Pontificia Universidad Católica del Perú - FCI

XieXieLucas Notebook - Froz/Phibrain/Ands

November 9, 2017

Contents

1 Fenwick Tree

```
// Fenwick tree: O(log(n)) accumulated sum queries.

ll bitadd[N] ;
ll bitsub[N] ;
int n ;

void update( int idx, ll val1, ll val2 ){
        while( idx <=n ) {
                bitadd[idx] += val1 ;
                bitsub[idx] += val2 ;
                idx += idx & -idx ;
        }
}

void updaterange( int l , int r , ll val ){
        update( l , val , (l-1)*val ) ;
        update( r+1 , -val , -r*val) ;
}

ll get( int idx ){
        ll add = 0 , sub = 0, aux = idx ;
        while ( idx > 0 ){
                add += bitadd[idx] ;
                sub += bitsub[idx] ;
                idx -= idx & -idx ;
        }
        return aux*add - sub ;
}
```

2 Heavy Light Decomposition

```
//Heavy-Light Decomposition Tree for Commutative Operations
//Phibrain

inline ll ma(ll a, ll b){return ((a-b>0)? a:b);}
inline ll mi(ll a, ll b){return ((a-b>0)? b:a);}

struct ST{
    ll n;
    ll t[2*N];
    ll Op(ll &u, ll &v){ return ma(u,v); }
    inline void build(){
        RREP(i,n-1,1) t[i]=Op(t[i<<1], t[i<<1|1]);
    }
    inline void modify(ll p, ll val){
        for(t[p+=n] = val ; p >>= 1;) t[p] = Op(t[p<<1], t[p<<1|1]);
    }
    inline ll que(ll l, ll r){
        ll ansl=min, ansr=min;
        for(l += n, r += n; l < r; l >>= 1, r >>= 1){
            if(l&1) ansl = Op(ansl, t[l++]);
            if(r&1) ansr = Op(t[--r], ansr);
        }
        return Op(ansl, ansr);
    }
};
struct HLDES{
    ll n;
    ST st;
    vi adj[N];
    ll p[N],d[N],tsz[N],id[N],rt[N];
    ll gid;
    inline ll Op(ll val1, ll val2) {return ma(val1,val2);}
    inline ll make1(ll u,ll par,ll depth){
        p[u]=par; d[u]=depth; tsz[u]=1;
        for(auto v:adj[u])if(v!=p[u]) tsz[u]+=make1(v,u,depth+1);
        return tsz[u];
    }
    inline void make(){
        ll val=make1(0,-1,0);
    }
    inline void dfs(ll u, ll root){
        id[u]=gid++; rt[u]=root;
        ll w=0 , wsz=min;
        for(auto v: adj[u]) if(v!=p[u]){
            if(tsz[v]>wsz) {w=v; wsz=tsz[v];}
        }
        if(w) dfs(w,root);
        for(auto v:adj[u]) if(v!=p[u]) if(v!=w) dfs(v,v);
    }
    inline void upd(ll u, ll val){
        ll a=id[u];
        st.modify(a,val);
    }
    ll que(ll u, ll v){
        ll ans=0;// neutro?
        while(u!=-1){
            if(rt[u]==rt[v]){
                ll a=id[u], b=id[v];
                if(a>b) swap(a,b);
                ans=Op(ans,st.que(a,b+1));
                u=-1;
            }
            else{
                if(d[rt[u]]>d[rt[v]]) swap(u,v);
                ans=Op(ans,st.que(id[rt[v]],id[v]+1));
                v=p[rt[v]];
            }
        }
        return ans;
    }
    inline void build(){
        gid=0; st.n=n;
        make(); dfs(0,0);
        REP(i,0,n) st.t[i+n]=0;//val de cada t[i]
        st.build();
    }
};


//Heavy Light Decomposition General

struct HLDES{
  ll n;
  ST st1,st2;
  vi adj[N];
  vector<ii> ver[N];
  ll p[N],d[N],tsz[N],id[N],rt[N],ar[N],val[N],id1[N];
  ll gid,k;

  inline T Op(T &val1, T &val2){
    T ty;
    //Operacion del Heavy Light
    return ty;
  }

  inline ll make1(ll u,ll par,ll depth){
    p[u]=par; d[u]=depth; tsz[u]=1;
    for(auto v:adj[u])if(v!=p[u]) tsz[u]+=make1(v,u,depth+1);
    return tsz[u];
  }
  inline void make(){
    ll val=make1(0,-1,0);
```

```
}
inline void dfs(ll u, ll root){
  ar[gid]=val[u];
  id[u]=gid++; rt[u]=root;
  ll w=0LL , wsz=min;
  for(auto v: adj[u]) if(v!=p[u]){
    if(tsz[v]>wsz) {w=v; wsz=tsz[v];}
  }
  if(w) dfs(w,root);
  for(auto v:adj[u]) if(v!=p[u]) if(v!=w) dfs(v,v);
}
inline void solve(){
  ll ta;
  REP(i,0,n) ver[rt[i]].pb(mp(id[i],i));
  REP(i,0,n){
    if(ver[i].size()!=0){
      sort(all(ver[i]));
      ta=ver[i].size();
      ta=ver[i][ta-1].fst;
      for(auto j: ver[i]) id1[j.snd]=ta--;
    }
  }
}
inline ll LCA(ll u, ll v){
  while(rt[u]!=rt[v]){
    if(d[rt[u]]<d[rt[v]]) v=p[rt[v]];
    else u=p[rt[u]];
  }
  return d[u]>d[v]? v:u;
}
inline void upd(ll u, ll v, ll val){
  ll l, r, a, b;
  //Update del Heavy Light
}
inline ll que(ll u, ll v){
  //Query del HLD
}
inline void build(){
  REP(i,0,n) cin>>val[i];
  REP(i,0,n-1) {
    ll a,b; cin>>a>>b;
    a--;b--;
    adj[a].pb(b); adj[b].pb(a);
  }
  gid=0LL; k=0LL; st1.n=n; //st2.n=n;//st.made();
    make();
    dfs(0,0);
    REP(i,0,n) st1.ar[i]=ar[i];
    st1.build();
  }
} hld;
```

