ডাটা স্ট্রাকচার: সেগমেন্ট ট্রি-২ (লেজি প্রপাগেশন)

জুলাই ১৮, ২০১৩ by *শাফায়েত*

সেগমেন্ট ট্রির সবথেকে এলিগেন্ট অংশ হলো লেজি প্রপাগেশন টেকনিক। আমরা আগের পর্বে যে সেগমেন্ট ট্রি যেভাবে আপডেট করেছি তাতে একটা বড় সমস্যা ছিলো। আমরা একটা নির্দিষ্ট ইনডেক্স আপডেট করতে পেরেছি, কিন্তু একটা রেঞ্জের মধ্যে সবগুলো ইনডেক্স আপডেট করতে গেলেই বিপদে পরে যাবো। সে কারণেই আমাদের লেজি প্রপাগেশন শিখতে হবে, প্রায় সব সেগমেন্ট ট্রি প্রবলেমেই এই টেকনিকটা কাজে লাগবে। এই পর্বটা পড়ার আগে অবশ্যই তোমাকে সেগমেন্ট ট্রির একদুটি প্রবলেম সলভ করে আসতে হবে, এছাড়া তোমার বুঝতে সমস্যা হবে। আগের পর্বে লেজি প্রপাগেশন দরকার হয়না এমন কয়েকটি প্রবলেম দিয়েছি, আগে সেগুলো সলভ করতে হবে।

তোমাকে একটি অ্যারে দেয়া আছে n সাইজের এবং তোমাকে কুয়েরি করা হচ্ছে **i থেকে j ইনডেক্সের মধ্যে সবগুলো এলিমেন্টের যোগফল বলতে হবে**। আর আপডেট অপারেশনে তোমাকে বলা হলো **i থেকে j ইনডেক্সের মধ্যে সবগুলো সংখ্যার সাথে একটি নির্দিষ্ট সংখ্যা x যোগ করতে**।

যেমন যদি অ্যারেটা শুরুতে হয়তো ছিলো এরকম:

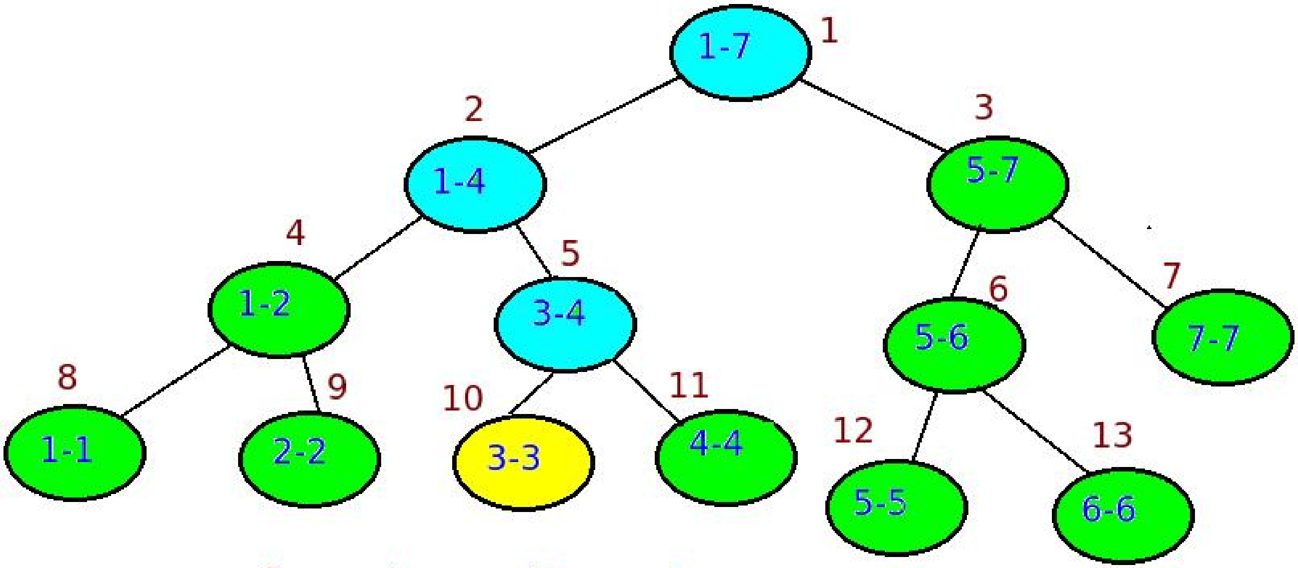
> *4 1 2 3 9 8 7*

তাহলে i=3,j=5 ইনডেক্সের মধ্যে সবগুলো সংখ্যার সাথে ২ যোগ করলে অ্যারেটা হবে:

4 1 2+2 3+2 9+2 8 7

আবার i=3,j=4 ইনডেক্সের মধ্যে সবগুলো সংখ্যার যোগফল হবে ২+২+৩+২=৯।

> আগের প্রবলেমে আমরা খালি একটা ইনডেক্স আপডেট করেছিলাম। তখন আমি শুধু ৩ নম্বর ইনডেক্সে আপডেট দেখানোর জন্য এই ছবিটা একেছিলাম:



যে নোডটি আপডেট করবো সেই নোডে পৌছানোর পথের সবগুলো নোড আপডেট হয়ে যাবে

আমরা সেগমেন্ট ট্রি এর একদম নিচে গিয়ে ৩ যেখানে আছে সেই নোডটা আপডেট করেছি এবং সেই পথের বাকিনোডগুলোকেও আপডেট করে দিয়েছি উপরে ওঠার সময়।

এখন তোমাকে **i=১ এবং j=৪** এই রেঞ্জের সবার সাথে **x** যোগ করতে বলা হলো। একটা সলিউশন হতে পারে তুমি আগের মতো করেই ১,২,৩,৪ ইত্যাদি ইনডেক্স আলাদা করে আপডেট করলে। তাহলে প্রতি আপডেটে কমপ্লেক্সিটি logn এবং সর্বোচ্চ n টি আপডেটের জন্য nlogn সময় লাগবে। এটা খুব একটা ভালো সলিউশন না, আমরা এখানে logn এই আপডেট করতে পারি।

কিছু ডেফিনেশন:

যেকোনো ট্রি স্ট্রাকচারে লিফ নোড হলো সবথেকে নিচের নোডগুলো। লিফ ছাড়া বাকি সবনোড হলো ইন্টারনাল নোড।

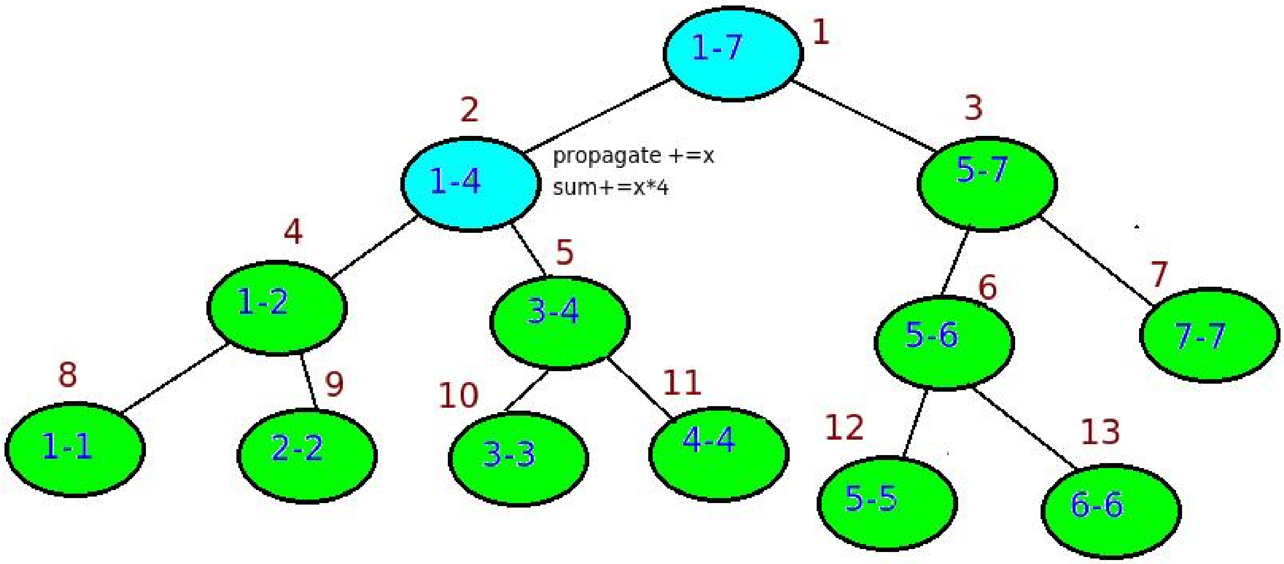
সেগমেন্ট ট্রি তে লিফ নোড গুলোতে কি থাকে? সেখানে থাকে কোনো একটি ইনডেক্সের অরিজিনাল ভ্যালু।

