sum( (a + d*i) % m ) for i in [0, n-1]
ll modsum(ll a, ll d, ll m, ll n) {
  a = ((a % m) + m) % m, d = ((d % m) + m) % m;
  return n * a + d * sumsq(n) - m * divsum(a, d, m, n);
}
```

2.7. FFT

Description: FFT trên ℝ

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vector<complex<long double>> R(2, 1);
  static vector<C> rt(2, 1);  // (^ 10% faster if double)
  for (static int k = 2; k < n; k *= 2) {
    R.resize(n);
    rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    rep(i, k, 2 * k) rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
  }
  vi rev(n);
  rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
        auto x = (double*)&rt[j + k],
             y = (double*)&a[i + j + k];
        C z(x[0] * y[0] - x[1] * y[1],
             x[0] * y[1] + x[1] * y[0]);
        a[i + j + k] = a[i + j] - z;
        a[i + j] += z;
      }
}
vd conv(const vd& a, const vd& b) {
  if (a.empty() || b.empty()) return {};
  vd res(sz(a) + sz(b) - 1);
  int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
  vector<C> in(n), out(n);
  copy(all(a), begin(in));
  rep(i, 0, sz(b)) in[i].imag(b[i]);
  fft(in);
  for (C& x : in) x *= x;
  rep(i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
  fft(out);
  rep(i, 0, sz(res)) res[i] = imag(out[i]) / (4 * n);
  return res;
}
```

2.8. NTT

Description: FFT trên trường hữu hạn với modulo nguyên tố **bất kỳ**.

```
#include "FFT.h"

typedef vector<ll> vl;
template <int M>
vl convMod(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  vl res(sz(a) + sz(b) - 1);
  int B = 32 - __builtin_clz(sz(res)), n = 1 << B, cut = int(sqrt(M));
  vector<C> L(n), R(n), outs(n), outl(n);
  rep(i, 0, sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
  rep(i, 0, sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
  fft(L), fft(R);
  rep(i, 0, n) {
    int j = -i & (n - 1);
```

```
    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
  }
  fft(outl), fft(outs);
  rep(i, 0, sz(res)) {
    ll av = ll(real(outl[i]) + .5), cv = ll(imag(outs[i]) + .5);
    ll bv = ll(imag(outl[i]) + .5) + ll(real(outs[i]) + .5);
    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
  }
  return res;
}
```

2.9. FST

Description: Tính tích chập AND, OR, XOR.

```
void FST(vi& a, bool inv) {
  for (int n = sz(a), step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step) rep(j, i, i + step) {
        int &u = a[j], &v = a[j + step];
        tie(u, v) = inv ? pii(v - u, u) : pii(v, u + v);  // AND
        // inv ? pii(v, u - v) : pii(u + v, u); // OR /// include-line
        // pii(u + v, u - v);                   // XOR /// include-line
      }
  }
  // if (inv) for (int& x : a) x /= sz(a); // XOR only /// include-line
}
vi conv(vi a, vi b) {
  FST(a, 0);
  FST(b, 0);
  rep(i, 0, sz(a)) a[i] *= b[i];
  FST(a, 1);
  return a;
}
```

2.10. BerlekampMassey

Description: Phục hồi một dãy truy hồi cấp n từ $2n$ số hạng đầu tiên trong $O(n^2)$.

```
vector<ll> berlekampMassey(vector<ll> s) {
  int n = sz(s), L = 0, m = 0;
  vector<ll> C(n), B(n), T;
  C[0] = B[0] = 1;

  ll b = 1;
  rep(i, 0, n) {
    ++m;
    ll d = s[i] % mod;
    rep(j, 1, L + 1) d = (d + C[j] * s[i - j]) % mod;
    if (!d) continue;
    T = C;
    ll coef = d * modpow(b, mod - 2) % mod;
    rep(j, m, n) C[j] = (C[j] - coef * B[j - m]) % mod;
    if (2 * L > i) continue;
    L = i + 1 - L;
    B = T;
    b = d;
    m = 0;
  }

  C.resize(L + 1);
  C.erase(C.begin());
  for (ll& x : C) x = (mod - x) % mod;
  return C;
}
```