3 LCA Tree

```
const int MAX = 1e4;
const int LGMAX = 15;
//LCA construction in O(n*log(n)) with O(log(n)) queries.
struct LCATree{
        int n;
        vector<int> adj[MAX];
        int p[MAX][LGMAX]; // 2^j ancestor of node i
        int L[MAX];        // Depth of node i
        int q[MAX];        // (Queue used internally).

        LCATree(int N):n(N){}

        void dfs(int u, int h){
            L[u] = h;
            REP(i,0,sz(adj[u])){
                int v = adj[u][i];
                if (v != p[u][0]) {
                    p[v][0] = u;
                    dfs(v, h+1);
                }
            }
        }
        void buildlca(int r){
                REP(i,0,n) REP(pw,0,LGMAX) p[i][pw] = -1;
            dfs(r, 0);
                for (int pw = 1; (1<<pw) < n; pw++){
                REP(i,0,n) if (p[i][pw-1] != -1) p[i][pw] = p[p[i][pw-1]][
                     pw-1];
                }
        }
        int lca(int u, int v){
                if (L[u] < L[v]) swap(u,v);
                for (int pw = LGMAX-1; pw >= 0; pw--)
```

```
                        if (L[u] - (1<<pw) >= L[v])
                                u = p[u][pw];
                if (u == v) return u;
                for (int pw = LGMAX-1; pw >= 0; pw--){
                        if (p[u][pw] != p[v][pw]) {
                                u = p[u][pw];
                                v = p[v][pw];
                        }
                }
                return p[u][0];
        }
};

int main() {
        int n = 1e3;
    LCATree T(n);
    //Initialize n and the adj[] list
        T.buildlca(0); //Place the root instead of 0
        //Ready to answer queries
        return 0;
}
```