সেগমেন্ট ট্রি তে ইন্টারনাল(লিফ ছাড়া বাকি সব) নোডগুলোতে কি থাকে? সেখানে থাকে নিচে যতগুলো লিফ নোড আছে সবগুলোর মার্জ করা ফলাফল।

সেগমেন্ট ট্রি তে একটা নোডের রেঞ্জ হলো সেই নোডে যেসব ইনডেক্সের মার্জ করা রেজাল্ট আছে। যেমন ছবিতে ৩ নম্বর নোডের রেঞ্জ ৫ থেকে ৭।

যেমন উপরের ছবিতে ১০ নম্বর নোডে আছে ৩ নম্বর ইনডেক্সের ভ্যালু আর ২ নম্বর নোডে আছে ১,২,৩,৪ নম্বর নোডের যোগফল। কি দরকার এতগুলো লিফ নোডে কষ্ট করে গিয়ে আপডেট করে আসা? তার থেকে আমরা কি ২ নম্বর নোডে এসে সেখানে এই ইনফরমেশনটা রেখে দিতে পারিনা “কারেন্ট নোডের নিচের সবগুলো ইনডেক্সের সাথে x যোগ হবে” ? তার মানে যখন দেখছি **একটা নোডের রেঞ্জ যখন পুরোপুরি কুয়েরির ভিতরে থাকে তখন সেই নোডের নিচে আর না গিয়ে সেখানে প্রপাগেশন ভ্যালুটা সেভ করে রাখতে পারি। একটা নোডের প্রপাগেশন ভ্যালু হলো সেই নোডের রেঞ্জের মধ্যে সব ইনডেক্সের মধ্যে যে ভ্যালুটা যোগ হবে সেটা।**

নিচের ছবিতে দেখো কিভাবে এই এক্সট্রা ইনফরমেশনটা সেভ করা হচ্ছে:



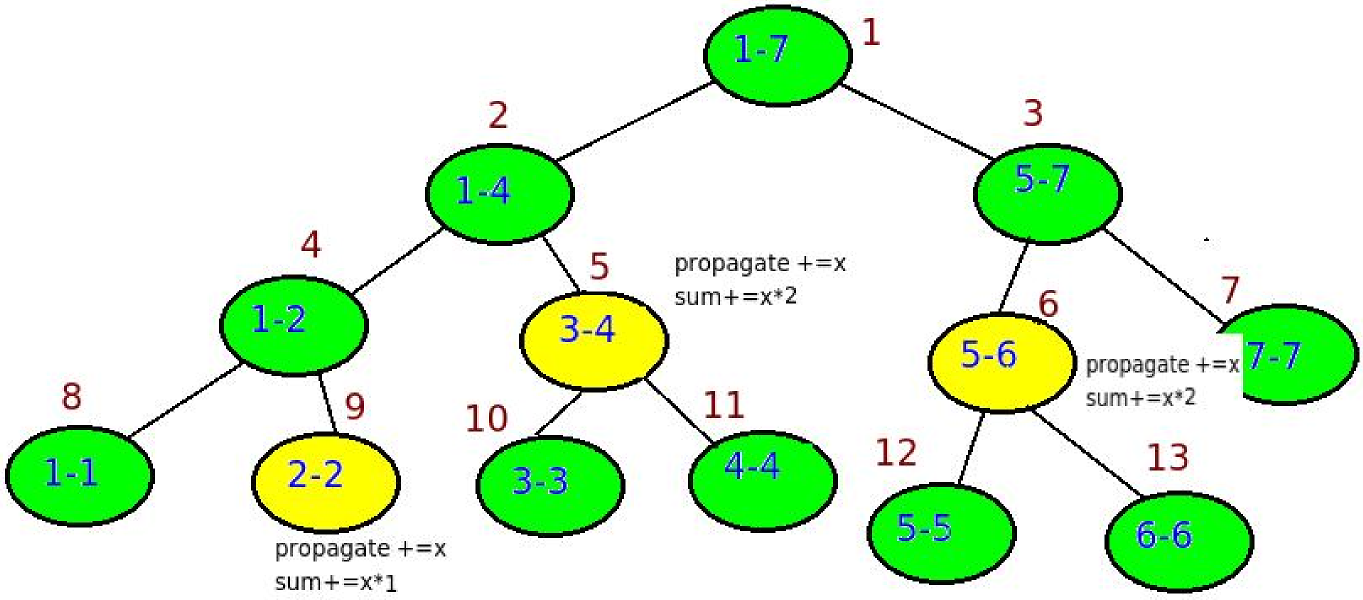
১ থেকে ৪ ইনডেক্সের সবার সাথে x যোগ করতে চাই

২ নম্বর নোডে গিয়ে বলে দিলাম নিচের সবার সাথে x যোগ হবে

২ নম্বর নোডের নিচে ৪ টি লিফ নোড আছে, সবার সাথে x যোগ করলে ২ নম্বর নোডের সাম এর সাথে 4x যোগ হবে।

প্রতিটি নোডে যোগফল ছাড়াও আরেকটা ভ্যারিয়েবল রাখতে হবে যেটার নাম দিয়েছি propagate। এই ভ্যারিয়েবলটার কাজ হলো তার নিচের লিফ নোডগুলোর সাথে কত যোগ করতে হবে তার হিসাব রাখা। propagate এর মান শুরুতে থাকে শূন্য। এরপর যে রেঞ্জটা আপডেট করতে বলবে সেই রেঞ্জের “রিলেভেন্ট নোড” গুলোতে গিয়ে propagate এর সাথে x যোগ করে আসবো। (আগের পর্বেই জেনেছো “রিলেভেন্ট নোড” হলো যেসব নোডের রেঞ্জ পরোপুরি কুয়েরির ভিতরে আছে)

আরেকটা উদাহরণ, ২ থেকে ৬ নম্বর নোড যদি আপডেট করতে হয় তাহলে হলুদ নোডগুলোতে গিয়ে বলে দিবো নিচের নোডগুলোর সাথে x যোগ করতে:

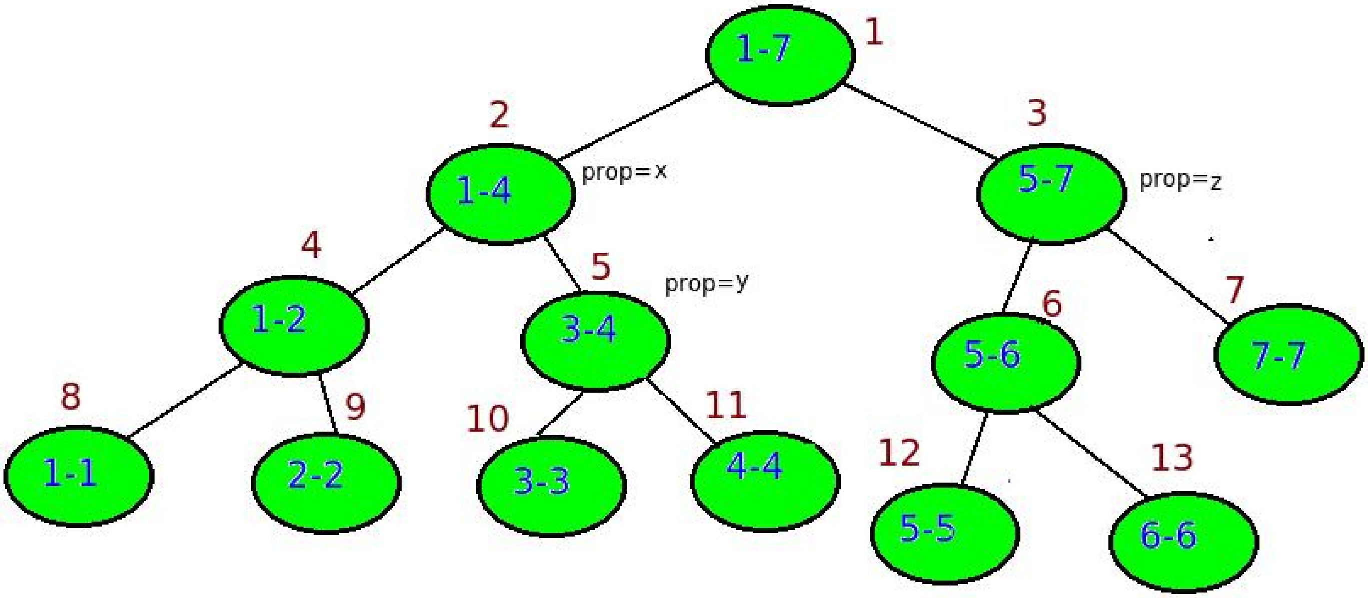


আগের আপডেট ফাংশনের মতোই কোনো একটা নোড আপডেট করার পর সেই পথের সবগুলো নোড আপডেট করে উঠতে হবে। আমরা কোডটা দেখি:

```
struct info {
    i64 prop, sum;
} tree[mx * 3]; //sum ছাড়াও নিচে অতিরিক্ত কত যোগ হচ্ছে সেটা রাখবো prop এ
void update(int node, int b, int e, int i, int j, i64 x)
{
    if (i > e || j < b)
        return;
    if (b >= i && e <= j) //নোডের রেঞ্জ আপডেটের রেঞ্জের ভিতরে
    {
        tree[node].sum += ((e - b + 1) * x); //নিচে নোড আছে e-b+1 টি, তাই e-b+1 বার x যোগ হবে এই
        tree[node].prop += x; //নিচের নোডগুলোর সাথে x যোগ হবে
        return;
    }
    int Left = node * 2;
    int Right = (node * 2) + 1;
    int mid = (b + e) / 2;
    update(Left, b, mid, i, j, x);
    update(Right, mid + 1, e, i, j, x);
    tree[node].sum = tree[Left].sum + tree[Right].sum + (e - b + 1) * tree[node].prop;
    //উপরে উঠার সময় পথের নোডগুলো আপডেট হবে
    //বাম আর ডান পাশের সাম ছাড়াও যোগ হবে নিচে অতিরিক্ত যোগ হওয়া মান
}
```

