Initial Setup and Definitions

Definitions

```
typedef long long ll;
typedef vector< int > vi;
typedef vector< vi > vvi;
typedef pair< int, int > pii;
typedef vector< pii > vpii;
typedef vector< vpii > vvpii;
```

Fast Input

```
ios_base::sync_with_stdio(false);
cin.tie(NULL);
cout.setf(ios::fixed);
cout.precision(4);
```

Mathematics

```
#define gcd(a, b) __gcd(a, b)
#define lcm(a, b) gcd(a, b) ? (a / gcd(a, b) * b) : 0
const double PI = 3.14159265358979323846264338327950;
const ll PRIME_BASE = (1 << 61) - 1;
```

Strings

Rolling Hashing

```
class RollingHashing {
  ll p, m, ns;
  vector< ll > pows, hash;

  RollingHashing(string s) {
    // if WA then other p and other m
    // if still WA then double hashing
    // if still WA maybe is not the answer RH
    p = 31; m = 1e9 + 7;

    ns = s.size();
    pows.resize(ns + 2);
    for(int i = 1; i < ns + 2; i++)
      pows[i] = (pows[i - 1] * p) % m;

    hash.resize(ns + 1);
    hash[0] = 0;
    for(int i = 1; i <= ns; i++) {
      ll char_to_num = S[i - 1] - 'a' + 1;
      ll prev_hash = hash[i - 1];
      hash[i] = ((char_to_num * pows[i - 1]) % m + prev_hash)
     % m;
    }
  }

  ll compute_hashing(ll i, ll j) {
    return (hash[j] - hash[i] + m) % m;
  }
}
```

Algorithms

Binary Pow

```
ll binpow(ll a, ll b, ll mod) {
  a %= m;
  ll res = 1;
  while (b > 0) {
    if (b & 1)
      res = (res * a) % mod;
    a = (a * a) % mod;
    b >>= 1;
  }
  return res;
}
```

Segment Tree

```
template <class T>
stuct SegmenTree {
  int N;
  vector<T> ST;
  T (*merge)(T, T);
  void build(int n, int l, int r, vector<T> &vs) {
    if(l == r) ST[n] = vs[l];
    else {
      build(n * 2, l, (r + l) / 2, vs);
      build(n * 2 + 1, (r + l) / 2 + 1, vs);
      ST[n] = merge(ST[n * 2], ST[n * 2 + 1]);
    }
  }
  SegmentTree(vector<T> &vs, T (*m)(T a, T b)) {
    merge = m;
    N = vs.size();
    ST.resize(4 * N + 3);
    build(1, 0, N - 1, vs);
  }
  T query(int i, int j) {
    return query(0, N - 1, 1, i, j);
  }
  T query(int l, int r, int n, int i, int j) {
    if(l >= i && r <= j) return ST[n];
    int mid = (r + l) / 2;
    if(mid < i) return query(mid + 1, r, n*2+1, i, j);
    if(mid >= j) return query(l, mid, n*2, i, j);
    return merge(query(l, mid, n * 2, i, j),
                 query(mid + 1, r, n * 2 + 1, i, j));
  }
  void update(int pos, T val) {
    update(0, N - 1, 1, pos, val);
  }
  void update(int l, int r, int n, int pos, T val) {
    if(r < pos || pos < l) return;
    if(l == r) ST[n] = val;
    else {
      int mid = (r + l) / 2;
      update(l, mid, n * 2, pos, val);
      update(mid + 1, r, n * 2 + 1, pos, val);
      ST[n] = merge(ST[n * 2], ST[n * 2 + 1]);
    }
  }
};
```

Segment Tree Lazy

```
template <class T>
struct SegmentTree {
  int N; vector<T> ST, lazy;
  vector<bool> bit; T (*merge)(T, T);
  void build(int n, int l, int r, vector<T> &vs) {
    if(l == r) ST[n] = vs[l];
    else {
      build(2 * n, l, (r + l) / 2, vs);
      build(2 * n + 1, (r + l) / 2 + 1, r, vs);
      ST[n] = merge(ST[n * 2], ST[n * 2 + 1]);
    }
  }
  SegmentTree(vector<T> &vs, T (*m)(T a, T b)) {
    merge = m; N = vs.size();
    ST.resize(4 * N + 3); lazy.assign(4 * N + 3, T());
    bit.assign(4 * N + 3, false); build(1, 0, N - 1, vs);
  }
  void push(int n, int i, int j) {
    if(bit[n]) {
      ST[n] += lazy[n];
      if(i != j) {
        lazy[2 * n] += lazy[n];
        lazy[2 * n + 1] += lazy[n];
        bit[2 * n] = 1; bit[2 * n + 1] = 1;
      }
      lazy[n] = T(); bit[n] = 0;
    }
  }
  void apply(int n, int i, int j, T val) {
    ST[n] += val;
    if(i != j) {
      lazy[2 * n] += val;
      lazy[2 * n + 1] += val;
      bit[2 * n] = 1; bit[2 * n + 1] = 1;
    }
  }
  T query(int i, int j) {
    return query(0, N - 1, 1, i, j); }
  T query(int l, int r, int n, int i, int j) {
    push(n, l, r);
    if(i <= l && r <= j) return ST[n];
    int mid = (r + l) / 2;
    if(mid<i||j<l) return query(mid + 1, r, 2 * n + 1, i, j);
    if(mid >= j || r < i) return query(l, mid, 2 * n, i, j);
    return merge(query(l, mid, 2 * n, i, j),
                 query(mid + 1, r, 2 * n + 1, i, j));
  }
  void update(int i, int j, T val) {
    update(0, N - 1, 1, i, j, val); }
  void update(int l,int r,int n,int i,int j, T val) {
    push(n, l, r);
    if(r < i || j < l) return;
    if(i <= l && r <= j) {
      apply(n, l, r, val); return;
    }
    int mid = (r + l) / 2;
    update(l, mid, 2 * n, i, j, val);
    update(mid + 1, r, 2 * n + 1, i, j, val);
    ST[n] = merge(ST[2 * n], ST[2 * n + 1]);
  }};
```

Union Find

```
struct UnionFind {
  vector<int> p, r;
  UnionFind(int n) {
    r.assign(n+1, 0);
    p.assign(n+1, 0);
    for(int i=1; i<=n; i++) p[i] = i;
  }
  int findSet(int i) {
    return (p[i] == i)? i:(p[i] = findSet(p[i]));
  }
  bool isSameSet(int i, int j) {
    return findSet(i) == findSet(j);
  }
  void unionSet(int i, int j) {
    if (!isSameSet(i, j)) {
      int x = findSet(i), y = findSet(j);
      if (r[x] > r[y]) p[y] = x;
      else {
        p[x] = y;
        if (r[x] == r[y]) r[y]++;
      }
    }
  }
};
```

Maths

Fraction

```
struct Fraction {
  ll numerator, denominator;
  Fraction(ll a, ll b){
    numerator = a, denominator = b;
  }
  Fraction simplify(Fraction f){
    ll g = gcd(f.numerator, f.denominator);
    return Fraction(f.numerator/g, f.denominator/g);
  }
  Fraction add(Fraction f){
    ll l = lcm(denominator, f.denominator);
    numerator *= (l/denominator);
    numerator += f.numerator * (l/f.denominator);
    return simplify(Fraction(numerator, l));
  }
};
```