East West University

Intra University Programming Camp

Variant Implementations

Of

Segment Tree

Prepared by: Abdullah Al Masud Tushar

rctushar07@gmail.com

```
int tree[M*3];
int prop[M*3];

// প্রায় সব সেগমেন্ট ট্রি প্রবলেমের ইনিট ফাংশনের স্ট্রাকচার এইরকম ।
// এই ফাংশনের কাজ হল শুরুতেই 1D এরে তে সেভ হওয়া ভ্যালু  গুলি ট্রি এর লিফ নোডে সেভ করা

void init(int nod,int b,int e)
{
    if(b==e)
    {
        tree[nod]=ara[b]; // ইনপুট এরে ট্রি এর লিফ নোডে সেট হচ্ছে
        return;
    }

    int left=nod*2;       // বর্তমান নোডের  বাম নোড
    int right=nod*2+1;    // বর্তমান নোডের ডান নোড
    int mid=(b+e)/2;      // বর্তমান নোডের রেঞ্জকে দুই ভাগ করলে মাঝের
                          // যে পয়েন্ট পাওয়া যাবে  ( b to mid | mid+1 to e)

    init(left,b,mid);     // বাম পাশের নোড কল হচ্ছে  begin থেকে mid রেঞ্জ পর্যন্ত
    init(right,mid+1,e);  // ডান পাশের নোড কল হচ্ছে  mid+1 থেকে end রেঞ্জ পর্যন্ত

    tree[nod]=tree[left]+tree[right]; // বর্তমান নোডে সেভ হচ্ছে নিচের লেফট-রাইট
                                      // চাইল্ড নোডের মারজ করা ভ্যালু
}
```