4 Lazy Propagation Segment Tree

```
// lazy propagation con propagacion y el update
//ejemplo de update en [l,r> la serie de fibonaci con a y b como primeros
    numeros (f[1]=a,f[2]=b)
//notar la forma de updatepro y proh;
//made preprocess y find el fib de posicion n con a y b como primeros
    numeros

inline ll ss(ll val) {return val%MOD;}

ll dpf[N];

inline void made(){
  dpf[1]=1 ; dpf[2]=1;
  REP(i,3,N) dpf[i]=ss(dpf[i-1]+dpf[i-2]);
}
inline ll find(ll a, ll b, ll n) {
  if(n<3) return n==1? a:b;
  return ss(a*dpf[n-2]+b*dpf[n-1]);
}

struct ST{
  ii lazy[4*N];
  ll tree[4*N], ar[N];
  ll n;
  inline void updatepro(ii laz,ll id, ll l,ll r){
    ll ta=r-l, sum=(find(laz.fst,laz.snd,ta+2)-laz.snd+MOD)%MOD;
    tree[id]=ss(tree[id]+sum);
    lazy[id].fst=ss(lazy[id].fst+laz.fst);
    lazy[id].snd=ss(lazy[id].snd+laz.snd);
  }
  inline void proh(ll id, ll l,ll r){
    ll mid=(l+r)>>1, ta=mid-l;
    updatepro(lazy[id],2*id,l,mid);
    ii laz;
    laz.fst=find(lazy[id].fst,lazy[id].snd,ta+1);
    laz.snd=find(lazy[id].fst,lazy[id].snd,ta+2);
    updatepro(laz,2*id+1,mid,r);
    lazy[id]={0LL,0LL};
  }
  inline void updateRange(ll x, ll y, ll a, ll b, ll id, ll l,ll r){
    if(x>=r || y<=l) return;
    if(x<=l && r<=y){
      ll ta=l-x; ii laz;
      laz.fst=find(a,b,ta+1); laz.snd=find(a,b,ta+2);
      updatepro(laz,id,l,r);
      return;
    }
    proh(id,l,r);ll mid=(l+r)>>1;
    updateRange(x,y,a,b,2*id,l,mid);
    updateRange(x,y,a,b,2*id+1,mid,r);
    tree[id]=ss(tree[2*id]+tree[2*id+1]);
  }
  inline ll getSum(ll x,ll y,ll id,ll l,ll r){
    if(x>=r || l>=y) return 0;
    if(x<=l && r<=y) return tree[id];
    proh(id,l,r);ll mid=(l+r)>>1;
    ll ez,ez1,ez2;
    ez1=getSum(x,y,2*id,l,mid);
    ez2=getSum(x,y,2*id+1,mid,r);ez=ss(ez1+ez2);
    return ez;
  }
  inline void build1( ll id, ll l, ll r){
    if (l > r) return ;
    if (r-l<2){tree[id] = ar[l];return;}
    ll mid = (l + r)>>1;
    build1(2*id, l,mid); build1(2*id+1, mid, r);
    tree[id] = ss(tree[id*2 ] + tree[id*2 + 1]);
  }
  inline void upd(ll x, ll y, ll a, ll b){
    updateRange(x,y,a,b,1,0,n);
  }
  inline void build(){
    build1(1,0,n);
  }
  inline ll que(ll x, ll y){
    return getSum(x,y,1,0,n);
  }
};
```

5 Link Cut Tree

```
//Link cut tree

const int N = 1e5 + 2;

struct Node {
    Node *left, *right, *parent;
    bool revert;
    Node() : left(0), right(0), parent(0), revert(false) {}
    bool isRoot() {
        return parent == NULL ||
            (parent->left != this && parent->right != this);
    }
    void push() {
        if (revert) {
            revert = false;
            Node *t = left;
            left = right;
            right = t;
            if (left != NULL) left->revert = !left->revert;
            if (right != NULL) right->revert = !right->revert;
        }
    }
};
struct LinkCutTree{
    Node nos[N];

    LinkCutTree(){
        REP(i,0,N) nos[i] = Node();
    }

    void connect(Node *ch, Node *p, bool isLeftChild) {
        if (ch != NULL) ch->parent = p;
        if (isLeftChild) p->left = ch;
        else p->right = ch;
    }

    void rotate(Node *x){
        Node* p = x->parent;
        Node* g = p->parent;
        bool isRoot = p->isRoot();
        bool leftChild = x == p->left;

        connect(leftChild ? x->right : x->left, p, leftChild);
        connect(p, x, !leftChild);
        if (!isRoot) connect(x, g, p == g->left);
        else x->parent = g;
    }

    void splay(Node *x){
        while (!x->isRoot()) {
            Node *p = x->parent;
            Node *g = p->parent;
            if (!p->isRoot()) g->push();
            p->push();
            x->push();
            if (!p->isRoot()) {
                rotate((x == p->left) == (p == g->left) ? p : x);
            }
            rotate(x);
        }
        x->push();
    }

    Node *expose(Node *x) {
        Node *last = NULL, *y;
        for (y = x; y != NULL; y = y->parent) {
            splay(y);
            y->left = last;
            last = y;
        }
        splay(x);
        return last;
    }

    void makeRoot(Node *x) {
        expose(x);
        x->revert = !x->revert;
    }

    bool connected(Node *x, Node *y) {
        if (x == y) return true;
        expose(x);
        expose(y);
        return x->parent != NULL;
    }

    bool link(Node *x, Node *y) {
        if (connected(x, y)) return false;
        makeRoot(x);
        x->parent = y;
        return true;
    }

    bool cut(Node *x, Node *y) {
        makeRoot(x);
        expose(y);
        if (y->right != x || x->left != NULL || x->right != NULL)
            return false;
        y->right->parent = NULL;
        y->right = NULL;
        return true;
    }
};
```

