Team notebook

September 5, 2017

Contents

1 Data Structure

1.1 BIT

```
using vi = vector < int >;
using vii = vector < vi >;

struct BIT_2D {
    int n;
    vii tree;

    BIT_2D () {}
    BIT_2D ( int _n ) : n( _n ), tree( _n, vi( _n, 0 ) ) {}
    ~BIT_2D () {}
    void update_y( int x, int y, int v ) {
        for( ; y<n; y+=(y&-y) ) {
            tree[x][y] += v;
        }
    }

    void update( int x, int y, int v ) {
        for( ; x<n; x+=(x&-x) ) {
            update_y( x, y, v );
        }
    }

    int query_y( int x, int y ) {
        int ret = 0;
        for( ; y; y-=(y&-y) ) {
            ret += tree[x][y];
        }
        return ret;
    }

    int query( int x, int y ) {
        int ret = 0;
        for( ; x; x-=(x&-x) ) {
            ret += query_y( x, y );
        }
        return ret;
    }

    int query( int x1, int y1, int x2, int y2 ) {
        return ( query( x2, y2 ) - query( x2, y1-1 ) - query( x1-1, y2 ) +
            query( x1-1, y1-1 ) );
    }
}

struct BIT {
    int n;
    vi tree;

    BIT () {}
    BIT ( int _n ) : n( _n ), tree( _n, 0 ) {}
    ~BIT () {}
    void update( int x, int v ) {
        for( ; x<n; y+=(x&-x) ) {
            tree[x] += v;
        }
    }

    int query( int x ) {
        int ret = 0;
        for( ; x; x-=(x&-x) ) {
            ret += tree[x];
        }
        return ret;
    }

    int query( int x, int y, int x2, int y2 ) {
        return ( query( y ) - query( x-1 ) );
    }
}
```

