← Notes

Segment Tree and Lazy Propagation

186 Segment-tree Lazy-propagation CodeMonk Code Monk

There are many problems in which you need to query on intervals or segments of a collection of items. For example, finding the sum of all the elements in an array from index **left to right**, or finding the minimum of all the elements in an array from index **left to right**. These problems can be easily solved with one of the most powerful data structures, **Segment Tree** (in-short **Segtree**).

What is Segment Tree ?
Segment tree or segtree is a basically a binary tree used for storing the intervals or segments. Each node in the segment tree represents an interval.
Consider an array **A** of size **N** and a corresponding segtree **T**:

1. The root of **T** will represent the whole array **A[0:N-1]**.
2. Each leaf in the segtree **T** will represent a single element **A[i]** such that **0 <= i < N**.
3. The internal nodes in the segtree tree **T** represent union of elementary intervals **A[i:j]** where **0 <= i < j < N**.

The root of the segtree will represent the whole array **A[0:N-1]**. Then we will break the interval or segment into half and the two children of the root will represent the **A[0:(N-1) / 2]** and **A[(N-1) / 2 + 1:(N-1)]**. So in each step we will divide the interval into half and the two children will represent the two halves. So the height of the segment tree will be $\log_2 N$. There are **N** leaves representing the **N** elements of the array. The number of internal nodes is **N-1**. So total number of nodes are **2*N - 1**.

Once we have built a segtree we cannot change its structure i.e., its structure is static. We can update the values of nodes but we cannot change its structure. Segment tree is recursive in nature. Because of its recursive nature, Segment tree is very easy to implement. Segment tree provides two operations:

1. **Update:** In this operation we can update an element in the Array and reflect the corresponding change in the Segment tree.
2. **Query:** In this operation we can query on an interval or segment and return the answer to the problem on that particular interval.

Implementation:

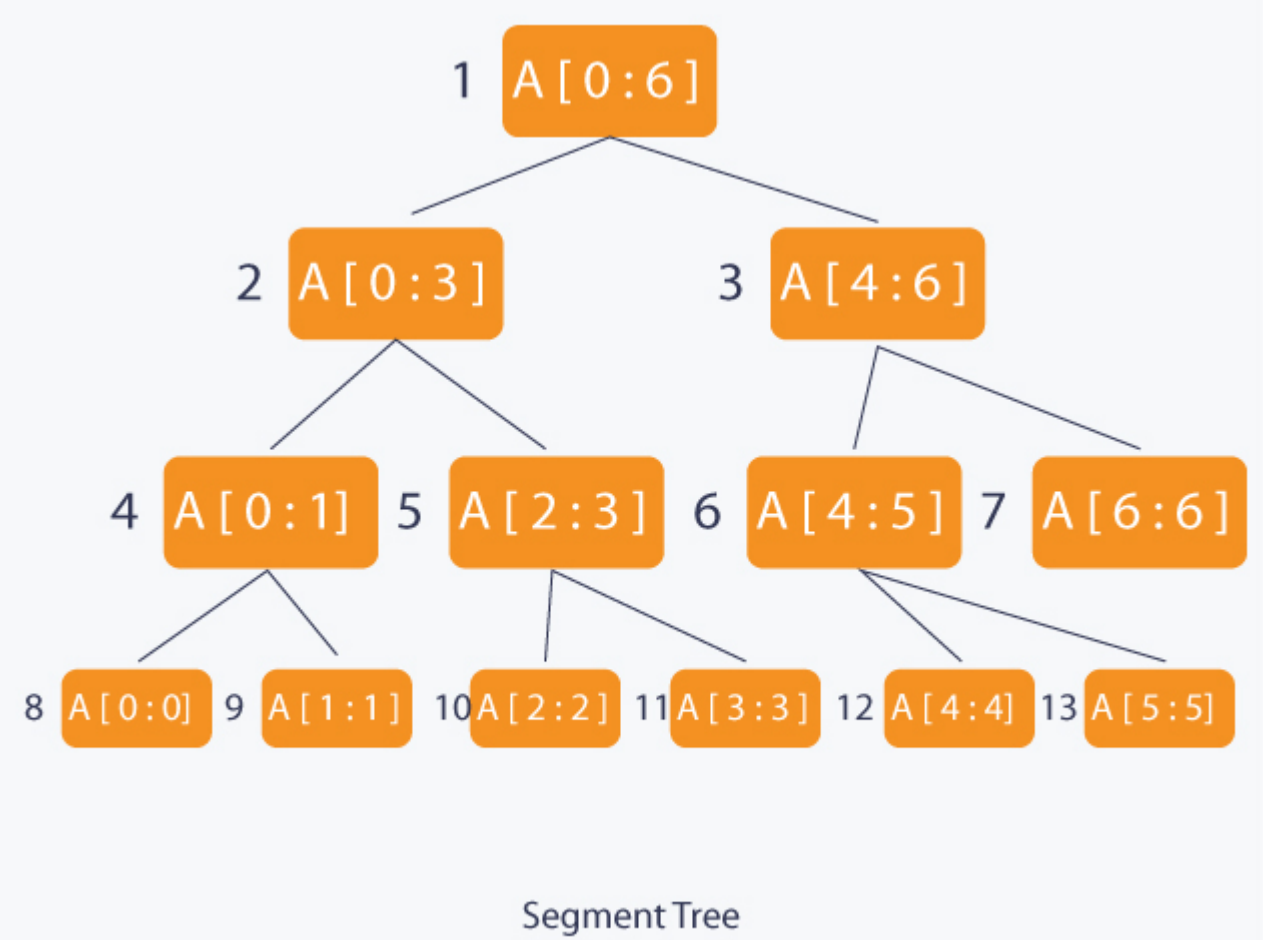
Since a segtree is a **binary tree**, we can use a simple linear array to represent the segment tree. In almost any segtree problem we need think about **what we need to store in the**

segment tree?.
For example, if we want to find the sum of all the elements in an array from index **left to right**, then at each node (except leaf nodes) we will store the sum of its children nodes. If we want to find the minimum of all the elements in an array from index **left to right**, then at each node (except leaf nodes) we will store the minimum of its children nodes.