3. Hình

3.1. Point

```
template <class T>
int sgn(T x) {
  return (x > 0) - (x < 0);
}
template <class T>
struct Point {
  typedef Point P;
```

```
  T x, y;
  explicit Point(T x = 0, T y = 0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x, y) < tie(p.x, p.y); }
  bool operator==(P p) const { return tie(x, y) == tie(p.x, p.y); }
  P operator+(P p) const { return P(x + p.x, y + p.y); }
  P operator-(P p) const { return P(x - p.x, y - p.y); }
  P operator*(T d) const { return P(x * d, y * d); }
  P operator/(T d) const { return P(x / d, y / d); }
  T dot(P p) const { return x * p.x + y * p.y; }
  T cross(P p) const { return x * p.y - y * p.x; }
  T cross(P a, P b) const { return (a - *this).cross(b - *this); }
  T dist2() const { return x * x + y * y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this / dist(); }  // makes dist()=1
  P perp() const { return P(-y, x); }        // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
  }
  friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << "," << p.y << ")";
  }
};
```

3.2. ConvexHull

```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts) + 1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(pts)))
    for (P p : pts) {
      while (t >= s + 2 && h[t - 2].cross(h[t - 1], p) <= 0) t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

4. Cấu trúc dữ liệu

4.1. DSURollback

```
struct DSURollback {
  vi e;
  vector<pii> st;
  DSURollback(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)]; }
  int find(int x) { return e[x] < 0 ? x : find(e[x]); }
  int time() { return sz(st); }
  void rollback(int t) {
    for (int i = time(); i-- > t;) e[st[i].first] = st[i].second;
    st.resize(t);
  }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]});
    st.push_back({b, e[b]});
    e[a] += e[b];
    e[b] = a;
    return true;
  }
};
```

4.2. PersistentIT

```
struct Node {
  int left, right;  // ID of left child & right child
  long long ln;     // Max value of node
```

```
  Node() {}
  Node(long long ln, int left, int right) : ln(ln), left(left), right(right)
{}
} it[11000111];  // Each node has a position in this array, called ID
int nNode;

int ver[MN];  // ID of root in each version

// Update max value of a node
inline void refine(int cur) {
  it[cur].ln = max(it[it[cur].left].ln, it[it[cur].right].ln);
}

// Update a range, and return new ID of node
int update(int l, int r, int u, int x, int oldId) {
  if (l == r) {
    ++nNode;
    it[nNode] = Node(x, 0, 0);
    return nNode;
  }

  int mid = (l + r) >> 1;
  int cur = ++nNode;

  if (u <= mid) {
    it[cur].left = update(l, mid, u, x, it[oldId].left);
    it[cur].right = it[oldId].right;
    refine(cur);
  } else {
    it[cur].left = it[oldId].left;
    it[cur].right = update(mid + 1, r, u, x, it[oldId].right);
    refine(cur);
  }

  return cur;
}

// Get max of range. Same as usual IT
int get(int nodeId, int l, int r, int u, int v) {
  if (v < l || r < u) return -1;
  if (u <= l && r <= v) return it[nodeId].ln;

  int mid = (l + r) >> 1;
  return max(get(it[nodeId].left, l, mid, u, v),
             get(it[nodeId].right, mid + 1, r, u, v));
}

// When update:
++nVer;
ver[nVer] = update(1, n, u, x, ver[nVer - 1]);

// When query:
res = get(ver[t], 1, n, u, v);
```

4.3. Splay

```
using splay_key = int;
struct splay_node {
  splay_node *parent = nullptr, *child[2] = {nullptr, nullptr};
  splay_key key;
  int size = 1;
  static int get_size(splay_node *x) { return x == nullptr ? 0 : x->size; }
  int parent_index() const {
    if (parent == nullptr) return -1;
    return this == parent->child[0] ? 0 : 1;
  }
  void set_child(int index, splay_node *x) {
    child[index] = x;
    if (x != nullptr) x->parent = this;
  }
  int sum = 0;
  int get_sum(splay_node *x) { return x == nullptr ? 0 : x->sum; }
  void join() {
    sum = get_sum(child[0]) + get_sum(child[1]) + key;
    size = get_size(child[0]) + get_size(child[1]) + 1;
  }
```