1.2 BigInt

```
using ll = long long;
const int base = 1000000000;
const int base_digits = 9;

struct bigint {
    vector<int> z;
    int sign;

    bigint() :
        sign(1) {
    }

    bigint(long long v) {
        *this = v;
    }

    bigint(const string &s) {
        read(s);
    }

    void operator=(const bigint &v) {
        sign = v.sign;
        z = v.z;
    }

    void operator=(long long v) {
        sign = 1;
        if (v < 0)
            sign = -1, v = -v;
        z.clear();
        for (; v > 0; v = v / base)
            z.push_back(v % base);
    }

    bigint operator+(const bigint &v) const {
        if (sign == v.sign) {
            bigint res = v;

            for (int i = 0, carry = 0; i < (int) max(z.size(), v.z.size())
                || carry; ++i) {
                if (i == (int) res.z.size())
                    res.z.push_back(0);
            res.z[i] += carry + (i < (int) z.size() ? z[i] : 0);
            carry = res.z[i] >= base;
            if (carry)
                res.z[i] -= base;
        }
        return res;
    }
    return *this - (-v);
}

bigint operator-(const bigint &v) const {
    if (sign == v.sign) {
        if (abs() >= v.abs()) {
            bigint res = *this;
            for (int i = 0, carry = 0; i < (int) v.z.size() || carry; ++i) {
                res.z[i] -= carry + (i < (int) v.z.size() ? v.z[i] : 0);
                carry = res.z[i] < 0;
                if (carry)
                    res.z[i] += base;
            }
            res.trim();
            return res;
        }
        return -(v - *this);
    }
    return *this + (-v);
}

void operator*=(int v) {
    if (v < 0)
        sign = -sign, v = -v;
    for (int i = 0, carry = 0; i < (int) z.size() || carry; ++i) {
        if (i == (int) z.size())
            z.push_back(0);
        long long cur = z[i] * (long long) v + carry;
        carry = (int) (cur / base);
        z[i] = (int) (cur % base);
        //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur), "c"(base));
    }
    trim();
}

bigint operator*(int v) const {
```

```
    bigint res = *this;
    res *= v;
    return res;
}

friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &b1) {
    int norm = base / (b1.z.back() + 1);
    bigint a = a1.abs() * norm;
    bigint b = b1.abs() * norm;
    bigint q, r;
    q.z.resize(a.z.size());

    for (int i = a.z.size() - 1; i >= 0; i--) {
        r *= base;
        r += a.z[i];
        int s1 = b.z.size() < r.z.size() ? r.z[b.z.size()] : 0;
        int s2 = b.z.size() - 1 < r.z.size() ? r.z[b.z.size() - 1] : 0;
        int d = ((long long) s1 * base + s2) / b.z.back();
        r -= b * d;
        while (r < 0)
            r += b, --d;
        q.z[i] = d;
    }

    q.sign = a1.sign * b1.sign;
    r.sign = a1.sign;
    q.trim();
    r.trim();
    return make_pair(q, r / norm);
}

friend bigint sqrt(const bigint &a1) {
    bigint a = a1;
    while (a.z.empty() || a.z.size() % 2 == 1)
        a.z.push_back(0);

    int n = a.z.size();

    int firstDigit = (int) sqrt((double) a.z[n - 1] * base + a.z[n - 2]);
    int norm = base / (firstDigit + 1);
    a *= norm;
    a *= norm;
    while (a.z.empty() || a.z.size() % 2 == 1)
```

```
        a.z.push_back(0);

    bigint r = (long long) a.z[n - 1] * base + a.z[n - 2];
    firstDigit = (int) sqrt((double) a.z[n - 1] * base + a.z[n - 2]);
    int q = firstDigit;
    bigint res;

    for(int j = n / 2 - 1; j >= 0; j--) {
        for(; ; --q) {
            bigint r1 = (r - (res * 2 * base + q) * q) * base * base +
                 (j > 0 ? (long long) a.z[2 * j - 1] * base + a.z[2 * j
                 - 2] : 0);
            if (r1 >= 0) {
                r = r1;
                break;
            }
        }
        res *= base;
        res += q;

        if (j > 0) {
            int d1 = res.z.size() + 2 < r.z.size() ? r.z[res.z.size()
                 + 2] : 0;
            int d2 = res.z.size() + 1 < r.z.size() ? r.z[res.z.size()
                 + 1] : 0;
            int d3 = res.z.size() < r.z.size() ? r.z[res.z.size()] : 0;
            q = ((long long) d1 * base * base + (long long) d2 * base
                 + d3) / (firstDigit * 2);
        }
    }

    res.trim();
    return res / norm;
}

bigint operator/(const bigint &v) const {
    return divmod(*this, v).first;
}

bigint operator%(const bigint &v) const {
    return divmod(*this, v).second;
}

void operator/=(int v) {
    if (v < 0)
```

```
        sign = -sign, v = -v;
    for (int i = (int) z.size() - 1, rem = 0; i >= 0; --i) {
        long long cur = z[i] + rem * (long long) base;
        z[i] = (int) (cur / v);
        rem = (int) (cur % v);
    }
    trim();
}

bigint operator/(int v) const {
    bigint res = *this;
    res /= v;
    return res;
}

int operator%(int v) const {
    if (v < 0)
        v = -v;
    int m = 0;
    for (int i = z.size() - 1; i >= 0; --i)
        m = (z[i] + m * (long long) base) % v;
    return m * sign;
}

void operator+=(const bigint &v) {
    *this = *this + v;
}
void operator-=(const bigint &v) {
    *this = *this - v;
}
void operator*=(const bigint &v) {
    *this = *this * v;
}
void operator/=(const bigint &v) {
    *this = *this / v;
}

bool operator<(const bigint &v) const {
    if (sign != v.sign)
        return sign < v.sign;
    if (z.size() != v.z.size())
        return z.size() * sign < v.z.size() * v.sign;
    for (int i = z.size() - 1; i >= 0; i--)
        if (z[i] != v.z[i])
            return z[i] * sign < v.z[i] * sign;
    return false;
}

bool operator>(const bigint &v) const {
    return v < *this;
}
bool operator<=(const bigint &v) const {
    return !(v < *this);
}
bool operator>=(const bigint &v) const {
    return !(*this < v);
}
bool operator==(const bigint &v) const {
    return !(*this < v) && !(v < *this);
}
bool operator!=(const bigint &v) const {
    return *this < v || v < *this;
}

void trim() {
    while (!z.empty() && z.back() == 0)
        z.pop_back();
    if (z.empty())
        sign = 1;
}

bool isZero() const {
    return z.empty() || (z.size() == 1 && !z[0]);
}

bigint operator-() const {
    bigint res = *this;
    res.sign = -sign;
    return res;
}

bigint abs() const {
    bigint res = *this;
    res.sign *= res.sign;
    return res;
}

long long longValue() const {
    long long res = 0;
    for (int i = z.size() - 1; i >= 0; i--)
        res = res * base + z[i];
    return res * sign;
}

friend bigint gcd(const bigint &a, const bigint &b) {
    return b.isZero() ? a : gcd(b, a % b);
}
friend bigint lcm(const bigint &a, const bigint &b) {
    return a / gcd(a, b) * b;
}

void read(const string &s) {
    sign = 1;
    z.clear();
    int pos = 0;
    while (pos < (int) s.size() && (s[pos] == ’-’ || s[pos] == ’+’)) {
        if (s[pos] == ’-’)
            sign = -sign;
        ++pos;
    }
    for (int i = s.size() - 1; i >= pos; i -= base_digits) {
        int x = 0;
        for (int j = max(pos, i - base_digits + 1); j <= i; j++)
            x = x * 10 + s[j] - ’0’;
        z.push_back(x);
    }
    trim();
}

friend istream& operator>>(istream &stream, bigint &v) {
    string s;
    stream >> s;
    v.read(s);
    return stream;
}

friend ostream& operator<<(ostream &stream, const bigint &v) {
    if (v.sign == -1)
        stream << ’-’;
    stream << (v.z.empty() ? 0 : v.z.back());
    for (int i = (int) v.z.size() - 2; i >= 0; --i)
        stream << setw(base_digits) << setfill(’0’) << v.z[i];
    return stream;
}
```

```
static vector<int> convert_base(const vector<int> &a, int old_digits,
     int new_digits) {
    vector<long long> p(max(old_digits, new_digits) + 1);
    p[0] = 1;
    for (int i = 1; i < (int) p.size(); i++)
        p[i] = p[i - 1] * 10;
    vector<int> res;
    long long cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int) a.size(); i++) {
        cur += a[i] * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >= new_digits) {
            res.push_back(int(cur % p[new_digits]));
            cur /= p[new_digits];
            cur_digits -= new_digits;
        }
    }
    res.push_back((int) cur);
    while (!res.empty() && res.back() == 0)
        res.pop_back();
    return res;
}

typedef vector<long long> vll;

static vll karatsubaMultiply(const vll &a, const vll &b) {
    int n = a.size();
    vll res(n + n);
    if (n <= 32) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                res[i + j] += a[i] * b[j];
        return res;
    }

    int k = n >> 1;
    vll a1(a.begin(), a.begin() + k);
    vll a2(a.begin() + k, a.end());
    vll b1(b.begin(), b.begin() + k);
    vll b2(b.begin() + k, b.end());

    vll a1b1 = karatsubaMultiply(a1, b1);
    vll a2b2 = karatsubaMultiply(a2, b2);
```

```
    for (int i = 0; i < k; i++)
        a2[i] += a1[i];
    for (int i = 0; i < k; i++)
        b2[i] += b1[i];

    vll r = karatsubaMultiply(a2, b2);
    for (int i = 0; i < (int) a1b1.size(); i++)
        r[i] -= a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
        r[i] -= a2b2[i];

    for (int i = 0; i < (int) r.size(); i++)
        res[i + k] += r[i];
    for (int i = 0; i < (int) a1b1.size(); i++)
        res[i] += a1b1[i];
    for (int i = 0; i < (int) a2b2.size(); i++)
        res[i + n] += a2b2[i];
    return res;
}

bigint operator*(const bigint &v) const {
    vector<int> a6 = convert_base(this->z, base_digits, 6);
    vector<int> b6 = convert_base(v.z, base_digits, 6);
    vll a(a6.begin(), a6.end());
    vll b(b6.begin(), b6.end());
    while (a.size() < b.size())
        a.push_back(0);
    while (b.size() < a.size())
        b.push_back(0);
    while (a.size() & (a.size() - 1))
        a.push_back(0), b.push_back(0);
    vll c = karatsubaMultiply(a, b);
    bigint res;
    res.sign = sign * v.sign;
    for (int i = 0, carry = 0; i < (int) c.size(); i++) {
        long long cur = c[i] + carry;
        res.z.push_back((int) (cur % 1000000));
        carry = (int) (cur / 1000000);
    }
    res.z = convert_base(res.z, 6, base_digits);
    res.trim();
    return res;
}
long long sod() {
    long long res = 0;
```

```
        long long ret = 0;
        for (int i = z.size() - 1; i >= 0; i--) {
            res = res * base + z[i];
            while( res ) {
                ret += res % 10;
                res /= 10;
            }
        }
        return ret;
    }
};

bigint random_bigint(int n) {
    string s;
    for (int i = 0; i < n; i++) {
        s += rand() % 10 + ’0’;
    }
    return bigint(s);
}
```

1.3 DisjointSet

```
/**
    Implementation of Disjoint-Set Union Data Structure
    Running time:
        O(nlog(n))
    Usage:
        - call make_set() to reset the set
        - call find_rep() to get the set of the vertex
        - call union_() to merge to sets
    Input:
        - n, number of sets
    Tested Problems:
      UVA:
        10608 - Friends
        11503 - Virtual Friends
        10583 - Ubiquitous Religions
**/

struct Disjoint_Set {
    int n;
    vector < int > par, cnt, rnk;
    Disjoint_Set( int n ) : n(n), rnk(n), par(n), cnt(n) {}

    void make_set() {
        for(int i=0; i<n; i++) {
            par[i] = i;
            cnt[i] = 1;
            rnk[i] = 0;
        }
    }

    int find_rep( int x ) {
        if(x != par[ x ]) {
            par[ x ] = find_rep( par[ x ] );
        }
        return par[ x ];
    }

    int union_( int u, int v ) {
        if( ( u = find_rep( u ) ) != ( v = find_rep( v ) ) ) {
            if( rnk[ u ] < rnk[ v ] ) {
                cnt[ v ] += cnt[ u ];
                par[ u ] = par[ v ];
                return cnt[v];
            } else {
                rnk[ u ] = max( rnk[ u ], rnk[ v ] + 1 );
                cnt[ u ] += cnt[ v ];
                par[ v ] = par[ u ];
            }
        }
        return cnt[u];
    }
} DS( sz );
```