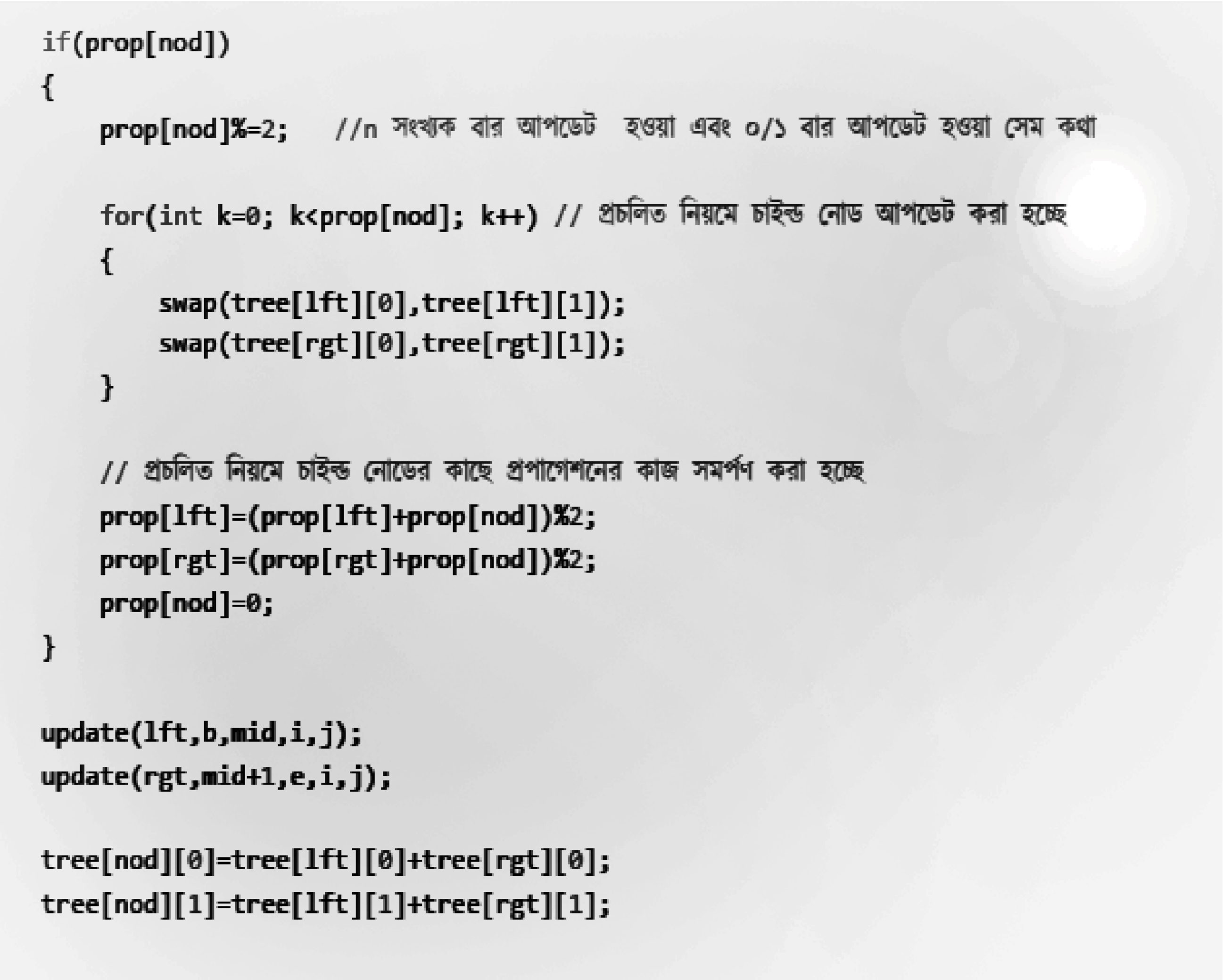
```
void update(int nod,int b,int e,int i,int j,int val)
{
    if(i>e||j<b)           // গিভেন রেঞ্জ ট্রি নোডের রেঞ্জের বাইরে চলে গেলে ঐ রেঞ্জ বাতিল হবে
        return;

    if(b>=i&&e<=j)         // গিভেন রেঞ্জের কোন অংশ ট্রি নোডের রেঞ্জের ভেতর এঁটে গেলে
    {
        tree[nod]+=(e-b+1)*val; // বর্তমান নোডের ভেতর যতগুলি ইলেমেন্ট আছে   তাদের
                                // সবার সাথে val যোগ করার পর টোটাল মান সেভ হবে।
        prop[nod]+=val;         //এই এরেতে সেভ হবে নিচের নোডগুলোতে আগামীতে কত
                                // ভ্যালু দিয়ে আপডেট করার দরকার আছে।
        return;
    }

    int left=nod*2;
    int right=nod*2+1;
    int mid=(b+e)/2;

    // বর্তমান নোডের প্রপাগেশন এরে যদি শুন্য না হয় তাহলে নিচের  নোডগুলোতে
    // ১->  ডান - বাম ট্রি নোডে  বর্তমান নোডের প্রপাগেশন ভ্যালু দিয়ে আপডেট হবে
    // ২-> ডান - বাম নোডের প্রপাগেশন এরেতে বর্তমান নোডের প্রপাগেশন ভ্যালু পাঠিয়ে দিবে
    // এরপর নিজে খালি হয়ে যাবে। কারণ বর্তমান নোডের এখন পর্যন্ত যেসব ভ্যালু যোগ করার কুএরি
    // করা হয়েছে সেসব কাজ বর্তমান নোডের প্রপাগেশন ভ্যালু এখন কমপ্লিট করে ফেলেছে। একই কাজ
    // সম্পন্ন করার জন্য সে তখন চাইল্ড নোডে সেসব কাজ সমর্পণ করে দেয়।

    if(prop[nod])
    {
        tree[left]+=(mid-b+1)*prop[nod];
        tree[right]+=(e-mid)*prop[nod];
        prop[left]+=prop[nod];
        prop[right]+=prop[nod];
        prop[nod]=0;
    }

        update(left,b,mid,i,j,val);
        update(right,mid+1,e,i,j,val);
        tree[nod]=tree[left]+tree[right];
    }
```

```
// এই ফানশনের কাজ কোন একটা গিভেন রেঞ্জ (i to j)এর টোটাল যোগফল পাওয়া ।
// একটি ইনডেক্স এর মান জানার ক্ষেত্রে ফাংশন কলের সময়  প্যারামিটার হিসাবে i,j তে
// একই ভ্যালু পাঠাতে হবে।
// কম বেশি প্রায় সব লেজি প্রোপাগেশনের কুএরি ফাংশনের ইমপ্লিম্যান্টেশন স্ট্রাকচার এইরকম ।
// প্রবলেম কন্সট্রেইন্ট এর উপর ভিত্তি করে ভেতরের অংশের মডিফিকেশন করতে হয়


int qury(int nod,int b,int e,int i,int j)
{
    if(i>e||j<b)   // রেঞ্জ বাতিল হবার কারনে শুন্য রিটার্ন করবে
        return 0;

    if(b>=i&&e<=j)
    return tree[nod]; // রেঞ্জের ভেতর এঁটে গেলে  ঐ অংশের মান রিটার্ন করবে

    int left=nod*2;
    int right=nod*2+1;
    int mid=(b+e)/2;

    if(prop[nod])      // এই অংশ আপডেট এবং কুএরি ফাংশনের ক্ষেত্রে একই থাকবে
    {
        tree[left]+=(mid-b+1)*prop[nod];
        tree[right]+=(e-mid)*prop[nod];
        prop[left]+=prop[nod];
