Data Structure

Created	@2025년 5월 17일 오후 6:36
Tags	2-SAT FW FW2D LAZY SEG SEG2D gmSEG iterLAZY iterSEG

SEG

```
// usage: SEG(n, arr); update(tar, val); ll sum = query(qs, qe);
// memo : 0-indexed
struct SEG {
    ll n;
    vl seg, arr;

    SEG(ll n, vl& arr): n(n),arr(arr),seg(n<<2){
        init(1,0,n-1);
    }

    void init(ll node,ll l,ll r) {
        if (l==r) {
            seg[node]=arr[l];
            return;
        }
        ll m=(l+r)>>1;
        init(node<<1,l,m);
        init(node<<1|1,m+1,r);
        seg[node]=seg[node<<1]+seg[node<<1|1];
    }

    ll range_sum(ll node, ll l, ll r, ll s, ll e) {
        if (r<s||e<l) return 0;
        if (s<=l&&r<=e) return seg[node];
        ll m=(l+r)>>1;
        return range_sum(node<<1,l,m,s,e) + range_sum(node<<1|1,m+1,r,s,e);
    }

    void point_update(ll node, ll l, ll r, ll tar, ll val) {
        if (r<tar||tar<l) return;
        if (l==r) {
            seg[node]=val;
            return;
        }
        ll m=(l+r)>>1;
        point_update(node<<1,l,m,tar,val);
        point_update(node<<1|1,m+1,r,tar,val);
        seg[node]=seg[node<<1]+seg[node<<1|1];
    }
    void update(ll tar,ll val){point_update(1,0,n-1,tar,val);}
    ll query(ll s,ll e){return range_sum(1,0,n-1,s,e);}
};
```

LAZY

```
// usage: LAZY(n, arr); update(qs, qe, val); ll sum = query(qs, qe);
// memo : 0-indexed
struct LAZY {
    ll n;
```

```
vl seg, lazy, arr;

LAZY(ll n, vl& arr): n(n),arr(arr),seg(n<<2),lazy(n<<2){
    init(1,0,n-1);
}

void init(ll node,ll l,ll r) {
    if (l==r) {
        seg[node]=arr[l];
        return;
    }
    ll m=(l+r)>>1;
    init(node<<1,l,m);
    init(node<<1|1,m+1,r);
    seg[node]=seg[node<<1]+seg[node<<1|1];
}

void relax(ll node, ll l, ll r) {
    if (lazy[node]==0) return;
    if (l<r) {
        ll m = (l+r)>>1;
        seg[node<<1]+=(m-l+1)*lazy[node];
        lazy[node<<1]+=lazy[node];
        seg[node<<1|1]+=(r-m)*lazy[node];
        lazy[node<<1|1]+=lazy[node];
    }
    lazy[node]=0;
}

ll range_sum(ll node, ll l, ll r, ll s, ll e) {
    if (r<s||e<l) return 0;
    if (s<=l&&r<=e) return seg[node];
    relax(node,l,r);
    ll m=(l+r)>>1;
    return range_sum(node<<1,l,m,s,e) + range_sum(node<<1|1,m+1,r,s,e);
}

void range_update(ll node, ll l, ll r, ll s, ll e, ll val) {
        if (r<s||e<l) return;
        if (s<=l&&r<=e) {
            seg[node]+=(r-l+1)*val;
            lazy[node]+=val;
            return;
        }
        relax(node,l,r);
        ll m=(l+r)>>1;1
        range_update(node<<1,l,m,s,e,val);
        range_update(node<<1|1,m+1,r,s,e,val);
        seg[node]=seg[node<<1]+seg[node<<1|1];
    }
    void update(ll s,ll e,ll val){range_update(1,0,n-1,s,e,val);}
    ll query(ll s,ll e){return range_sum(1,0,n-1,s,e);}
};
```