1.4 Fraction

```
struct fraction {
    ll up, down;
    ll gcd, lcm;
    void thik_kor() {
        gcd = __gcd( up, down );
        up /= gcd;
        down /= gcd;
    }
```

```
  fraction operator += ( const fraction &rhs ) {
    gcd = __gcd( down, rhs.down );
    lcm = ( down / gcd ) * rhs.down;
    up = ( ( lcm / down ) * up ) + ( ( lcm / rhs.down ) * rhs.up );
    down = lcm;
    thik_kor();
    return *this;
  }
} ans, inp;
```

1.5 LCA

```
/**
  Implementation of LCA ( Lowest Common Ancestor ) with sparse table
  Running time:
    O( n * log ( n ) )
  Usage:
    - call dfs( 0, 0, 0 )
    - call init()
    - call query() to get output
  Input:
    - Graph
    - n, nodes
  Output:
    - Lowest Common Ancestor
  Tested Problems:
   SPOJ:
    LCA - Lowest Common Ancestor
    QTREE2 - Query on a tree II
**/
#include <bits/stdc++.h>
using namespace std;

const int sz = 1005;
int lg[sz];
int lvl[sz];
int table[sz][12];
int par[sz];

vector < int > G[sz];

void dfs( int fr, int u, int dep ) {
  lvl[u] = dep;
  par[u] = fr;
  for( int v: G[u] ) {
    if( fr == v ) continue;
    dfs( u, v, dep+1 );
  }
}

int init( int n ) {
  memset( table, -1, sizeof table );
  for( int i=0; i<n; i++ ) {
    table[i][0] = par[i];
  }
  for( int j=1; ( 1 << j ) < n; j++ ) {
    for( int i=0; i<n; i++ ) {
      if( table[i][j-1] == -1 ) continue;
      table[i][j] = table[ table[i][j-1] ][j-1];
    }
  }
  for( int i=0; i<10; i++ ) lg[1 << i] = i;
  for( int i=1; i<sz; i++ ) if( !lg[i] ) lg[i] = lg[i-1];
}

int query( int n, int p, int q ) {
  int log;
  if( lvl[p] < lvl[q] ) swap( p, q );
  log = lg[ lvl[ p ] ];
  for( int i=log; i>=0; i-- ) {
    if( lvl[p] - ( 1 << i ) >= lvl[q] ) {
      p = table[ p ][ i ];
    }
  }
  if( p == q ) return p;
  for( int i=log; i>=0; i-- ) {
    if( table[ p ][ i ] != -1 && table[ p ][ i ] != table[ q ][ i ] ) {
      p = table[ p ][ i ];
      q = table[ q ][ i ];
    }
  }
  return par[p];
}

int dist( int n, int p, int q ) {
  int lca = query( n, p, q );
  return lvl[p] + lvl[q] - 2 * lvl[lca];
}
```

1.6 Mo'sAlgo

```cpp
/**
    Implementation of Mo's Algo with SQRT-Decomposition Data Structure
    Running time:
        O( ( n + q ) * sqrt( n ) * f() )
    Mo's Algo is a algorithm to process queries offline
    For it to work, this condition must be satisified:
        1) There can be no updates in the array
        2) All queries must be known beforehand
    Tested Problems:
      CF:
        220B - Little Elephant and Array
**/
#include <bits/stdc++.h>
using namespace std;

using piii = pair < pair < int, int >, int >;
const int mx = 1e5 + 1;
int BLOCK_SIZE;
int n, m;
int calc;
int ar[mx];
int ans[mx];
unordered_map < int, int > cnt;
piii query[mx];

struct {
    bool operator()( const piii &a, const piii &b ) {
        int block_a = a.first.first / BLOCK_SIZE;
        int block_b = b.first.first / BLOCK_SIZE;
        if( block_a != block_b ) {
            return block_a < block_b;
        }
        return a.first.second < b.first.second;
    }
} cmp;

void add( int x ) {
    calc -= ( cnt[x] == x ? 1 : 0 );
    cnt[x]++;
    calc += ( cnt[x] == x ? 1 : 0 );
}

void remove( int x ) {
    calc -= ( cnt[x] == x ? 1 : 0 );
    cnt[x]--;
    calc += ( cnt[x] == x ? 1 : 0 );
}

int main() {
    #ifdef LU_SERIOUS
        freopen( "in.txt", "r", stdin );
//        freopen( "out.txt", "w+", stdout );
    #endif // LU_SERIOUS
    while( ~scanf( "%d %d", &n, &m ) ) {

        BLOCK_SIZE = sqrt( n );
        cnt.clear();
        calc = 0;

        for( int i=0; i<n; i++ ) scanf( "%d", ar+i );

        for( int i=0; i<m; i++ ) {
            scanf( "%d %d", &query[i].first.first, &query[i].first.second
                );
            query[i].second = i;
        }

        sort( query, query+m, cmp );

        int mo_l = 0, mo_r = -1;

        for( int i=0; i<m; i++ ) {
            int left = query[i].first.first - 1;
            int right = query[i].first.second - 1;

            while( mo_r < right ) {
                mo_r++;
                add( ar[mo_r] );
            }

            while( mo_r > right ) {
                remove( ar[mo_r] );
                mo_r--;
            }

            while( mo_l < left ) {
                remove( ar[mo_l] );
                mo_l++;
            }

            while( mo_l > left ) {
                mo_l--;
                add( ar[mo_l] );
            }

            ans[ query[i].second ] = calc;
        }

        for( int i=0; i<m; i++ ) {
            printf( "%d\n", ans[i] );
        }
    }
    return 0;
}
```

1.7 PersistantSegmentTree

```
const int sz = 1e5 + 10;
int ar[sz];

struct node {
    node *left;
    node *right;
    int val;

    node( int val = 0, node *left = nullptr, node *right = nullptr ) {
        this->val = val;
        this->left = left;
        this->right = right;
    }

    void build( int l, int r ) {
        if( l == r ) {
            this->val = ar[l];
            return;
        }
        int mid = ( l + r ) >> 1;
        this->left = new node();
        this->right = new node();
        this->left->build( l, mid );
        this->right->build( mid + 1, r );
        this->val = this->left->val + this->right->val;
    }

    node *update( int l, int r, int idx, int x ) {
        if( r < idx || idx < l ) {
            return this;
        }
        if( l == r ) {
            node *ret = new node( this->val, this->left, this->right );
            ret->val += x;
            return ret;
        }
        int mid = ( l + r ) >> 1;
        node *ret = new node( this->val );
        ret->left = this->left->update( l, mid, idx, x );
        ret->right = this->right->update( mid + 1, r, idx, x );
        ret->val = ret->left->val + ret->right->val;
        return ret;
    }

    int query( int l, int r, int i, int j ) {
        if( r < i || l > j ) {
            return 0;
        }
        if( i <= l && r <= j ) {
            return this->val;
        }
        int mid = ( l + r ) >> 1;
        return this->left->query( l, mid, i, j ) + this->right->query( mid + 1, r, i, j );
    }
} *root[sz];

int main() {
    ar[] = { 1, 2, 3 };
    root[0] = new node();
    root[0]->build( 0, 2 );
    root[1] = root[0]->update( 0, 2, 1, 1 );
    root[1] = root[1]->update( 0, 2, 1, 1 );
    printf( "%d\n", root[0].query( 0, 2, 0, 2 ) );
    printf( "%d\n", root[1].query( 0, 2, 0, 2 ) );
}
```