        prop[right]+=prop[nod];
        prop[nod]=0;
    }

    int p1=qury(left,b,mid,i,j);
    int p2=qury(right,mid+1,e,i,j);
    return p1+p2;
}
```

```
// এই ফাংশনের কাজ  ট্রি এর বর্তমান অবস্থায় i তম ইলেমেন্ট খুঁজে বের করা,
// ডিলিট করা এবং ডিলিট হওয়া এরের আইডি নাম্বার শো করা

void dilit(int nod,int b,int e,int i)
{
    if(b==e)
    {
        tree[nod]=0;          // ভ্যালু ১ থেকে ০ করার মানে হল
                              // এই নোডের ইলেমেন্ট ডিলিট হয়েছে
        pf("%d\n",tid[nod]);
        return;
    }

    int left=nod*2;
    int right=nod*2+1;
    int mid=(b+e)/2;

    //যদি লেফট ট্রি এর টোটাল ইলেমেন্ট i তম ইলেমেন্টের সমান অথবা বড় হয়
    // তাহলে বলা যায় i তম ইলেমেন্ট লেফট  ট্রি এর কোন এক জায়গায় বিদ্যমান

    //আর সেটা যদি না হয় তাহলে  অবশ্যই সেটা রাইট ট্রি এর কোন এক জায়গায় বিদ্যমান।
    // এক্ষেত্রে যেহেতু লেফট ট্রি এর সম্পূর্ণ অংশ বাদ দিচ্ছি তাই i থেকে  tree[left]
    // এর টোটাল ইলেমেন্ট বাদ দিয়ে যে নতুন পজিশন i পাওয়া যাবে সেটা খুঁজে বেড়াব আগের নিয়মে


    if(tree[left]>=i)
        dilit(left,b,mid,i);
    else
        dilit(right,mid+1,e,i-tree[left]);

    tree[nod]=tree[left]+tree[right];
}
```

```
// এই অংশটা চিন্তা সামান্য জটিল তবে বেশ মজার।
// দুইটা কুএরি থাকবে, একটা হল i - j রেঞ্জের ভেতর ইলেমেন্টগুলোর
// মান এক করে বাড়াবে এবং আরেকটা হল i-j রেঞ্জের ভেতর কতোগুলো নাম্বার
// ২ দ্বারা বিভাজ্য যায় তা শো করবে

// আইডিয়াঃ এই ধরনের প্রবলেম লেজি প্রপাগেশন দিয়ে সল্ভ করা যায় তবে এক্ষেত্রে
// ট্রি এর ভেতর আলাদা ভাবে ইনফরমেশন রাখতে হবে। ২ দিয়ে বিভাজ্য হবার  মানে হল
// কোন একটা নির্দিষ্ট রেঞ্জের ভেতর ইলেমেন্টগুলোকে মড করলে সেটা ০ অথবা ১ হবে।
// শুন্য হলে সেটা অবশ্যই দুই দিয়ে বিভাজ্য, নতুবা নয়। তাহলে আমাদের প্রবলেমটায় এখন
// মূল কাজ হবে প্রতিটা নোডে কতোগুলো ০ এবং কতগুলো ১ আছে সেটার ইনফরম্যাশন সেভ রাখা।
// আর কোন রেঞ্জের ভেতর একবার আপডেট হলে ঐ নোডের tree[][0] এবং  tree[][1]
// এর ভ্যালু গুলো অদল বদল হবে। কারণ , ঐ রেঞ্জের ইলেমেন্টগুলোকে আপডেট দেয়ার ফলে
// এতক্ষণ ২ দিয়ে মড করার পর যাদের মান ০ হত তাদের মান  ১ পাওয়া যাবে এবং  যাদের ১ পাওয়া যেতো
// তাদের পাওয়া যাবে ০ , সহজ কথায় সুয়াপ হবে। নিচের  ফাংশনগুলোতে সেটাই করা হয়েছে।

int tree[M*3][3];
int prop[M*3];

// এই ধরনের প্রবলেমের ক্ষেত্রে ইনিট ফাংশন স্ট্রাকচার নিচের মতো হবে।
void init(int nod,int b,int e)
{
    if(b==e)
    {
        tree[nod][0]=1; // শুরুতে সবগুলো নাম্বার ০ তাই লিফ নোডে (x%2)=০ আছে ১ টা করে