```
  int query() { return get_sum(child[0]); }
};
struct splay_tree {
  static const int POOL_SIZE = 1 << 12;
  static vector<splay_node *> node_pool;
  static splay_node *new_node(const splay_key &key) {
    if (node_pool.empty()) {
      splay_node *ptr = new splay_node[POOL_SIZE];
      for (int i = POOL_SIZE - 1; i >= 0; i--) node_pool.push_back(ptr + i);
    }
    splay_node *node = node_pool.back();
    node_pool.pop_back();
    node->key = key;
    node->join();
    return node;
  }
  ~splay_tree() {}
  splay_node *root = nullptr;
  bool empty() const { return root == nullptr; }
  int size() const { return root == nullptr ? 0 : root->size; }
  splay_node *set_root(splay_node *x) {
    if (x != nullptr) x->parent = nullptr;
    return root = x;
  }
  void rotate_up(splay_node *x, bool x_join = true) {
    splay_node *p = x->parent, *gp = p->parent;
    int index = x->parent_index();
    if (gp == nullptr)
      set_root(x);
    else
      gp->set_child(p->parent_index(), x);
    p->set_child(index, x->child[!index]);
    x->set_child(!index, p);
    p->join();
    if (x_join) x->join();
  }
  void splay(splay_node *x) {
    while (x != root) {
      splay_node *p = x->parent;
      if (p != root)
        rotate_up(x->parent_index() == p->parent_index() ? p : x, false);
      rotate_up(x, false);
    }
    x->join();
  }
  void check_splay(splay_node *x, int depth) {
    assert(x != nullptr);
    int n = size(), log_n = 32 - __builtin_clz(n);
    // Splay when deep or with a certain random chance when small.
    if (depth > 2 * log_n) splay(x);
  }
  pair<splay_node *, int> insert(const splay_key &key,
                                 bool require_unique = false) {
    return insert(new_node(key), require_unique);
  }
  // Returns {new node pointer, index (number of existing elements that are
  // strictly less)}
  pair<splay_node *, int> insert(splay_node *x, bool require_unique = false)
{
    if (root == nullptr) return {set_root(x), 0};
    splay_node *current = root, *prev = nullptr;
    int below = 0, depth = 0;
    while (current != nullptr) {
      prev = current;
      depth++;
      if (current->key < x->key) {
        below += splay_node::get_size(current->child[0]) + 1;
        current = current->child[1];
      } else {
        if (require_unique && !(x->key < current->key)) {
          below += splay_node::get_size(current->child[0]);
          check_splay(current, depth);
          return {current, below};
        }
        current = current->child[0];
      }
```

```
    }
    prev->set_child(prev->key < x->key ? 1 : 0, x);
    check_splay(x, depth);
    for (splay_node *node = x; node != nullptr; node = node->parent)
      node->join();
    return {x, below};
  }
  splay_node *begin() {
    if (root == nullptr) return nullptr;
    splay_node *x = root;
    int depth = 0;
    while (x->child[0] != nullptr) {
      x = x->child[0];
      depth++;
    }
    check_splay(x, depth);
    return x;
  }
  // To iterate through all nodes in order:
  // for (splay_node *node = tree.begin(); node != nullptr; node =
  // tree._next(node))
  splay_node *_next(splay_node *x) const {
    if (x == nullptr) return nullptr;
    if (x->child[1] != nullptr) {
      x = x->child[1];
      while (x->child[0] != nullptr) x = x->child[0];
      return x;
    }
    while (x->parent_index() == 1) x = x->parent;
    return x->parent;
  }
  splay_node *_prev(splay_node *x) const {
    if (x == nullptr) return nullptr;
    if (x->child[0] != nullptr) {
      x = x->child[0];
      while (x->child[1] != nullptr) x = x->child[1];
      return x;
    }
    while (x->parent_index() == 0) x = x->parent;
    return x->parent;
  }
  splay_node *last() {
    if (root == nullptr) return nullptr;
    splay_node *x = root;
    int depth = 0;
    while (x->child[1] != nullptr) {
      x = x->child[1];
      depth++;
    }
    check_splay(x, depth);
    return x;
  }
  void clear() {
    vector<splay_node *> nodes;
    nodes.reserve(size());
    for (splay_node *node = begin(); node != nullptr; node = _next(node))
      nodes.push_back(node);
    for (splay_node *node : nodes) {
      *node = splay_node();
      node_pool.push_back(node);
    }
    set_root(nullptr);
  }
  void erase(splay_node *x) {
    splay_node *new_x = nullptr, *fix_node = nullptr;
    if (x->child[0] == nullptr || x->child[1] == nullptr) {
      new_x = x->child[x->child[0] == nullptr ? 1 : 0];
      fix_node = x->parent;
    } else {
      splay_node *next = _next(x);
      assert(next != nullptr && next->child[0] == nullptr);
      new_x = next;
      fix_node = next->parent == x ? next : next->parent;
      next->parent->set_child(next->parent_index(), next->child[1]);
      next->set_child(0, x->child[0]);
      next->set_child(1, x->child[1]);
```