FW

```
//FW fw(n); fw.add(idx,val); ll sum=fw.range_query(l,r);
//memo : 0-indexed
struct FW {
    ll n;
    vl fw;
    FW(ll n):n(n),fw(n+1) {}

    void add(ll i,ll val) {
        for (++i;i<=n;i+=i&-i) fw[i]+=val;
    }

    ll query(ll i) {
        ll ret=0;
        for (++i;i>0;i-=i&-i) ret+=fw[i];
        return ret;
    }
    ll range_query(ll l, ll r){return query(r)-query(l-1);}
};
```

FW2D

```
//FW2D fw2d(n); fw2d.add(i,j,val); ll sum=fw2d.range_query(x1,y1,x2,y2);
//memo : 0-indexed
struct FW2D {
    ll n,m;
    vvl fw;
    FW2D(ll n, ll m):n(n),m(m),fw(n+1,vl(m+1)) {}

    void add(ll i,ll j,ll val) {
        for (++i;i<=n;i+=i&-i)
            for (ll k=j+1;k<=m;k+=k&-k)
                fw[i][k]+=val;
    }

    ll query(ll i,ll j) {
        ll ret=0;
        for (++i;i>0;i-=i&-i)
            for (ll k=j+1;k>0;k-=k&-k)
                ret+=fw[i][k];
        return ret;
    }

    ll range_query(ll x1, ll y1, ll x2, ll y2) {
        return query(x2,y2)+query(x1-1,y1-1)-query(x1-1,y2)-query(x2,y1-1);
    }
};
```

SEG2D

```
//usage: SEG2D seg2d(n,m,arr); seg2d.update(i,j,val); ll sum=seg2d.query
(x1,y1,x2,y2);
```

```
//memo: 0-indexed
struct SEG2D {
    ll n,m;
    vvl seg,arr;

    SEG2D(ll n,ll m,vvl& arr): n(n),m(m),arr(arr) {
        seg.assign(n<<2,vl(m<<2));
        build_x(1,0,n-1);
    }

    void build_y(ll vx,ll vy,ll lx,ll ly,ll rx,ll ry) {
        if (ly==ry) {
            if (lx==rx) seg[vx][vy]=arr[lx][ly];
            else seg[vx][vy]=seg[vx<<1][vy]+seg[vx<<1|1][vy];
            return;
        }
        ll mid=(ly+ry)>>1;
        build_y(vx,vy<<1,lx,ly,rx,mid);
        build_y(vx,vy<<1|1,lx,mid+1,rx,ry);
        seg[vx][vy]=seg[vx][vy<<1]+seg[vx][vy<<1|1];
    }
    void build_x(ll vx,ll lx,ll rx) {
        if (lx!=rx) {
            ll mid=(lx+rx)>>1;
            build_x(vx<<1,lx,mid);
            build_x(vx<<1|1,mid+1,rx);
        }
        build_y(vx,1,lx,0,rx,m-1);
    }
    void update_y(ll vx,ll vy,ll lx,ll ly,ll rx,ll ry,ll x,ll y,ll val) {
        if (ly==ry) {
            if (lx==rx) seg[vx][vy]=val;
            else seg[vx][vy]=seg[vx<<1][vy]+seg[vx<<1|1][vy];
            return;
        }
        ll mid=(ly+ry)>>1;
        if (y<=mid) update_y(vx,vy<<1,lx,ly,rx,mid,x,y,val);
        else update_y(vx,vy<<1|1,lx,mid+1,rx,ry,x,y,val);
        seg[vx][vy]=seg[vx][vy<<1]+seg[vx][vy<<1|1];
    }
    void update_x(ll vx,ll lx,ll rx,ll x,ll y,ll val) {
        if (lx!=rx) {
            ll mid=(lx+rx)>>1;
            if (x<=mid) update_x(vx<<1,lx,mid,x,y,val);
            else update_x(vx<<1|1,mid+1,rx,x,y,val);
        }
        update_y(vx,1,lx,0,rx,m-1,x,y,val);
    }

    ll query_y(ll vx,ll vy,ll ly,ll ry,ll y1,ll y2) {
        if (ry<y1||y2<ly) return 0;
        if (y1<=ly&&ry<=y2) return seg[vx][vy];
        ll mid=(ly+ry)>>1;
        return query_y(vx,vy<<1,ly,mid,y1,y2)+query_y(vx,vy<<1|1,mid+1,ry,y1,y
2);
    }

    ll query_x(ll vx,ll lx,ll rx,ll x1,ll x2,ll y1,ll y2){
        if (rx<x1||x2<lx) return 0;
        if (x1<=lx&&rx<=x2) return query_y(vx,1,0,m-1,y1,y2);
        ll mid=(lx+rx)>>1;
        return query_x(vx<<1,lx,mid,x1,x2,y1,y2)+query_x(vx<<1|1,mid+1,rx,x1,
x2,y1,y2);
    }
    void update(ll x,ll y,ll val){update_x(1,0,n-1,x,y,val);}
    ll query(ll x1,ll y1,ll x2,ll y2){return query_x(1,0,n-1,x1,x2,y1,y2);}
};
```