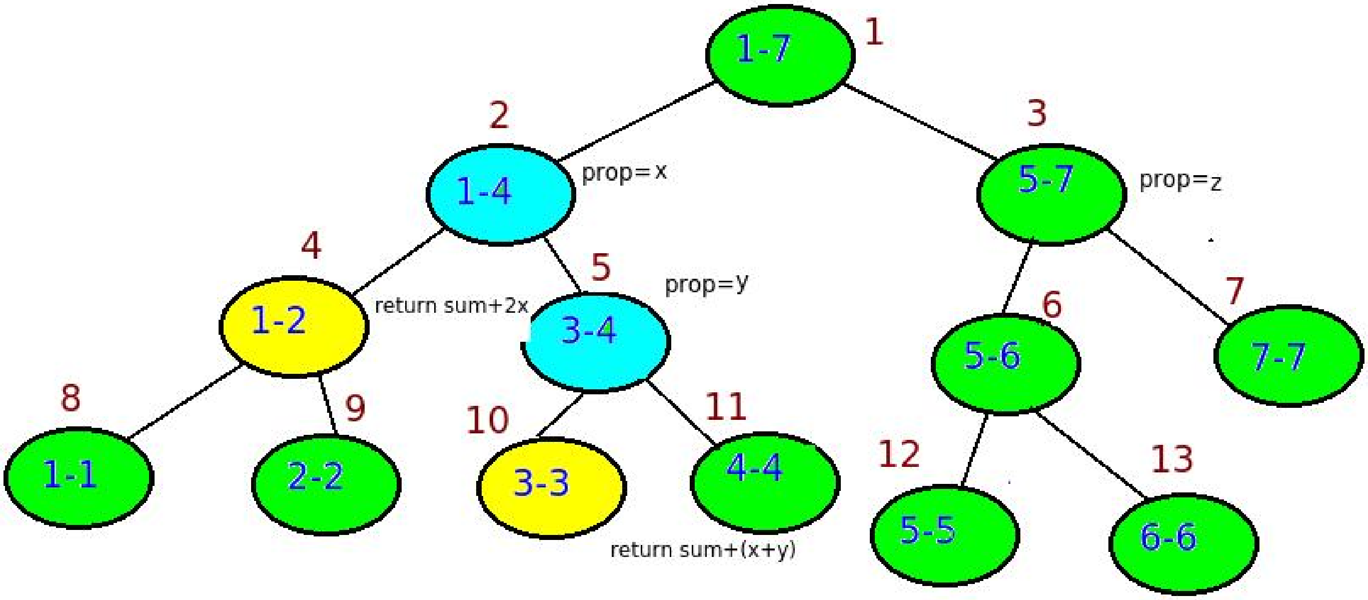
আগের আপডেট ফাংশনের সাথে পার্থক্য হলো এখন একটা রেঞ্জে আপডেট করছি এবং কোনো নোডের রেঞ্জ আপডেট রেঞ্জের ভিতরে হলে আমরা নিচে না গিয়ে বলে দিচ্ছি যে নিচের ইনডেক্সগুলোতে x যোগ হবে।

এখন মনে করো আমরা বেশ কয়েকবার আপডেট ফাংশন কল করেছি। নোডগুলোর প্রপাগেটেড ভ্যালুগুলা আপডেট হয়ে নিচের মতো হয়েছে:



যেসব নোডের ভ্যালু লিখিনি সেগুলোতে শূণ্য আছে মনে করো। তাহলে উপরের ছবির মানে হলো ১-৪ ইনডেক্সের সাথে x যোগ হবে, ৫-৭ এর সাথে z যোগ হবে ইত্যাদি।
এবার প্রশ্ন হলো কুয়েরি করবো কিভাবে?

ধরো আমাদের কুয়েরির রেঞ্জ হলো ১-৩। সাধারণভাবে আমরা আমাদের রিলেভেন্ট নোড ৪ আর ১০ থেকে ভ্যালু নিয়ে যোগ করে দিতাম। কিন্তু আমরা জানি ২ নম্বর নোডের নিচের সবার সাথে x যোগ হয়েছে, তাই ৪ নম্বর নোডের নিচের সবার সাথেও x যোগ হয়েছে! তাই আমরা ৪ নম্বর রেঞ্জে রাখা ভ্যালুর সাথে যোগ করে দিবো 2*x, কারণ ৪ নম্বর নোডের রেঞ্জে ২টি ইনডেক্স আছে এবং তাদের সবার সাথে x যোগ হয়েছে। ঠিক একই ভাবে যেহেতু ২ এবং ৫ নম্বর নোডের নিচে আছে ১০, তাই ১০ নম্বর নোডের রেঞ্জ ১টি ইনডেক্স আছে এবং তার সাথে যোগ হবে (x+y)।