```
    }
    if (x == root)
      set_root(new_x);
    else
      x->parent->set_child(x->parent_index(), new_x);
    int depth = 0;
    for (splay_node *node = fix_node; node != nullptr; node = node->parent)
{
      node->join();
      depth++;
    }
    if (fix_node != nullptr) check_splay(fix_node, depth);
    *x = splay_node();
    node_pool.push_back(x);
  }
  // Returns {node pointer, index (number of existing elements that are
strictly
  // less)}
  pair<splay_node *, int> lower_bound(const splay_key &key) {
    splay_node *current = root, *prev = nullptr, *answer = nullptr;
    int below = 0, depth = 0;
    while (current != nullptr) {
      prev = current;
      depth++;
      if (current->key < key) {
        below += splay_node::get_size(current->child[0]) + 1;
        current = current->child[1];
      } else {
        answer = current;
        current = current->child[0];
      }
    }
    if (prev != nullptr) check_splay(prev, depth);
    return make_pair(answer, below);
  }
  bool contains(const splay_key &key) {
    splay_node *node = lower_bound(key).first;
    return node != nullptr && node->key == key;
  }
  bool erase(const splay_key &key) {
    splay_node *x = lower_bound(key).first;
    if (x == nullptr || x->key != key) return false;
    erase(x);
    return true;
  }
  splay_node *node_at_index(int index) {
    if (index < 0 || index >= size()) return nullptr;
    splay_node *current = root;
    int depth = 0;
    while (current != nullptr) {
      int left_size = splay_node::get_size(current->child[0]);
      depth++;
      if (index == left_size) {
        check_splay(current, depth);
        return current;
      }
      if (index < left_size) {
        current = current->child[0];
      } else {
        current = current->child[1];
        index -= left_size + 1;
      }
    }
    assert(false);
  }
};
vector<splay_node *> splay_tree::node_pool;
splay_tree s;
int prefix_sum(int k) {
  // returns sum of elements < k
  auto node = s.insert(k).first;
  s.splay(node);
  int res = node->query();
  s.erase(node);
  return res;
}
```

```
// keywords: insert, erase, lower_bound, node_at_index, contains
```

5. Đồ thị

5.1. 2SAT

```
struct TwoSat {
  int N;
  vector<vi> gr;
  vi values;  // 0 = false, 1 = true

  TwoSat(int n = 0) : N(n), gr(2 * n) {}

  int addVar() {  // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }

  void either(int f, int j) {
    f = max(2 * f, -1 - 2 * f);
    j = max(2 * j, -1 - 2 * j);
    gr[f].push_back(j ^ 1);
    gr[j].push_back(f ^ 1);
  }
  void setValue(int x) { either(x, x); }

  void atMostOne(const vi& li) {  // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i, 2, sz(li)) {
      int next = addVar();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }

  vi val, comp, z;
  int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x;
    z.push_back(i);
    for (int e : gr[i])
      if (!comp[e]) low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
        x = z.back();
        z.pop_back();
        comp[x] = low;
        if (values[x >> 1] == -1) values[x >> 1] = x & 1;
      } while (x != i);
    return val[i] = low;
  }

  bool solve() {
    values.assign(N, -1);
    val.assign(2 * N, 0);
    comp = val;
    rep(i, 0, 2 * N) if (!comp[i]) dfs(i);
    rep(i, 0, N) if (comp[2 * i] == comp[2 * i + 1]) return 0;
    return 1;
  }
};
```

5.2. HopcroftKarp

Description: Cặp ghép cực đại trên đồ thị 2 phía trong $O\left(E\sqrt{V}\right)$.

Usage: `vi btoa(m, -1); hopcroftKarp(g, btoa);`

```
bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
  if (A[a] != L) return 0;
  A[a] = -1;
  for (int b : g[a])
    if (B[b] == L + 1) {
```