1.8 SegmentTree

```
struct info {
    int prop, sum;
} tree[ mx * 3 ];

void update( int node, int b, int e, int i, int j, int x ) {
//    cerr << b << " " << e << " " << i << " " << j << " " << x << "\n";
    if( i > e || j < b ) {
        return;
    }
    if( b >= i && e <= j ) {
        tree[node].sum = ( e - b + 1 ) * x;
        tree[node].prop = x;
        return;
    }

    int left = node << 1;
    int right = left | 1;
    int mid = (b + e) >> 1;

    if( tree[node].prop != -1 ) {
        tree[left].sum = ( mid - b + 1 ) * tree[node].prop;
        tree[right].sum = ( e - mid ) * tree[node].prop;
        tree[node].sum = tree[left].sum + tree[right].sum;
        tree[left].prop = tree[node].prop;
        tree[right].prop = tree[node].prop;
        tree[node].prop = -1;
    }

    update(left, b, mid, i, j, x);
    update(right, mid + 1, e, i, j, x);

    tree[node].sum = tree[left].sum + tree[right].sum;
}

int query( int node, int b, int e, int i, int j ) {
    if( i > e || j < b ) {
        return 0;
    }
    if( b >= i and e <= j ) {
        return tree[node].sum;
    }

    int left = node << 1;
    int right = left | 1;
    int mid = (b + e) >> 1;

    if( tree[node].prop != -1 ) {
        tree[left].sum = ( mid - b + 1 ) * tree[node].prop;
        tree[right].sum = ( e - mid ) * tree[node].prop;
        tree[node].sum = tree[left].sum + tree[right].sum;
        tree[left].prop = tree[node].prop;
        tree[right].prop = tree[node].prop;
        tree[node].prop = -1;
    }

    int p1 = query( left, b, mid, i, j );
    int p2 = query( right, mid + 1, e, i, j );


    return p1 + p2;
}
```

1.9 SqrtDecomposition

```
/**
    Implementation of SQRT-Decomposition Data Structure
    Running time:
        O( ( n + q ) * sqrt( n ) * f() )
    Usage:
        - call int() to initialize the array
        - call update() to update the element in a position
        - call query() to get ans from segment [L...R]
    Input:
        - n, number of elements
        - n elements
        - q queries
    Tested Problems:
      lightOJ:
        1082 - Array Queries
**/
#include <bits/stdc++.h>
using namespace std;

const int mx = 1e5 + 1;
const int sz = 1e3 + 1;
const int inf = 1e9;
```

```
int BLOCK_SIZE;
int n, q, t, cs, x, y;
int BLOCKS[sz];
int ar[mx];

int getID( int idx ) {
    return idx / BLOCK_SIZE;
}

void init() {
    for( int i=0; i<sz; i++ ) BLOCKS[i] = inf;
}

void update( int idx, int val ) {
    int id = getID( idx );
    BLOCKS[id] = min( val, BLOCKS[id] );
}

int query( int l, int r ) {
    int le = getID( l );
    int ri = getID( r );
    int ret = inf;

    if( le == ri ) {
        for( int i=l; i<=r; i++ ) {
            ret = min( ret, ar[i] );
        }
        return ret;
    }

    for( int i=l; i<(le+1)*BLOCK_SIZE; i++ ) ret = min( ret, ar[i] );
    for( int i=le+1; i<ri; i++ ) ret = min( ret, BLOCKS[i] );
    for( int i=ri*BLOCK_SIZE; i<=r; i++ ) ret = min( ret, ar[i] );

    return ret;
}

int main() {
    #ifdef LU_SERIOUS
        freopen( "in.txt", "r", stdin );
//       freopen( "out.txt", "w+", stdout );
    #endif // LU_SERIOUS
    scanf( "%d", &t );
    for( cs=1; cs<=t; cs++ ) {
        scanf( "%d %d", &n, &q );
        BLOCK_SIZE = sqrt( n );
        init();
        for( int i=0; i<n; i++ ) {
            scanf( "%d", &ar[i] );
            update( i, ar[i] );
        }
        printf( "Case %d:\n", cs );
        for( int i=0; i<q; i++ ) {
            scanf( "%d %d", &x, &y );
            printf( "%d\n", query( x-1, y-1 ) );
        }
    }
    return 0;
}
```

2 Graph

2.1 ArticulationPoint

```
/**
An O(V+E) approach:
-------------------
- perform a DFS on the graph.
- compute d(i) and low(i) for each vertex 1 ... i
        d(i):  dfs number of i, represents the discovery time.
        low(i): the least dfn reachable from i throguh a path consisting
            of zero or
               more edges follwoing by zero or one back edges.
- vertex u is an AP if and only if:
        - u is the root of the dfs tree and has at least two children.
        - u is not the root and has a child v for which low(v) >= d(u).
 **/
#include <bits/stdc++.h>
using namespace std;

const int mx = 1e4 + 10;
vector < int > G[mx];
int tim, root, n, m, a, b;
int ap[mx], vis[mx], low[mx], d[mx], par[mx];

void ap_dfs(int u) {
    tim++;
    int cnt = 0;
    low[u] = tim;
    d[u] = tim;
    vis[u] = 1;
    int v;
    for(int i=0; i<G[u].size(); i++) {
        v = G[u][i];
        if( v == par[u] ) continue;
        if( !vis[v] ) {
            par[u] = v;
            ap_dfs( v );
            low[u] = min( low[u], low[v] );
            /// d[u] < low[v] if bridge is needed
            if( d[u] <= low[v] && u != root ) {
                ap[u] = 1;
            }
            cnt++;
        } else {
            low[u] = min( low[u], d[v] );
        }
        if( u == root && cnt > 1 ) ap[u] = 1;
    }
}

int main() {

        return 0;
}
```

2.2 CentroidDecomposition

```
/**
    Centroid Decomposition
    Running time:
        O( n * log ( n ) )
    Usage:
        - call rec() to decompose
    Input:
        - Graph
    Output:
        - Centroid Tree
    Tested Problems:
      CodeForces:
        321C/322E - Ciel the Commander
**/
const int sz = 1e5 + 10;
vector < int > G[sz];
char ans[sz];
int tr[sz], fl[sz];

void dfs( int u, int p ) {
    tr[u] = 1;
    for( int v: G[u] ) {
        if( v != p && !fl[v] ) {
            dfs( v, u );
            tr[u] += tr[v];
        }
    }
}

int centroid( int u ) {
    dfs( u, -1 );
    int ret = u;
    int found = 0, par = -1;
    while( 1 ) {
        found = 0;
        for( int v: G[ret] ) {
            if( !fl[v] && v != par && tr[v] >= ( tr[u] + 1 ) / 2 ) {
                found = 1;
                par = ret;
                ret = v;
                break;
            }
        }
        if( !found ) break;
    }
    return ret;
}

void rec( int u, char a ) {
    u = centroid( u );
    fl[u] = 1;
    ans[u] = a;
    for( int v: G[u] ) {
        if( !fl[v] ) rec( v, a + 1 );
    }
    return;
}
```

