quwsarohi@gmail.com Section: Data Structure Page : 3

**// Segment Tree Lazy Propagation**

**// (l, r) : tree segment, (x, y) : update segment**

vector<pair<ull, ull> > tree;

void update(ll pos, ll l, ll r, ll x, ll y, ll val) {

if(y < l || x > r)

return;

if(x <= l && r <= y) { // Tree segment in update segment

tree[pos].fi += (r-l+1)\*val;

tree[pos].se += val; // Propagate

return;

}

ll mid = (l+r)/2LL;

update(pos\*2LL, l, mid, x, y, val);

update(pos\*2LL + 1, mid+1, r, x, y, val);

tree[pos].fi = tree[pos\*2].fi + tree[pos\*2+1].fi + (r-l+1)\*tree[pos].se;

}

ll query(ll pos, ll l, ll r, ll x, ll y, ll carry) { // Pass propagate value through carry

if(y < l || x > r)

return 0;

if(x <= l && r <= y)

return tree[pos].fi + (carry \* (r-l+1));

ll mid = (l+r)/2LL;

ll lft = query(pos\*2LL, l, mid, x, y, carry + tree[pos].se);

ll rht = query(pos\*2LL + 1, mid+1, r, x, y, carry + tree[pos].se);

return lft + rht;

}

**// SPOJ GSS3 - Can you answer these queries III**

**// Segment Tree (Range Maximum Sum, Query, Update)**

struct node {

ll sum, prefix, suffix, ans;

node(ll val = 0) {

sum = prefix = suffix = ans = val;

}

void merge(node left, node right) {

sum = left.sum + right.sum;

prefix = max(left.prefix, left.sum+right.prefix);

suffix = max(right.suffix, right.sum+left.suffix);

ans = max(left.ans, max(right.ans, left.suffix+right.prefix));

}

};

node tree[201000];

ll v[50010];

void init(int pos, int l, int r) { // Call with init(1, 1, value\_len)

if(l == r) {

tree[pos] = node(v[l]);

return;

}

int mid = (l+r)/2;

init(pos\*2, l, mid);

init(pos\*2+1, mid+1, r);

quwsarohi@gmail.com Section: Data Structure Page : 4

tree[pos] = node(-INF);

tree[pos].merge(tree[pos\*2], tree[pos\*2+1]);

}

void update(int pos, int l, int r, int x, int val) {

if(l == r && l == x) {

tree[pos] = node(val);

return;

}

if(x < l || r < x)

return;

int mid = (l+r)/2;

update(pos\*2, l, mid, x, val);

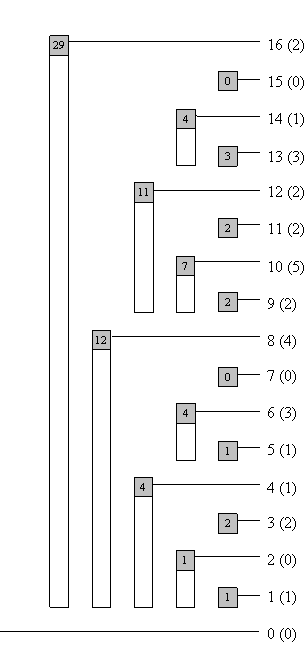
update(pos\*2+1, mid+1, r, x, val);

tree[pos] = node(-INF);

tree[pos].merge(tree[pos\*2], tree[pos\*2+1]);

}

node query(int pos, int l, int r, int x, int y) {

 if(r < x || y < l)

return node(-INF);

if(x <= l && r <= y)

return tree[pos];

int mid = (l+r)/2;

node lft = query(pos\*2, l, mid, x, y);

node rht = query(pos\*2+1, mid+1, r, x, y);

node parent = node(-INF);

parent.merge(lft, rht);

return parent;

}

**// 1D Fenwick Tree**

long long tree[100010];

int MaxVal;

void update(int idx, int val) {

while(idx <= MaxVal) {

tree[idx] += val;

idx += (idx & -idx);

}

}

long long read(int idx) {

long long sum = 0;

while(idx > 0) {

sum += tree[idx];

idx -= (idx & -idx);

}

return sum;

}

long long readSingle(int idx) {

long long sum = tree[idx]; Fig: Fenwick Tree with values

if(idx > 0) {

quwsarohi@gmail.com Section: Data Structure Page : 5

int z = idx - (idx & -idx);

--idx;

while(idx != z) {

sum -= tree[idx];

idx -= (idx & -idx);

}

}

return sum;

}

**// Tested Version**

**// Complexity : LogN**

int binarySearch(int cSum) { // Binary search for the cumulative sum

int idx = 0, tIdx; // Returns the greater index if value is present more than once

int bitmask = highBitMaxVal;

while(bitmask != 0 && idx < MaxVal) {

tIdx = idx + bitmask;

if(cSum == tree[tIdx])

return tIdx;

if(cSum > tree[tIdx]) {

idx = tIdx;

cSum -= tree[tIdx];

}

bitmask >>= 1;

}

if(cSum != 0)

return -1;

else

return idx;

}

// Complexity : (logN)^2

int binarySearch(int k) { // Trustworthy Binary Search (Tested)

int low = 0, high = MaxVal, mid;

while(high - low > 1) {

mid = (low + high) >> 1;

if(read(mid) >= k)

high = mid;

else

low = mid;

}

return high;

}

**// 2D Fenwick Tree**

long long tree[1010][1010];

int xMax = 1001, yMax = 1001;

void update(int x, int y, int val) { // Updates from min point to max point

int y1;

while(x <= xMax) {

y1 = y;

quwsarohi@gmail.com Section: Data Structure Page : 6

while(y1 <= yMax) {

tree[x][y1] += val;

y1 += (y1 & -y1);

}

x += (x & -x);

}

}

long long read(int x, int y) {

long long sum = 0;

int y1;

while(x > 0) {

y1 = y;

while(y1 > 0) {

sum += tree[x][y1];

y1 -= (y1 & -y1);

}

x -= (x & -x);

}

return sum;

}

// LightOJ 1097 - Lucky Number

// Given numbers 1 to N, Grab 2nd number and start deleting every 2nd number that occurs in series, grab 3rd number (suppose x)

// and delete every x’th number from the series, continue while there exists n’th value in the remaining sequence

int BITsize() { // Returns remaining numbers in the sequence (if tree[i] == 1, then value exists)

return read(MaxVal);

}

void build() {

int lim;

for(int i = 1; i <= MaxVal; ++i) { //Adding all numbers in BIT

update(i, 1);

for(int i = 2; i <= BITsize(); i+=2) //Marking all even numbers

v.push\_back(i);

for(int i = 0; i < (int)v.size(); ++i) // Deleting all even numbers

update(v[i], -1);

for(int i = 2; i <= (lim = BITsize()); ++i) { // Starting from 2nd index

v.clear();

int pos = binarySearch(i);

if(pos > lim)

break;

for(int j = pos; j <= lim; j += pos) // Marking

v.push\_back(binarySearch(j));

for(int j = 0; j < (int)v.size(); ++j) // Deleting

update(v[j], -1);

}

}