```
      B[b] = 0;
      if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
        return btoa[b] = a, 1;
    }
  return 0;
}

int hopcroftKarp(vector<vi>& g, vi& btoa) {
  int res = 0;
  vi A(g.size()), B(btoa.size()), cur, next;
  for (;;) {
    fill(all(A), 0);
    fill(all(B), 0);
    /// Find the starting nodes for BFS (i.e. layer 0).
    cur.clear();
    for (int a : btoa)
      if (a != -1) A[a] = -1;
    rep(a, 0, sz(g)) if (A[a] == 0) cur.push_back(a);
    /// Find all layers using bfs.
    for (int lay = 1;; lay++) {
      bool islast = 0;
      next.clear();
      for (int a : cur)
        for (int b : g[a]) {
          if (btoa[b] == -1) {
            B[b] = lay;
            islast = 1;
          } else if (btoa[b] != a && !B[b]) {
            B[b] = lay;
            next.push_back(btoa[b]);
          }
        }
      if (islast) break;
      if (next.empty()) return res;
      for (int a : next) A[a] = lay;
      cur.swap(next);
    }
    /// Use DFS to scan for augmenting paths.
    rep(a, 0, sz(g)) res += dfs(a, 0, g, btoa, A, B);
  }
}
```

5.3. GeneralMatching

Description: Thuật toán Blossom tìm cặp ghép cực đại trên đồ thị thường trong $O(V^3)$. Đánh chỉ số từ 0.

```
struct GeneralMatching {
  int n;
  vector<int> match;
  GeneralMatching(int n): n(n), match(n, -1), g(n), timer(-1), label(n),
parent(n), orig(n), aux(n, -1) {}

  void add_edge(int u, int v) {
    g[u].push_back(v), g[v].push_back(u);
  }

  int get_match() {
    for (int i = 0; i < n; i++) if (match[i] == -1) bfs(i);
    int res = 0;
    for (int i = 0; i < n; i++) if (match[i] >= 0) ++res;
    return res / 2;
  }

 private:
  int lca(int x, int y) {
    for (timer++;; swap(x, y)) {
      if (x == -1) continue;
      if (aux[x] == timer) return x;
      aux[x] = timer;
      x = (match[x] == -1 ? -1 : orig[parent[match[x]]]);
    }
  }

  void blossom(int v, int w, int a) {
    while (orig[v] != a) {
```

```
        parent[v] = w;
        w = match[v];
        if (label[w] == 1) {
          label[w] = 0;
          q.push_back(w);
        }
        orig[v] = orig[w] = a;
        v = parent[w];
      }
    }

    void augment(int v) {
      while (v != -1) {
        int pv = parent[v], nv = match[pv];
        match[v] = pv;
        match[pv] = v;
        v = nv;
      }
    }

    int bfs(int root) {
      fill(label.begin(), label.end(), -1);
      iota(orig.begin(), orig.end(), 0);
      q.clear();
      label[root] = 0;
      q.push_back(root);
      for (int i = 0; i < (int)q.size(); ++i) {
        int v = q[i];
        for (auto x : g[v]) {
          if (label[x] == -1) {
            label[x] = 1;
            parent[x] = v;
            if (match[x] == -1) {
              augment(x);
              return 1;
            }
            label[match[x]] = 0;
            q.push_back(match[x]);
          } else if (label[x] == 0 && orig[v] != orig[x]) {
            int a = lca(orig[v], orig[x]);
            blossom(x, v, a), blossom(v, x, a);
          }
        }
      }
      return 0;
    }

  private:
    vector<vector<int>> g;
    int timer;
    vector<int> label, parent, orig, aux, q;
};
```

5.4. PushRelabel

Description: Thuận toán Push-relabel trong $O\left(V^2\sqrt{E}\right)$.

```
struct PushRelabel {
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g;
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs;
  vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2 * n), H(n) {}

  void addEdge(int s, int t, ll cap, ll rcap = 0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s]) - 1, 0, rcap});
  }

  void addFlow(Edge& e, ll f) {
```

```cpp
    Edge& back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
    e.f += f;
    e.c -= f;
    ec[e.dest] += f;
    back.f -= f;
    back.c += f;
    ec[back.dest] -= f;
  }
  ll calc(int s, int t) {
    int v = sz(g);
    H[s] = v;
    ec[t] = 1;
    vi co(2 * v);
    co[0] = v - 1;
    rep(i, 0, v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);