2.3 EdmondsKarp

```
/**
    Implementation of Edmonds-Karp max flow algorithm
    Running time:
        O(|V|*|E|^2)
    Usage:
        - add edges by add_edge()
        - call max_flow() to get maximum flow in the graph
    Input:
        - n, number of nodes
        - directed, true if the graph is directed
        - graph, constructed using add_edge()
        - source, sink
    Output:
        - Maximum flow
    Tested Problems:
      CF:
        653D - Delivery Bears
      UVA:
        820 - Internet Bandwidth
        10330 - Power Transmission
**/

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

const int INF = 1e9;

struct edmonds_karp {
    int n;
    vector < int > par;
    vector < bool > vis;
    vector < vector < int > > graph;

    edmonds_karp () {}
    edmonds_karp( int _n ) : n( _n ), par( _n ), vis( _n ), graph( _n,
         vector< int > ( _n, 0 ) ) {}
    ~edmonds_karp() {}

    void add_edge( int from, int to, int cap, bool directed ) {
        this->graph[ from ][ to ] += cap;
        this->graph[ to ][ from ] = directed ? graph[ to ][ from ] + cap :
             graph[ to ][ from ] ;
    }

    bool bfs( int src, int sink ) {
        int u;
        fill( vis.begin(), vis.end(), false );
        fill( par.begin(), par.end(), -1 );
        vis[ src ] = true;
        queue < int > q;
        q.push( src );
        while( !q.empty() ) {
            u = q.front();
            q.pop();
            if( u == sink ) return true;
            for(int i=0; i<n; i++) {
                if( graph[u][i] > 0 and not vis[i] ) {
                    q.push( i );
                    vis[ i ] = true;
                    par[ i ] = u;
                }
            }
        }
        return par[ sink ] != -1;
    }

    int min_val( int i ) {
        int ret = INF;
        for( ; par[ i ] != -1; i = par[ i ] ) {
            ret = min( ret, graph[ par[i] ][ i ] );
        }
        return ret;
    }

    void augment_path( int val, int i ) {
        for( ; par[ i ] != -1; i = par[ i ] ) {
            graph[ par[i] ][ i ] -= val;
            graph[ i ][ par[i] ] += val;
        }
    }

    int max_flow( int src, int sink ) {
        int min_cap, ret = 0;
        while( bfs( src, sink ) ) {
            augment_path( min_cap = min_val( sink ), sink );
            ret += min_cap;
        }
        return ret;
    }
};
```

2.4 EulerianPath

```
/**
    Implementation of Hierholzer's algorithm for finding Euler Path /
         Circuit
    Running time:
        O( | E | )
    Input:
        - adj, graph
    Tested Problems:
        CodeChef:
            TOURISTS - Tourists in Mancunia
**/
struct Edge;
typedef list< Edge >::iterator iter;

struct Edge {
    int next_vertex;
    iter reverse_edge;

    Edge( int next_vertex )
        :next_vertex(next_vertex)
        { }
};

const int sz = 1e5 + 10;
int num_vertices;
list < Edge > adj[ max_vertices ];

vector< int > path;

void find_path( int v ) {
    while( adj[v].size() > 0 ) {
        int vn = adj[v].front().next_vertex;
        adj[vn].erase( adj[v].front().reverse_edge );
        adj[v].pop_front();
        find_path( vn );
    }
    path.push_back( v );
}

void add_edge( int a, int b ) {
    adj[ a ].push_front( Edge( b ) );
    iter ita = adj[ a ].begin();
    adj[ b ].push_front( Edge( a ) );
    iter itb = adj[ b ].begin();
    ita->reverse_edge = itb;
    itb->reverse_edge = ita;
}
```

2.5 FloyedWarshall

```
/**
    Implementation of Floyd Warshall Alogrithm
    Running time:
        O( |v| ^ 3 )
    Input:
        - n, number vertex
        - graph, inputed as an adjacency matrix
    Tested Problems:
      UVA:
        544 - Heavy Cargo - MaxiMin path
        567 - Risk - APSP
**/

using vi = vector < int >;
using vvi = vector < vi >;

/// mat[i][i] = 0, mat[i][j] = distance from i to j, path[i][j] = i
void APSP( vvi &mat, vvi &path ) {

    int V = mat.size();
    for( int via=0; via; via<V; via++ ) {

        for( int from=0; from<V; from++ ) {

            for( int to=0; to<V; to++ ) {

                if( mat[ from ][ via ] + mat[ via ][ to ] < mat[ from ][
                     to ] ) {
                    mat[ from ][ to ] = mat[ from ][ via ] + mat[ via ][ to
                         ];
                    path[ from ][ to ] = path[ via ][ to ];
                }
            }
        }
    }
}

/// prints the path from i to j
void print( int i, int j ) {
    if( i != j ) {
        print( i, path[i][j] );
    }
    cout << j << "\n";
}

/// check if negative cycle exists
bool negative_cycle( vvi &mat ) {
    APSP( mat );
    return mat[0][0] < 0;
}

void transtitive_closure( vvi &mat ) {

    int V = mat.size();
    for( int via=0; via; via<V; via++ ) {

        for( int from=0; from<V; from++ ) {

            for( int to=0; to<V; to++ ) {

                mat[ from ][ to ] |= ( mat[ from ][ via ] & mat[ via ][ to
                    ] );
            }
        }
    }
}

/// finding a path between two nodes that maximizes the minimum cost
void mini_max( vvi &mat ) {

    int V = mat.size();
    for( int via=0; via; via<V; via++ ) {

        for( int from=0; from<V; from++ ) {

            for( int to=0; to<V; to++ ) {

                mat[ from ][ to ] = min( mat[ from ][ to ], max( mat[ from
                    ][ via ], mat[ via ][ to ] ) );
            }
        }
    }
}

/// finding a path between two nodes that minimizes the maximum cost
/// eg: max load a truck can carry from one node to another node where
/// the paths have weight limit
void maxi_min( vvi &mat ) {

    int V = mat.size();
    for( int via=0; via; via<V; via++ ) {

        for( int from=0; from<V; from++ ) {

            for( int to=0; to<V; to++ ) {

                mat[ from ][ to ] = max( mat[ from ][ to ], min( mat[ from
                    ][ via ], mat[ via ][ to ] ) );
            }
        }
    }
}
```

2.6 Kosaraju

```
#include <bits/stdc++.h>
using namespace std;
int p, t;
bool vis[1001];
vector<int> G[1001], gT[1001];
map<string,int> mp;
stack < int > top_sorted;

void dfs_top_sort(int u) {
        vis[u] = true;
        for(int v: G[u]) {
                if(!vis[v]) {
                        dfs_top_sort( v );
                }
        }
        top_sorted.push( u );
}

void top_sort() {
        for(int i=1; i<=p; i++) {
                if(!vis[i]) {
                        dfs_top_sort(i);
                }
        }
}

void dfs_kosaraju(int u) {
        vis[u] = true;
        for(int v: gT[u]) {
                if(!vis[v]) {
                        dfs_kosaraju( v );
                }
        }
}

int kosaraju() {
        memset( vis, false, sizeof(vis) );
        top_sort();
        int u, ret = 0;
        memset( vis, false, sizeof(vis) );
        while(!top_sorted.empty()) {
                u = top_sorted.top();
                top_sorted.pop();
                if(!vis[u])
                        dfs_kosaraju( u ), ret++;
        }
        return ret;
}
```

