Contents

1. Contest

1.1. commands

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

2. Mathematics

2.1. MillerRabin

Description: Very fast Rabin Miller. Guarantee to work with `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: Returns the smallest $x > 0$ s.t. $a^x = b \bmod m$, or -1 otherwise. `modLog(a,1,m)` outputs the order of a.

Time: $O(\sqrt{m})$.

```
ll modLog(ll a, ll b, ll m) {                                                    h
  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: Finds x s.t. $x^2 = a \bmod p$ ($-x$ gives the other solution).

Time: $O(\log^2 p)$ worst case, $O(\log p)$ for most p.

```
ll modsqrt(ll a, ll p) {                                                         h
  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

```
ull pollard(ull n) {                                                    h
  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

```
template <typename T>                                                   h
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

```
ll sumsq(ll to) { return to / 2 * ((to - 1) | 1); }                     h
/// ^ 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. LinearRec

```
typedef vector<ll> Poly;                                                    h
ll linearRec(Poly S, Poly tr, ll k) {
  int n = sz(tr);

  auto combine = [&](Poly a, Poly b) {
    Poly res(n * 2 + 1);
    rep(i, 0, n + 1) rep(j, 0, n + 1) res[i + j] =
        (res[i + j] + a[i] * b[j]) % mod;
    for (int i = 2 * n; i > n; --i)
      rep(j, 0, n) res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
    res.resize(n + 1);
    return res;
  };

  Poly pol(n + 1), e(pol);
  pol[0] = e[1] = 1;

  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
  }

  ll res = 0;
  rep(i, 0, n) res = (res + pol[i + 1] * S[i]) % mod;
  return res;
}
```

3. Geometry

3.1. Point

```
template <class T>                                                          h
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;                                                        h
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. Data Structures

4.1. RMQ

h

```
template <class T>
struct RMQ {
  vector<vector<T>> jmp;
  RMQ(const vector<T>& V) : jmp(1, V) {
    for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k) {
      jmp.emplace_back(sz(V) - pw * 2 + 1);
      rep(j, 0, sz(jmp[k])) jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j +
pw]);
    }
  }
  T query(int a, int b) {
    assert(a < b);  // or return inf if a == b
    int dep = 31 - __builtin_clz(b - a);
    return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
  }
};
```

4.2. DSURollback

h

```
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;
  }
};
```

5. Graph

5.1. Biconnected

h

```
vi num, st;
vector<vector<pii>> ed;
int Time;
template <class F>
int dfs(int at, int par, F& f) {
  int me = num[at] = ++Time, top = me;
  for (auto [y, e] : ed[at])
    if (e != par) {
      if (num[y]) {
        top = min(top, num[y]);
        if (num[y] < me) st.push_back(e);
      } else {
        int si = sz(st);
        int up = dfs(y, e, f);
        top = min(top, up);
```

```
        if (up == me) {
          st.push_back(e);
          f(vi(st.begin() + si, st.end()));
          st.resize(si);
        } else if (up < me)
          st.push_back(e);
        else { /* e is a bridge */
        }
      }
    }
  return top;
}

template <class F>
void bicomps(F f) {
  num.assign(sz(ed), 0);
  rep(i, 0, sz(ed)) if (!num[i]) dfs(i, -1, f);
}
```

5.2. 2SAT

```
struct TwoSat {                                                    h
  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;
  }
};
```

5.3. HopcroftKarp

Description: Fast bipartite matching in $O\left(E\sqrt{V}\right)$.

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);
  }
}
```

5.4. GeneralMatching

Description: Blossom algorithm in $O(V^3)$. Index from 0.

```
struct GeneralMatching {
  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;
  }

  int n;
  vector<int> match;

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;
};
```

5.5. PushRelabel

Description: Push-relabel max flow algorithm.

Time: $O\left(V^2\sqrt{E}\right)$.

```
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) {
    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); }
};
```

5.6. GomoryHu

```
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. Strings

6.1. MinRotation

```
int minRotation(string s) {                                                    h
  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 {                                                           h
  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 {                                                          h
  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. Misc

7.1. pbds

```
#include <ext/pb_ds/assoc_container.hpp>                                h
#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

```
#pragma once                                                            h

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

8. Tricks & Notes

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. Stirling 1st

Number of permutations of n elements with k cycles.

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

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

8.1.4. Stirling 2nd

Number of ways to partition n elements into exactly k groups.

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

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

8.2. Burnside's Lemma

Given a group G of symmetries and a set X, the number of elements of X **up to symmetry** $|\frac{X}{G}|$ equals:

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

Where X^g is the set of elements in X fixed by g.

8.3. Super interpretation of square

Ví dụ, cho một cây có N đỉnh. Trong mỗi cách tách cây thành các phần, mỗi cách có giá trị là tổng bình phương độ lớn các thành phần liên thông. Tính tổng giá trị tất cả các cách tách.

Cách giải $O(n^2)$: Có một song ánh giữa đáp án của bài toán, với tổng các