    for (int hi = 0;;) {
      while (hs[hi].empty())
        if (!hi--) return -ec[s];
      int u = hs[hi].back();
      hs[hi].pop_back();
      while (ec[u] > 0)  // discharge u
        if (cur[u] == g[u].data() + sz(g[u])) {
          H[u] = 1e9;
          for (Edge& e : g[u])
            if (e.c && H[u] > H[e.dest] + 1) H[u] = H[e.dest] + 1, cur[u] =
&e;
          if (++co[H[u]], !--co[hi] && hi < v)
            rep(i, 0, v) if (hi < H[i] && H[i] < v)-- co[H[i]], H[i] = v +
1;
          hi = H[u];
        } else if (cur[u]->c && H[u] == H[cur[u]->dest] + 1)
          addFlow(*cur[u], min(ec[u], cur[u]->c));
        else
          ++cur[u];
    }
  }
  bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

5.5. Hungarian

```cpp
template <typename T>
pair<T, vector<int>> Hungarian(int n, int m, T c[][N]) {
  vector<T> v(m), dist(m);
  vector<int> L(n, -1), R(m, -1);
  vector<int> index(m), prev(m);
  auto getc = [&](int i, int j) { return c[i][j] - v[j]; };

  iota(index.begin(), index.end(), 0);
  for (int f = 0; f < n; ++f) {
    for (int j = 0; j < m; ++j) {
      dist[j] = getc(f, j), prev[j] = f;
    }
    T w = 0;
    int j, l = 0, s = 0, t = 0;
    while (true) {
      if (s == t) {
        l = s, w = dist[index[t++]];
        for (int k = t; k < m; ++k) {
          j = index[k];
          T h = dist[j];
          if (h <= w) {
            if (h < w) t = s, w = h;
            index[k] = index[t], index[t++] = j;
          }
        }
        for (int k = s; k < t; ++k) {
          j = index[k];
          if (R[j] < 0) goto augment;
        }
      }
      int q = index[s++], i = R[q];
      for (int k = t; k < m; ++k) {
```

```
        j = index[k];
        T h = getc(i, j) - getc(i, q) + w;
        if (h < dist[j]) {
          dist[j] = h, prev[j] = i;
          if (h == w) {
            if (R[j] < 0) goto augment;
            index[k] = index[t], index[t++] = j;
          }
        }
      }
    }
  augment:
    for (int k = 0; k < l; ++k) v[index[k]] += dist[index[k]] - w;
    int i;
    do {
      i = R[j] = prev[j];
      swap(j, L[i]);
    } while (i != f);
  }
  T ret = 0;
  for (int i = 0; i < n; ++i) ret += c[i][L[i]];
  return {ret, L};
}
```

5.6. GomoryHu

Description: Tính maxflow của từng cặp đỉnh trong $N-1$ lần chạy luồng.

```
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
  vector<Edge> tree;
  vi par(N);
  rep(i, 1, N) {
    PushRelabel D(N);  // Dinic also works
    for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
    tree.push_back({i, par[i], D.calc(i, par[i])});
    rep(j, i + 1, N) if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
  }
  return tree;
}
```

6. Xâu

6.1. MinRotation

Tìm cyclic shift của xâu có thứ tự từ điển nhỏ nhất trong $O(n)$.

```
int minRotation(string s) {
  int a = 0, N = sz(s);
  s += s;
  rep(b, 0, N) rep(k, 0, N) {
    if (a + k == b || s[a + k] < s[b + k]) {
      b += max(0, k - 1);
      break;
    }
    if (s[a + k] > s[b + k]) {
      a = b;
      break;
    }
  }
  return a;
}
```

6.2. SuffixArray

```
struct SuffixArray {
  vi sa, lcp;
  SuffixArray(string& s, int lim = 256) {  // or basic_string<int>
    int n = sz(s) + 1, k = 0, a, b;
    vi x(all(s)), y(n), ws(max(n, lim));
    x.push_back(0), sa = lcp = y, iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
      p = j, iota(all(y), n - j);
      rep(i, 0, n) if (sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      rep(i, 0, n) ws[x[i]]++;
```

```
      rep(i, 1, lim) ws[i] += ws[i - 1];
      for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
      swap(x, y), p = 1, x[sa[0]] = 0;
      rep(i, 1, n) a = sa[i - 1], b = sa[i],
                   x[b] = (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 :
p++;
    }
    for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
      for (k &&k--, j = sa[x[i] - 1]; s[i + k] == s[j + k]; k++);
  }
};
```