2.7 Kruskal

```
/**
    Implementation of Kruskal's minimum spanning tree algorithm
    Running time:
        O(|E|log|V|)
    Usage:
        - initialize by calling init()
        - add edges by add_edge()
        - call kruskal() to generate minimum spanning tree
    Input:
        - n, number of nodes, provided when init() is called
        - graph, constructed using add_edge()
    Output:
        - weight of minimum spanning tree
        - prints the mst
    Tested Problems:
        UVA:
            1208 - Oreon
*/

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

struct edge {
    int u, v, cost;
    bool operator < (const edge& other) const{
        if( other.cost == this->cost ) {
        if( other.u == this->u ) {
            return other.v > this->v;
        } else {
            return other.u > this->u;
        }
        } else {
            return other.cost > this->cost;
        }
    }
};

vector< edge > edges;
vector < int > par, cnt, rank;
int N;

void init( int n ) {
    N = n;
    par.resize( n );
    cnt.resize( n );
    rank.resize( n );
}

void add_edge( int u, int v, int c ) {
    edges.push_back( { u, v, c } );
}
```

```
void make_set() {
    for(int i=0; i<N; i++) {
        par[i] = i;
        cnt[i] = 1;
        rank[i] = 0;
    }
}

int find_rep( int x ) {
    if(x != par[ x ]) {
        par[ x ] = find_rep( par[ x ] );
    }
    return par[ x ];
}

int kruskal() {
    int ret = 0;
    make_set();
    sort( edges.begin(), edges.end() );
    cout << "Case " << ++cs << ":\n";
    for( edge e : edges ) {
        int u = e.u;
        int v = e.v;
        if( ( u = find_rep( u ) ) != ( v = find_rep( v ) ) ) {
            if( rank[ u ] < rank[ v ] ) {
                cnt[ v ] += cnt[ u ];
                par[ u ] = par[ v ];
            } else {
                rank[ u ] = max( rank[ u ], rank[ v ] + 1 );
                cnt[ u ] += cnt[ v ];
                par[ v ] = par[ u ];
            }
            cout << city[ e.u ] << "-" << city[ e.v ] << " " << e.cost <<
                 "\n";
            ret += e.cost;
        }
    }
    return ret;
}
```

3 Matrix

3.1 MatrixExpo

```
/**
    Implementation of Matrix Exponentiation
    Running time:
        O( log( n ) )
    Input:
        - n, exponent
        - recurrence matrix, power of which to be determined
    Tested Problems:
      UVA:
        10229 - Modular Fibonacci
        10518 - How Many Calls?
        12470 - Tribonacci
**/
const int mat_sz = 2;
struct Matrix {
    int a[mat_sz][mat_sz];
    void clear() {
        memset(a, 0, sizeof(a));
    }
    void one() {
        for( int i=0; i<mat_sz; i++ ) {
            for( int j=0; j<mat_sz; j++ ) {
                a[i][j] = i == j;
            }
        }
    }
    Matrix operator + (const Matrix &b) const {
        Matrix tmp;
        tmp.clear();
        for (int i = 0; i < mat_sz; i++) {
            for (int j = 0; j < mat_sz; j++) {
                tmp.a[i][j] = a[i][j] + b.a[i][j];
                if (tmp.a[i][j] >= mod) {
                    tmp.a[i][j] -= mod;
                }
            }
        }
        return tmp;
    }
    Matrix operator * (const Matrix &b) const {
        Matrix tmp;
        tmp.clear();
        for (int i = 0; i < mat_sz; i++) {
            for (int j = 0; j < mat_sz; j++) {
                for (int k = 0; k < mat_sz; k++) {
                    tmp.a[i][k] += (long long)a[i][j] * b.a[j][k] % mod;
                    if (tmp.a[i][k] >= mod) {
                        tmp.a[i][k] -= mod;
                    }
                }
            }
        }
        return tmp;
    }
    Matrix pw(int x) {
        Matrix ans, num = *this;
        ans.one();
        while (x > 0) {
            if (x & 1) {
                ans = ans * num;
            }
            num = num * num;
            x >>= 1;
        }
        return ans;
    }
};
```

4 String

4.1 AhoCorasick

```
const int sz = 1e6 + 10;
const int MAX = 150 * 70 + 100;
char inp[sz], s[155][75];
int cnt[155];

struct AhoCorasick {
    vector < int > mark[MAX + 7];
    int state, failure[MAX + 7];
    int trie[MAX + 7][ 26 ];
```

```
    AhoCorasick() {
        init();
    }

    void init() {
        mark[0].clear();
        fill( trie[0], trie[0] + 26, -1 );
        state = 0;
    }

    int value( char c ) {
        return c - ’a’;
    }

    void add( char *s, int t ) {
        int root = 0, id;
        for( int i=0; s[i]; i++ ) {
            id = value( s[i] );
            if( trie[ root ][ id ] == -1 ) {
                trie[ root ][ id ] = ++state;
                mark[state].clear();
                fill( trie[state], trie[state + 1] + 26, - 1 );
            }
            root = trie[ root ][ id ];
        }
        mark[ root ].push_back( t );
    }

    void computeFailure() {
        queue < int > Q;
        failure[0] = 0;
        for( int i=0; i<26; i++ ) {
            if( trie[ 0 ][ i ] != -1 ) {
                failure[ trie[ 0 ][ i ] ] = 0;
                Q.push( trie[ 0 ][ i ] );
            }
            else trie[ 0 ][ i ] = 0;
        }
        while( !Q.empty() ) {
            int u = Q.front();
            Q.pop();
            for( int v: mark[ failure[ u ] ] ) mark[ u ].push_back( v );
            for( int i=0; i<26; i++ ) {
                if( trie[ u ][ i ] != -1 ) {
                    failure[ trie[ u ][ i ] ] = trie[ failure[ u ] ][ i ];
                    Q.push( trie[ u ][ i ] );
                }
                else trie[ u ][ i ] = trie[ failure[ u ] ][ i ];
            }
        }
    }
} automata;

void countFreq() {
    for( int i=0,root=0,id; inp[i]; i++ ) {
        id = automata.value( inp[i] );
        root = automata.trie[ root ][ id ];
        if( root == 0 ) continue;
        for( int v: automata.mark[ root ] ) cnt[v]++;
    }
}
```

4.2 KMP

```
/// complexity : o( n + m )
///solution reference loj 1255 Substring Frequency
#include <bits/stdc++.h>
using namespace std;

int t;
const int mx = 1e6 + 10;
char a[mx], b[mx];
int table[mx], lenA, lenB;

void hash_table( char *s ) {
        table[ 0 ] = 0;
        int i = 1, j = 0;
        while( i < lenB ) {
                if( s[i] == s[j] ) {
                        j++;
                        table[ i ] = j;
                        i++;
                } else {
                        if( j ) {
                                j = table[ j - 1 ];
                        } else {
                                table[ i ] = 0;
                                i++;
                        }
                }
        }
}

int kmp( char *s, char *m ) {
        hash_table( m );
        int i = 0, j = 0;
        int ans = 0;
        while( i < lenA ) {
                while( i < lenA && j < lenB && s[i] == m[j] ) {
                        i++;
                        j++;
                }
                if( j == lenB ) {
                        j = table[ j - 1 ];
                        ans++;
                } else if( i < lenA && s[i] != m[j] ) {
                        if( j ) {
                                j = table[ j - 1 ];
                        } else {
                                i++;
                        }
                }
        }
        return ans;
}

int main() {
#ifdef LU_SERIOUS
        freopen("in.txt", "r", stdin);
#endif // LU_SERIOUS
        scanf( "%d", &t );
        for(int cs=1; cs<=t; cs++) {
                lenA = 0; lenB = 0;
                scanf("%s", &a);
                scanf("%s", &b);
                lenA = strlen( a );
                lenB = strlen( b );
                printf( "Case %d: %d\n", cs, kmp( a, b ) );
        }
        return 0;
}
```