        tree[nod][1]=0;
        prop[nod]=0;
        return;
    }

    int lft,rgt,mid;

    lft=2*nod;
    rgt=2*nod+1;
```

```
    mid=(b+e)/2;

    init(lft,b,mid);
    init(rgt,mid+1,e);

    tree[nod][0]=tree[lft][0]+tree[rgt][0]; //  প্রতি  নোডে চাইল্ড নোড মারজ হবে
    tree[nod][1]=tree[lft][1]+tree[rgt][1]; // ''  ''  ''  ''  '' ''
    prop[nod]=0;



/ এই ফাংশনে কাজ লেজি প্রপাগেশন প্রসেসে আপডেট করা
oid update(int nod,int b,int e,int i,int j)

    if(i>e||j<b)
        return;
    if(b>=i&&e<=j)
    {
        swap(tree[nod][0],tree[nod][1]); // এই রেঞ্জের ০,১ এর ভ্যালু অদল বদল হয়েছে
        prop[nod]++;  // প্রপাগেশন ভ্যালু  বাড়িয়ে বলা হচ্ছে নিচের নোডে কতবার আপডেট করা লাগবে
        return;
    }

    int lft,rgt,mid;

    lft=2*nod;
    rgt=2*nod+1;
    mid=(b+e)/2;
```



```
if(prop[nod])
{
    prop[nod]%=2;   //n সংখ্যক বার আপডেট  হওয়া এবং ০/১ বার আপডেট হওয়া সেম কথা

    for(int k=0; k<prop[nod]; k++) // প্রচলিত নিয়মে চাইল্ড নোড আপডেট করা হচ্ছে
    {
        swap(tree[lft][0],tree[lft][1]);
        swap(tree[rgt][0],tree[rgt][1]);
    }

    // প্রচলিত নিয়মে চাইল্ড নোডের কাছে প্রপাগেশনের কাজ সমর্পণ করা হচ্ছে
    prop[lft]=(prop[lft]+prop[nod])%2;
    prop[rgt]=(prop[rgt]+prop[nod])%2;
    prop[nod]=0;
}

update(lft,b,mid,i,j);
update(rgt,mid+1,e,i,j);

tree[nod][0]=tree[lft][0]+tree[rgt][0];
tree[nod][1]=tree[lft][1]+tree[rgt][1];
```

```
// উপরের আপডেট ফাংশন এবং এই ফাংশনের স্কেলিটন প্রায় সেম।
// মূল পার্থক্য হল এই ফাংশন আপডেট অংশটাতে সরাসরি ভ্যালু রিটার্ন করে
int qury(int nod,int b,int e,int i,int j)
{
    if(i>e||j<b)
        return 0;
    if(b>=i&&e<=j)
    return tree[nod][0];

    int lft,rgt,mid;

    lft=2*nod;
    rgt=2*nod+1;
    mid=(b+e)/2;

    if(prop[nod])
    {
        prop[nod]%=2;
        for(int k=0; k<prop[nod]; k++)
        {
            swap(tree[lft][0],tree[lft][1]);
            swap(tree[rgt][0],tree[rgt][1]);
        }
        prop[lft]=(prop[lft]+prop[nod])%2;
        prop[rgt]=(prop[rgt]+prop[nod])%2;
        prop[nod]=0;
    }

    int p1=qury(lft,b,mid,i,j);
    int p2=qury(rgt,mid+1,e,i,j);

    return p1+p2;
}
```

Practice problem

Single node update – 1082, 1093, 1112

Lazy propagation – 1164, 1080, 1183

Insert/delete/kth value – 1087, 1097

Array compression/offline – 1089

Offline - 1207

Compound node - 1135 , codechef (ENTRY)