6.3. AhoCorasick

```
struct aho_corasick {
  struct node {
    int suffix_link = -1, exit_link = -1, cnt = 0, nxt[26];
    node() { fill(nxt, nxt + 26, -1); }
  };
  vector<node> g = {node()};
  void insert_string(const string &s) {
    int p = 0;
    for (char c : s) {
      if (g[p].nxt[c - 'a'] == -1) {
        g[p].nxt[c - 'a'] = g.size();
        g.emplace_back();
      }
      p = g[p].nxt[c - 'a'];
    }
    g[p].cnt++;
  }
  void build_automaton() {
    for (deque<int> q = {0}; q.size(); q.pop_front()) {
      int v = q.front(), suffix_link = g[v].suffix_link;
      if (v)
        g[v].exit_link =
            g[suffix_link].cnt ? suffix_link : g[suffix_link].exit_link;
      for (int i = 0; i < 26; i++) {
        int &nxt = g[v].nxt[i], nxt_sf = v ? g[suffix_link].nxt[i] : 0;
        if (nxt == -1)
          nxt = nxt_sf;
        else {
          g[nxt].suffix_link = nxt_sf;
          q.push_back(nxt);
        }
      }
    }
  }
};
```

7. Khác

7.1. pbds

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/rope>

using namespace __gnu_pbds;

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

const int RANDOM =
chrono::high_resolution_clock::now().time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
using fast_map = gp_hash_table<int, int, chash>;
```

7.2. LineContainer

```
struct Line {
  mutable ll k, m, p;
```

```cpp
  bool operator<(const Line& o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) {  // floored division
    return a / b - ((a ^ b) < 0 && a % b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k)
      x->p = x->m > y->m ? inf : -inf;
    else
      x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

7.3. Fraction

Chặt nhị phân tìm phân số dương lớn thứ k với mẫu số không vượt quá n.

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

using namespace std;

#define rep(i, a, b) for (int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
typedef unsigned long long ull;
typedef __int128_t i128;

struct Frac {
  i128 p, q;
};

i128 sumsq(ull to) { return i128(to) / 2 * ((to - 1) | 1); }

i128 divsum(ull to, ull c, ull k, ull m) {
  i128 res = k / m * sumsq(to) + c / m * to;
  k %= m;
  c %= m;
  if (!k) return res;
  i128 to2 = (to * k + c) / m;
  return res + (to - 1) * to2 - divsum(to2, m - 1 - c, m, k);
}

const i128 inf = 1e18 + 1;

i128 count(Frac f, ull n) { return divsum(n + 1, 0, f.p, f.q); }

void solve() {
  ull n, k;
  cin >> n >> k;

  vector<Frac> bound = {{0, 1}, {1, 0}};
  Frac cur = {1, 0};
  bool turn_left = false;
  while (true) {
    int i = sz(bound) - 1;
```

```cpp
    i128 lo = 0, hi = inf;
    while (lo < hi) {
      i128 mid = (lo + hi + 1) >> 1;
      Frac f{bound[i - 1].p + bound[i].p * mid,
             bound[i - 1].q + bound[i].q * mid};
      if (f.q > n) {
        hi = mid - 1;
        continue;
      }
      if (turn_left) {
        if (count(f, n) >= k)
          lo = mid;
        else
          hi = mid - 1;
      } else {
        if (count(f, n) < k)
          lo = mid;
        else
          hi = mid - 1;
      }
    }
    if (turn_left && lo == 0) break;
    Frac f{bound[i - 1].p + lo * bound[i].p, bound[i - 1].q + lo *
bound[i].q};
    bound.emplace_back(f);
    if (count(f, n) >= k) cur = f;
    turn_left = !turn_left;
  }
  i128 cnt = count(cur, n);
  i128 cnt_same = n / cur.q;
  Frac ans = {cur.p * (k - (cnt - cnt_same)), cur.q * (k - (cnt -
cnt_same))};
  cout << uint64_t(ans.p) << ' ' << uint64_t(ans.q) << '\n';
}
```

7.4. 1D1D

Nếu hàm $w(i, j)$ thoả mãn bất đẳng thức tứ giác: $w(a, c) + w(b, d) \leq w(a, d) + w(b, c)$ với mọi $a < b < c < d$, thì ta có thể tính hàm DP 1 chiều: $f(i) = \min_{0 \leq j < i} f(j) + w(j, i)$ trong $O(n \log n)$.