4.3 Manacher

```
const int sz = 2e5 + 10;
char inp[sz], str[sz];
int LPS[sz];

int call(){
    int len = 0;
    str[ len++ ]='*';
    for( int i=0; inp[i]; i++ ) {
        str[ len++ ] = inp[i];
        str[ len++ ] = '*';
    }
    str[ len ] = '\0';
    int c = 0, r = 0, ans = 0;
    for( int i=1; i<len-1; i++ ) {
        int _i = c - ( i - c );
        if( r > i ) LPS[i] = min( LPS[_i] , r - i );
        else LPS[i] = 0;
        while( i - 1 - LPS[i] >= 0 && str[ i - 1 - LPS[i] ] == str[ i + 1
             + LPS[i] ] ) {
            LPS[i]++;
        }
        if( i + LPS[i] > r ) {
            r = i + LPS[i];
            c = i;
        }
        ans = max( ans, LPS[i] );
    }
    return ans;
}
```

4.4 SArray(n logn)

```
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

typedef pair<int, int> ii;

#define MAX_N 100010                     // second approach: O(n log n)
char T[MAX_N];                // the input string, up to 100K characters
int n;                                    // the length of input string
int RA[MAX_N], tempRA[MAX_N];    // rank array and temporary rank array
int SA[MAX_N], tempSA[MAX_N]; // suffix array and temporary suffix array
int c[MAX_N];                               // for counting/radix sort

char P[MAX_N];               // the pattern string (for string matching)
int m;                                  // the length of pattern string

int Phi[MAX_N];                   // for computing longest common prefix
int PLCP[MAX_N];
int LCP[MAX_N]; // LCP[i] stores the LCP between previous suffix T+SA[i-1]
// and current suffix T+SA[i]

bool cmp(int a, int b)
{
    return strcmp(T + a, T + b) < 0; // compare
}

void constructSA_slow()             // cannot go beyond 1000 characters
{
    for (int i = 0; i < n; i++) SA[i] = i; // initial SA: {0, 1, 2, ...,
        n-1}
    sort(SA, SA + n, cmp); // sort: O(n log n) * compare: O(n) = O(n^2
        log n)
}

void countingSort(int k)                                     // O(n)
{
    int i, sum, maxi = max(300, n); // up to 255 ASCII chars or length of
        n
    memset(c, 0, sizeof c);                     // clear frequency table
    for (i = 0; i < n; i++)    // count the frequency of each integer rank
        c[i + k < n ? RA[i + k] : 0]++;
    for (i = sum = 0; i < maxi; i++)
    {
        int t = c[i];
        c[i] = sum;
        sum += t;
    }
    for (i = 0; i < n; i++)       // shuffle the suffix array if necessary
        tempSA[c[SA[i]+k < n ? RA[SA[i]+k] : 0]++] = SA[i];
    for (i = 0; i < n; i++)                 // update the suffix array SA
        SA[i] = tempSA[i];
}
```

```
void constructSA()        // this version can go up to 100000 characters
{
    int i, k, r;
    for (i = 0; i < n; i++) RA[i] = T[i];            // initial rankings
    for (i = 0; i < n; i++) SA[i] = i; // initial SA: {0, 1, 2, ..., n-1}
    for (k = 1; k < n; k <<= 1)      // repeat sorting process log n times
    {
        countingSort(k); // actually radix sort: sort based on the second item
        countingSort(0);        // then (stable) sort based on the first item
        tempRA[SA[0]] = r = 0;          // re-ranking; start from rank r = 0
        for (i = 1; i < n; i++)                  // compare adjacent suffixes
            tempRA[SA[i]] = // if same pair => same rank r; otherwise, increase r
                (RA[SA[i]] == RA[SA[i-1]] && RA[SA[i]+k] == RA[SA[i-1]+k]) ? r : ++r;
        for (i = 0; i < n; i++)                  // update the rank array RA
            RA[i] = tempRA[i];
        if (RA[SA[n-1]] == n-1) break;           // nice optimization trick
    }
}

void computeLCP_slow()
{
    LCP[0] = 0;                                         // default value
    for (int i = 1; i < n; i++)              // compute LCP by definition
    {
        int L = 0;                                     // always reset L to 0
        while (T[SA[i] + L] == T[SA[i-1] + L]) L++; // same L-th char, L++
        LCP[i] = L;
    }
}

void computeLCP()
{
    int i, L;
    Phi[SA[0]] = -1;                                    // default value
    for (i = 1; i < n; i++)                          // compute Phi in O(n)
        Phi[SA[i]] = SA[i-1];  // remember which suffix is behind this suffix
    for (i = L = 0; i < n; i++)           // compute Permuted LCP in O(n)
    {
        if (Phi[i] == -1)
        {
            PLCP[i] = 0;   // special case
            continue;
        }
        while (T[i + L] == T[Phi[i] + L]) L++;   // L increased max n times
        PLCP[i] = L;
        L = max(L-1, 0);                         // L decreased max n times
    }
    for (i = 0; i < n; i++)                          // compute LCP in O(n)
        LCP[i] = PLCP[SA[i]]; // put the permuted LCP to the correct position
}

ii stringMatching()                      // string matching in O(m log n)
{
    int lo = 0, hi = n-1, mid = lo;         // valid matching = [0..n-1]
    while (lo < hi)                                    // find lower bound
    {
        mid = (lo + hi) / 2;                          // this is round down
        int res = strncmp(T + SA[mid], P, m); // try to find P in suffix 'mid'
        if (res >= 0) hi = mid;     // prune upper half (notice the >= sign)
        else          lo = mid + 1;         // prune lower half including mid
    }                                    // observe '=' in "res >= 0" above
    if (strncmp(T + SA[lo], P, m) != 0) return ii(-1, -1); // if not found
    ii ans;
    ans.first = lo;
    lo = 0;
    hi = n - 1;
    mid = lo;
    while (lo < hi)           // if lower bound is found, find upper bound
    {
        mid = (lo + hi) / 2;
        int res = strncmp(T + SA[mid], P, m);
        if (res > 0) hi = mid;                          // prune upper half
        else         lo = mid + 1;          // prune lower half including mid
    }                          // (notice the selected branch when res == 0)
    if (strncmp(T + SA[hi], P, m) != 0) hi--;           // special case
    ans.second = hi;
    return ans;
} // return lower/upperbound as first/second item of the pair, respectively
```

```
ii LRS()                 // returns a pair (the LRS length and its index)
{
  int i, idx = 0, maxLCP = -1;
  for (i = 1; i < n; i++)                       // O(n), start from i = 1
    if (LCP[i] > maxLCP)
      maxLCP = LCP[i], idx = i;
  return ii(maxLCP, idx);
}

int owner(int idx)
{
  return (idx < n-m-1) ? 1 : 2;
}

ii LCS()                 // returns a pair (the LCS length and its index)
{
  int i, idx = 0, maxLCP = -1;
  for (i = 1; i < n; i++)                       // O(n), start from i = 1
    if (owner(SA[i]) != owner(SA[i-1]) && LCP[i] > maxLCP)
      maxLCP = LCP[i], idx = i;
  return ii(maxLCP, idx);
}

int main()
{
  //printf("Enter a string T below, we will compute its Suffix
      Array:\n");
  strcpy(T, "GATAGACA");
  //T = "ABCDE"
  n = (int)strlen(T);
  T[n++] = ’$’;
  // if ’\n’ is read, uncomment the next line
  //T[n-1] = ’$’; T[n] = 0;

  constructSA_slow();                                // O(n^2 log n)
  printf("The Suffix Array of string T = ’%s’ is shown below (O(n^2 log
      n) version):\n", T);
  printf("i\tSA[i]\tSuffix\n");
  for (int i = 0; i < n; i++) printf("%2d\t%2d\t%s\n", i, SA[i], T +
      SA[i]);

  constructSA();                                      // O(n log n)
  printf("\nThe Suffix Array of string T = ’%s’ is shown below (O(n log
      n) version):\n", T);
```

```
printf("i\tSA[i]\tSuffix\n");
for (int i = 0; i < n; i++) printf("%2d\t%2d\t%s\n", i, SA[i], T +
    SA[i]);

computeLCP();                                              // O(n)

// LRS demo
ii ans = LRS();              // find the LRS of the first input string
char lrsans[MAX_N];
strncpy(lrsans, T + SA[ans.second], ans.first);
printf("\nThe LRS is ’%s’ with length = %d\n\n", lrsans, ans.first);

// stringMatching demo
//printf("\nNow, enter a string P below, we will try to find P in
    T:\n");
strcpy(P, "A");
m = (int)strlen(P);
// if ’\n’ is read, uncomment the next line
//P[m-1] = 0; m--;
ii pos = stringMatching();
if (pos.first != -1 && pos.second != -1)
{
  printf("%s is found SA[%d..%d] of %s\n", P, pos.first, pos.second,
      T);
  printf("They are:\n");
  for (int i = pos.first; i <= pos.second; i++)
    printf(" %s\n", T + SA[i]);
}
else printf("%s is not found in %s\n", P, T);

// LCS demo
//printf("\nRemember, T = ’%s’\nNow, enter another string P:\n", T);
// T already has ’$’ at the back
strcpy(P, "CATA");
m = (int)strlen(P);
// if ’\n’ is read, uncomment the next line
//P[m-1] = 0; m--;
strcat(T, P);                                           // append P
strcat(T, "#");                                // add ’$’ at the back
n = (int)strlen(T);                                     // update n

// reconstruct SA of the combined strings
constructSA();                                         // O(n log n)
computeLCP();                                              // O(n)
printf("\nThe LCP information of ’T+P’ = ’%s’:\n", T);
```

```
    printf("i\tSA[i]\tLCP[i]\tOwner\tSuffix\n");
    for (int i = 0; i < n; i++)
        printf("%2d\t%2d\t%2d\t%2d\t%s\n", i, SA[i], LCP[i], owner(SA[i]),
             T + SA[i]);

    ans = LCS();       // find the longest common substring between T and
        P
    char lcsans[MAX_N];
    strncpy(lcsans, T + SA[ans.second], ans.first);
    printf("\nThe LCS is '%s' with length = %d\n", lcsans, ans.first);

    return 0;
}
```