iterSEG

```
// usage: iterSEG(n, arr); update(tar, val); ll sum = query(qs, qe);
// memo : 0-indexed
#define OP(a,b) (max((a),(b)))
#define E LLONG_MIN
```

```
struct iterSEG {
    ll n;
    vl seg;

    iterSEG(ll size, const vl& arr){
        n=1; while (n<size) n<<=1;
        seg.assign(2*n, E);
        for (ll i=0;i<size;++i) seg[n+i]=arr[i];
        for (ll i=n-1;i>0;--i)
            seg[i] = OP(seg[i<<1], seg[i<<1|1]);
    }

    void update(ll idx, ll val) {
        ll p = idx + n;
        seg[p] = val;
        for (p>>=1;p;p>>=1)
            seg[p] = OP(seg[p<<1], seg[p<<1|1]);
    }

    ll query(int l, int r) {
        ll resL=E, resR=E;
        for (l+=n,r+=n; l<=r; l>>=1,r>>=1) {
            if (l & 1) resL = OP(resL, seg[l++]);
            if (!(r & 1)) resR = OP(seg[r--], resR);
        }
        return OP(resL, resR);
    }
};
```

iterLAZY

```
// usage: iterLAZY(n, arr); update(qs, qe, val); ll sum = query(qs, qe);
// memo : 0-indexed
#define OP(a,b)             ((a)+(b))
#define E                   0LL
#define MAPPING(x,f,len)    ((x) + (f)*(len))
#define COMPOSE(f_new,f_old)  ((f_new) + (f_old))
```

```
#define lzE                 0LL

struct iterLAZY {
    ll n, sz, h;
    vl seg, lz;

    iterLAZY(ll size, vl& arr){
        n=1; while (n<size) n<<=1;
        h=0; for (ll i=n;i>1;i>>=1) ++h;
        seg.assign(2*n, E);
        lz .assign(  n, lzE);
        copy(arr.begin(),arr.end(),seg.begin()+n);
        for (ll i=n-1;i;--i) seg[i]=seg[i<<1]+seg[i<<1|1];
    }

   void apply(ll p, ll f, ll len_unused){
       seg[p] = MAPPING(seg[p], f, len_unused);
       if (p < n) lz[p] = COMPOSE(f, lz[p]);
   }

    void pull(ll p) {
       for (ll i = 1; p > 1; i <<= 1, p >>= 1) {
          ll parent = p>>1;
          ll combined = OP(seg[p], seg[p^1]);
          ll added = MAPPING(E, lz[parent], i*2);
          seg[parent] = OP(combined, added);
       }
    }

    void push(ll p) {
        for (ll s=h, len=1<<(h-1); s; --s, len>>=1) {
            ll i=p>>s;
            if (!lz[i]) continue;
            apply(i<<1,   lz[i], len);
            apply(i<<1|1, lz[i], len);
            lz[i]=lzE;
        }
    }
    void update(ll l,ll r,ll v) {
        ll s=(l+=n), e=(r+=n), len=1;
        push(s); push(e);
        for (; l<=r; l>>=1, r>>=1, len<<=1) {
            if (l&1) apply(l++, v, len);
            if (!(r&1)) apply(r--, v, len);
        }
        pull(s); pull(e);
    }

    ll query(ll l,ll r) {
        l+=n; r+=n;
        push(l); push(r);
        ll Lsum=E, Rsum=E;
        while (l<=r) {
            if (l&1) Lsum = OP(Lsum, seg[l++]);
            if (!(r&1)) Rsum = OP(seg[r--], Rsum);
            l>>=1; r>>=1;
        }
        return OP(Lsum, Rsum);
    }
};
```