6 Persistent Segment Tree

```
// Persistent segment tree implemented with pointers.
// Consider using a map<int, node*> which represents
// the segment tree at time t.
const int MAX = 1e6;
typedef int T;
T arr[MAX];
struct node {
        T val;
        node *l, *r;
        node(T val) : val(val), l(NULL), r(NULL) {}
        node(T val, node* l, node* r) : val(val), l(l), r(r) {}
};
// Identity element of Op()
const T OpId = 0;
// Associative query operation
T Op(T val1, T val2){
        return val1 + val2;
}
node* build(int a, int b) {
        if (a+1 == b) return new node(arr[a]);
        node* l = build(a, (a+b)/2);
        node* r = build((a+b)/2, b);
        return new node(Op(l->val, r->val), l, r);
}
// Branch and increment position p by val
node* update(node* u, int a, int b, int p, T val) {
        if (a > p || b <= p) return u;
        if (a+1 == b) return new node(Op(u->val, val));
        node* l = update(u->l, a, (a+b)/2, p, val);
        node* r = update(u->r, (a+b)/2, b, p, val);
        return new node(Op(l->val, r->val), l, r);
}
// Query t to get sum of values in range [i, j)
T query(node* u, int a, int b, int i, int j) {
        if (a >= j || b <= i) return OpId;
        if (a >= i && b <= j) return u->val;
        T q1 = query(u->l, a, (a+b)/2, i, j);
        T q2 = query(u->r, (a+b)/2, b, i, j);
        return Op(q1, q2);
}
map<int, node*> m;
node* st;
T val;
int n, p;
int main() {
        REP(i,0,n) arr[i] = 0; // Any starting values
        m.clear();
        st = build(0,n);
        m[0] = st;
```

```
       REP(i,0,n){
              // Modify position p with value val at time t
              st = update(st, 0, n, p, val);
              m[i] = st;
       }
       // Consider for example rectangular queries:
       // Sum of all nodes in [a,b]x[c,d] using one
       // coordinate as time and another as values
}
```

7 Segment Tree

```
// Iterative, fast, non-conmutative segment tree.
typedef int T;
const int MAX = 1e6;

// Identity element of the operation
const T OpId = 0;
// Associative internal operation
T Op(T& val1, T& val2){
    return val1 + val2;
}

// The user should fill t[n, 2*n)
T t[2*MAX];
int n;

void build(){
    for( int i = n-1 ; i > 0 ; i-- ) t[i] = Op(t[i<<1], t[i<<1|1]);
}

void modify( int p , T val ){
    for( t[p+=n] = val ; p >>= 1 ; ) t[p] = Op(t[p<<1], t[p<<1|1]);
}

T get( int l , int r ){ //[l,r)
    T ansl, ansr;
    ansl = ansr = OpId; //Initialize operation at Identity
    for( l += n, r += n ; l < r ; l >>= 1, r >>= 1 ){
           if(l&1) ansl = Op(ansl, t[l++]);
           if(r&1) ansr = Op(t[--r], ansr);
    }
    return Op(ansl, ansr);
}


int main(){
       // Read into t[n,2*n)
    build();
    // Answer queries
}
```

8 Wavelet Tree

```
/*
  Wavelet Tree Implementation
  Construction in O(nlogn)
  Queries in O(log(MAX))

  1 - based array!
*/



typedef vector<int> vi;

struct WT{
  int lo, hi;
  WT *l, *r; vi b;
  WT(int *from, int *to, int x, int y){
    lo = x, hi = y;
    if(lo == hi or from >= to) return;
    int mid = (lo+hi)/2;
    auto f = [mid](int x){
      return x <= mid;
    };
    b.reserve(to-from+1);
    b.pb(0);
    for(auto it = from; it != to; it++) b.pb(b.back() + f(*it));
    auto pivot = stable_partition(from, to, f);
    l = new WT(from, pivot, lo, mid);
    r = new WT(pivot, to, mid+1, hi);
  }
  //kth en [l,r]
  int kth(int l, int r, int k){
    if(l > r) return 0;
    if(lo == hi) return lo;
    int inLeft = b[r] - b[l-1]; //cantidad en los a primeros b[a]
    int lb = b[l-1];
    int rb = b[r];
    if(k <= inLeft) return this->l->kth(lb+1, rb , k);
    return this->r->kth(l-lb, r-rb, k-inLeft);
  }

  //cantidad de numeros menoes a K en [l,r]
  int LTE(int l, int r, int k) {
    if(l > r or k < lo) return 0;
    if(hi <= k) return r - l + 1;
    int lb = b[l-1], rb = b[r];
    return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb, r-rb, k);
  }

  //cantidad de numeros en [l,r] iguales a k
  int count(int l, int r, int k) {
    if(l > r or k < lo or k > hi) return 0;
    if(lo == hi) return r - l + 1;
    int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
    if(k <= mid) return this->l->count(lb+1, rb, k);
    return this->r->count(l-lb, r-rb, k);
  }
  ~WT(){
    delete l;
    delete r;
  }
};
```