4.5 SmallestStringRotation

```
/**
    Implementation of Lexicographically smallest string rotation
    Running time:
        O( 2 * s.size() )
    Input:
        - s, string
    Tested Problems:
        UVA:
            719 - Glass Beads
        DevSkill:
            DCP-207: Mina and Raju Part 2
**/
const int sz = 1e5 + 10;
int f[sz];

int calc( const string& s ) {
    int n = s.size();
    string t = s + s;
    memset( f, -1, sizeof f );
    int k = 0;
    for( int j = 1; j < 2 * n; ++j ) {
        int i = f[j - k - 1];
        while( i != -1 && t[j] != t[k + i + 1] ) {
            if( t[j] < t[k + i + 1] ) {
                k = j - i - 1;
            }
            i = f[i];
        }
        if( i == -1 && t[j] != t[k + i + 1] ) {
            if( t[j] < t[k + i + 1] ) {
                k = j;
            }
            f[j - k] = -1;
        } else {
            f[j - k] = i + 1;
        }
    }
    return k;
}
```

4.6 SuffixArray(n logn logn)

```
/**
    Implementation of Suffix Array
    Running time:
        O( n log( n ) log( n ) )
    Input:
        - s, string for that suffix array to be completed
    Output:
        - Suffix Array
    Tested Problems:
      SPOJ:
        SARRAY - Suffix Array
**/
#include <bits/stdc++.h>
using namespace std;

struct suffix {
    int index;
    int rank[2];
    bool operator < ( const suffix &other ) const {
        if( this->rank[0] == other.rank[0] ) {
            return this->rank[1] < other.rank[1];
        }
        return this->rank[0] < other.rank[0];
    }
};

vector < int > buildSuffixArray( const string &s ) {
    int n = int( s.size() );
    vector < int > sufArray;
    vector < suffix > suffixes( n );
    for( int i=0; i<n; i++ ) {
        suffixes[i].index = i;
        suffixes[i].rank[0] = s[i];
        suffixes[i].rank[1] = i + 1 < n ? s[i+1] : -1;
    }
    vector < int > ind( n );
    int nextIndex, rank, prev_rank, n_2 = n << 1;
    sort( suffixes.begin(), suffixes.end() );
    for( int k=4; k<n_2; k<<=1 ) {
        rank = 0;
        prev_rank = suffixes[0].rank[0];
        suffixes[0].rank[0] = rank;
        ind[ suffixes[0].index ] = 0;
        for( int i=1; i<n; i++ ) {
            if( suffixes[i].rank[0] == prev_rank && suffixes[i].rank[1] ==
                 suffixes[i-1].rank[1] ) {
                prev_rank = suffixes[i].rank[0];
                suffixes[i].rank[0] = rank;
            } else {
                prev_rank = suffixes[i].rank[0];
                suffixes[i].rank[0] = ++rank;
            }
            ind[ suffixes[i].index ] = i;
        }
        for( int i=0; i<n; i++ ) {
            nextIndex = suffixes[i].index + k / 2;
            suffixes[i].rank[1] = nextIndex < n ? suffixes[ ind[ nextIndex
                 ] ].rank[0] : -1;
        }
        sort( suffixes.begin(), suffixes.end() );
    }
    for( const suffix suf: suffixes ) {
        sufArray.push_back( suf.index );
    }
    return sufArray;
}

int main() {
    #ifdef CLown1331
        freopen( "in.txt", "r", stdin );
    #endif /// CLown1331
    string s;
    while( cin >> s ) {
        vector < int > sufArray = buildSuffixArray( s );
        for( const int ind: sufArray ) {
            cout << ind << "\n";
        }
        cerr << "----\n";
    }
    return 0;
}
```

4.7 Zalgo

```
int L = 0, R = 0;
for( int i = 1; i < n; i++ ) {
        if ( i > R ) {
                L = R = i;
                while ( R < n && s[R-L] == s[R] ) R++;
                z[i] = R-L; R--;
        } else {
        int k = i-L;
                if ( z[k] < R-i+1 ) z[i] = z[k];
                else {
                        L = i;
                        while ( R < n && s[R-L] == s[R] ) R++;
                        z[i] = R-L; R--;
                }
        }
}
int maxz = 0, res = 0;
for ( int i = 1; i < n; i++ ) {
        if ( z[i] == n-i && maxz >= n-i ) { res = n-i; break; }
        maxz = max( maxz, z[i]) ;
}
```