GoldenMine SEG

```
// usage: gmSEG(n, arr); update(tar, val); ll sum = query(qs, qe);
// memo : 0-indexed
struct node {
    ll l,r,res,total;
};
struct gmSEG {
    ll n;
    vl arr;
    vector<node> seg;

    gmSEG(ll n, vl& arr): n(n),arr(arr),seg(n<<2){
        init(1,0,n-1);
```

```
}

node merge(node& a,node& b) {
    if (a.total==LLONG_MIN) return b;
    if (b.total==LLONG_MIN) return a;
    node res;
    res.total = a.total + b.total;
    res.l = max(a.l, a.total + b.l);
    res.r = max(b.r, b.total + a.r);
    res.res = max({a.res, b.res, a.r + b.l});
    return res;
}

void init(ll node,ll l,ll r) {
    if (l==r) {
        seg[node]={arr[l],arr[l],arr[l],arr[l]};
        return;
    }
    ll m=(l+r)>>1;
    init(node<<1,l,m);
    init(node<<1|1,m+1,r);
    seg[node]=merge(seg[node<<1],seg[node<<1|1]);
}

node range_sum(ll node, ll l, ll r, ll s, ll e) {
    if (r<s||e<l) return {0,0,LLONG_MIN,LLONG_MIN};
    if (s<=l&&r<=e) return seg[node];
    ll m=(l+r)>>1;
    struct node left = range_sum(node<<1,l,m,s,e);
    struct node right = range_sum(node<<1|1,m+1,r,s,e);
    return merge(left,right);
}

void point_update(ll node, ll l, ll r, ll tar, ll val) {
    if (r<tar||tar<l) return;
    if (l==r) {
        seg[node]={val,val,val,val};
        return;
        }
        ll m=(l+r)>>1;
        point_update(node<<1,l,m,tar,val);
        point_update(node<<1|1,m+1,r,tar,val);
        seg[node]=merge(seg[node<<1],seg[node<<1|1]);
    }
    void update(ll tar,ll val){point_update(1,0,n-1,tar,val);}
    node query(ll s,ll e){return range_sum(1,0,n-1,s,e);}
};
```

2-SAT

```
// Usage: operation(lit(x), lit(y)); bool satisfiable = TwoSAT.solve()
// memo : 1-based variable
// O(N + M)  N:#variable M:#relation
inline ll neg(ll x) { return x%2 ? x+1 : x-1; }
inline ll lit(ll x) { return x>0 ? (x<<1)-1 : -x<<1; }
struct TwoSAT {
    ll N;                    // # boolean variables
    vvl g;                   // 1-based, size = 2*N+1
    vector<bool> assignment; // final truth values: +1=true, -1=false
    vl comp;

    TwoSAT(ll vars): N(vars), g((N << 1) + 1), assignment(N + 1, 0) {}

    void addImp(ll u, ll v) {
        g[u].push_back(v);
    }
    void addOr(ll x, ll y) {
        addImp(neg(x), y);
        addImp(neg(y), x);
    }
    void addTrue(ll x) {
        addImp(neg(x), x);
    }
    bool satisfiable() {
        SCC scc(g.size(), g);    // 0-index SCC
        comp = scc.scc_idx;
        for (ll i=1; i<=N; i++)
            if (comp[(i<<1)-1] == comp[i<<1])
                return false;
        return true;
    }
    bool solve(ll x) {
        return comp[(x<<1)-1] < comp[x<<1];
    }
    void solveAll() {
        for (ll i=1; i<=N; i++)
            assignment[i] = comp[(i<<1)-1]<comp[i<<1];
    }
};
```