```cpp
struct item {
  int l, r, p;
};

const int N = 1e5 + 3;
int n;
long long f[N];

long long w(int j, int i) {
  // một hàm cost bất kì thỏa mãn
  // bất đẳng thức tứ giác
}

void solve() {
  deque<item> dq;
  dq.push_back({1, n, 0});
  for (int i = 1; i <= n; ++i) {
    f[i] = f[dq.front().p] + w(dq.front().p, i);
    // deque chỉ lưu giá trị từ h[i + 1]
    // tới h[n]
    ++dq.front().l;

    // nếu l > r, ta loại đoạn này khỏi deque
    if (dq.front().l > dq.front().r) {
      dq.pop_front();
    }

    while (!dq.empty()) {
      auto [l, r, p] = dq.back();
      if (f[i] + w(i, l) < f[p] + w(p, l)) {
        dq.pop_back();
        // p không còn là giá trị của
```

```
        // h[l], h[l + 1], ..., h[r]
        // lúc này, h[l]=h[l+1]=...=h[r]=i.
      } else
        break;
    }

    if (dq.empty()) {
      dq.push_back({i + 1, n, i});
      // h[i+1]=h[i+2]=...=h[n]=i
    } else {
      // tìm nhị phân vị trí pos nhỏ nhất
      // thỏa mãn h[pos] = i
      auto& [l, r, p] = dq.back();
      int low = l, high = r;
      int pos = r + 1, mid;
      while (low <= high) {
        mid = (low + high) / 2;
        if (f[i] + w(i, mid) < f[p] + w(p, mid)) {
          pos = mid, high = mid - 1;
        } else {
          low = mid + 1;
        }
      }

      // cập nhật đoạn (l,r,p) thành (l,pos-1,p)
      r = pos - 1;
      if (pos <= n) {
        dq.push_back({pos, n, i});
        // h[pos]=h[pos+1]=...=h[n]=i
      }
    }
  }
}
```

8. Trick & Ghi chú

8.1. Sequences

8.1.1. Catalan

$$C_n = \frac{1}{n+1}\binom{2n}{n}, C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

8.1.2. Lucas

Let $n = n_k p^k + n_{k-1} p^{k-1} + ... + n_0$ and $m = m_k p^k + m_{k-1} p^{k-1} + ... + m_0$ in base p.

$$\binom{n}{m} = \prod_{i=0}^{k} \binom{n_i}{m_i} \bmod p$$

.

8.1.3. Number of Derangements

$d(n)$ là số hoán vị n phần tử mà không có i sao cho $p_i = i$.

$$d(n) = (n-1)(d(n-1) + d(n-2))$$

.

8.1.4. Số Stirling loại 1

Số hoán vị n phần tử có đúng k chu trình.

$$s(n,k) = s(n-1,k-1) + (n-1)s(n-1,k)$$

$$\sum_{k=0}^{n} s(n,k)x^k = x(x+1)...(x+n-1)$$

8.1.5. Số Stirling loại 2

Số cách chia n phần tử vào đúng k nhóm.

$$S(n,k) = kS(n-1,k) + S(n-1,k-1)$$

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k} (-1)^{k-j}\binom{k}{j} j^n$$

8.2. Bổ đề Burnside

Đặt G là nhóm hữu hạn tác động lên tập X. Với mỗi $g \in G$, gọi X^g là tập các điểm bất định bởi g ($\{x \in X \mid g.x = x\}$). Số quỹ đạo có thể có là:

$$\left|\frac{X}{G}\right| = \frac{1}{|G|}\sum_{g\in G}|X^g|$$

8.3. Super interpretation of kth powers

The square of the size of a set is equal to the number of ordered pairs of elements in the set. So we iterate over pairs and for each we compute the contribution to the answer.

Similarly, the k-th power is equal to the number of sequences (tuples) of length k.

$$E(X^2) = E(\#\text{ordered pairs}), E\left(X^k\right) = E(\#\text{ordered tuples})$$

8.4. Power technique

If you want to maintain the sum of k-th powers, it might help to also maintain the sum of smaller powers. For example, if the sum of 0-th, 1-th and 2-nd powers is S_0, S_1 and S_2, and we increase all elements by x, the new sums are S_0, $S_1 + S_0x$ and $S_2 + 2xS_1 + x^2S_0$.