তারমানে এটা পরিষ্কার যে কুয়েরি করা সময় কোনো নোডে যাবার সময় উপরের প্রপাগেটেড ভ্যালু গুলার যোগফল সাথে করে নিয়ে যেতে হবে। তাহলে কুয়েরির ফাংশনে carry নামের একটা প্যারামিটার যোগ করে দেই যার কাজ হবে ভ্যালুটা বয়ে নিয়ে যাওয়া:

```
int query(int node, int b, int e, int i, int j, int carry = 0)
{
    if (i > e || j < b)
        return 0;

    if (b >= i and e <= j)
        return tree[node].sum + carry * (e - b + 1); //সাম এর সাথে যোগ হবে সেই রেঞ্জের সাথে অতিরিক্ত

    int Left = node << 1;
    int Right = (node << 1) + 1;
    int mid = (b + e) >> 1;

    int p1 = query(Left, b, mid, i, j, carry + tree[node].prop); //প্রপাগেট ভ্যালু বয়ে নিয়ে যাচ্ছে carr
    int p2 = query(Right, mid + 1, e, i, j, carry + tree[node].prop);

    return p1 + p2;
}
```

আগের কোডের সাথে ডিফারেন্স হচ্ছে carry প্যারামিটারে এবং রিটার্ণভ্যালুতে। শুরুতে হিসাব করে রাখা sum এর সাথে যোগ হচ্ছে অতিরিক্ত যে ভ্যালু যোগ হয়েছে সেটা।

মোটামুটি এই হলো লেজি প্রপাগেশনের কাহিনী। সামারী করলে দাড়ায়:

> *১. রেঞ্জের আপডেট O(logn) এ করতে চাইলে লেজি প্রপাগেশন ব্যবহার করতেই হবে।*

২. লেজি প্রপাগেশনের কাজ হলো লিফ নোডে আপডেট না করে আগেই কোনো নোড আপডেট রেঞ্জের ভিতরে পড়লে সেখানে বলে দেয়া নিচের ইনডেক্সগুলো কিভাবে আপডেট হবে।

৩. কুয়েরি করার সময় উপরের নোডগুলোতে সেভ করা প্রপাগেশন ভ্যালুগুলো রিলেভেন্ট নোড ক্যারি করে নিয়ে আসতে হবে এবং সেই অনুযায়ী ভ্যালু রিটার্ণ করতে হবে।

অনেক সময় প্রবলেমে বলতে পারে একটা রেঞ্জের মধ্যে সবগুলো সংখ্যাকে x দিয়ে বদলে দিতে (x যোগ নয়), সেক্ষেত্রে প্রপাগেশন ভ্যালু হিসাবে রাখতে হবে নিচের সবনোডকে কোন ভ্যালু দিয়ে বদলে দিতে হবে সেটা। কুয়েরি করার সময় carry ভ্যালুতেও সামান্য পরিবর্তন আসবে, কিরকম পরিবর্তন আসবে সেটা বের করার দায়িত্ব তোমার উপরে ছেড়ে দিলাম।

সেগমেন্ট ট্রির মূল টিউটোরিয়াল এখানেই শেষ। পরবর্তী কোনো পর্বে ২ডি সেগমেন্ট ট্রি এবং কিছু প্রবলেম নিয়ে আলোচনা করবো। টিউটোরিয়ালের কোনো অংশ বুঝতে সমস্যা হলে ইমেইলে যোগাযোগ করতে পারো বা আরো ভালো হয় মন্তব্য অংশে সেটা জানালে।

প্র্যাকটিস প্রবলেম:
HORRIBLE
LITE
Lightoj Segment Tree Section

আগের পর্ব

ফেসবুকে মন্তব্য

1 comments