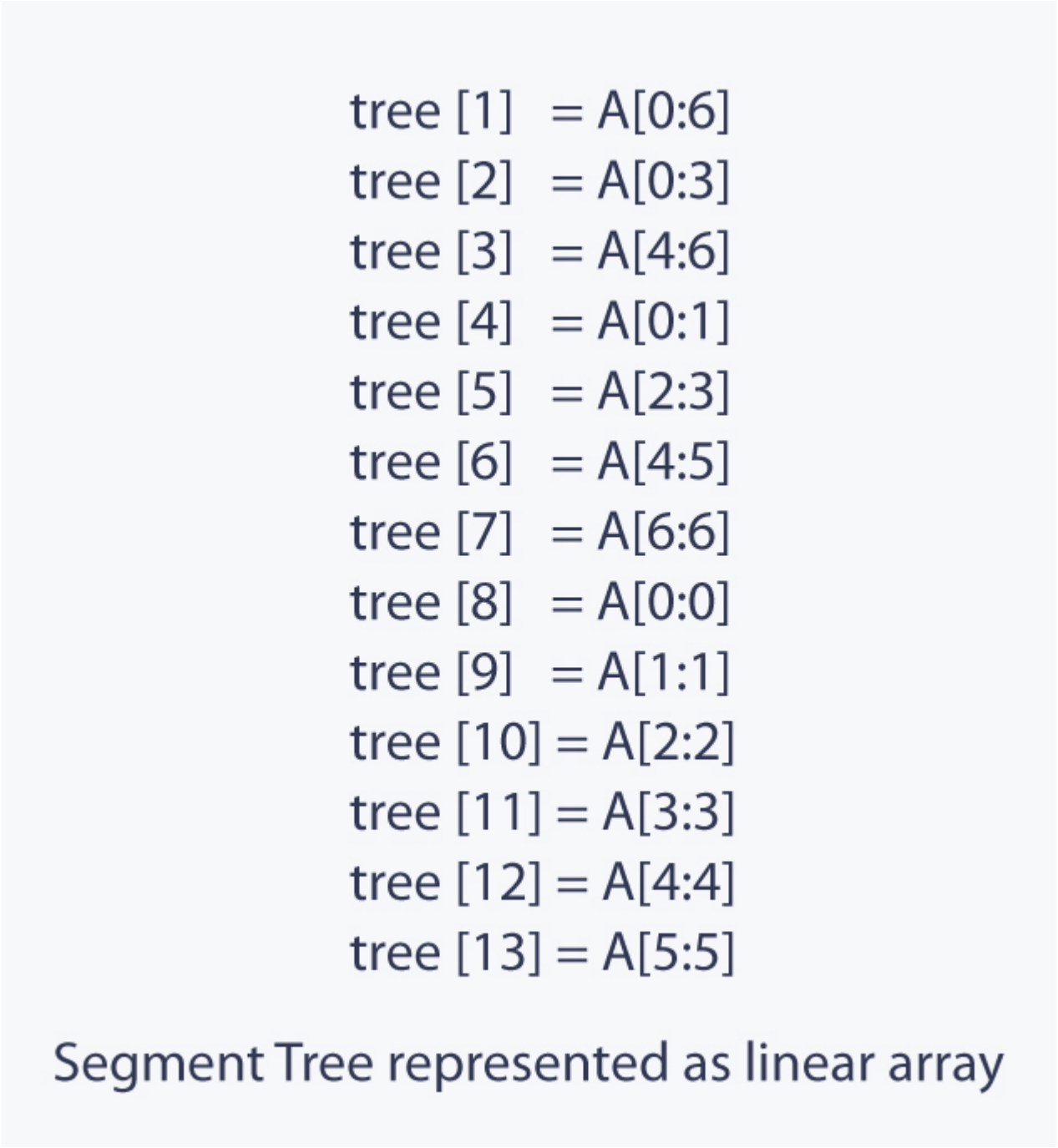
Once we know what we need to store in the segment tree we can build the tree using **recursion (bottom-up approach)**. We will start with the leaves and go up to the root and update the corresponding changes in the nodes that are in the path from leaves to root. Leaves represent a single element. In each step we will merge two children to form an internal node. Each internal node will represent a union of its children's intervals. Merging may be different for different problems. So, recursion will end up at root which will represent the whole array.

For update, we simply have to search the leaf that contains the element to update. This can be done by going to either on the left child or the right child depending on the interval which contains the element. Once we found the leaf, we will update it and again use the bottom-up approach to update the corresponding change in the path from leaf to root.

To make a query on the segment tree we will be given a range from **l to r**. We will recurse on the tree starting from the root and check if the interval represented by the node is completely in the range from **l to r**. If the interval represented by a node is completely in the range from **l to r**, we will return that node's value. The segtree of array **A** of size **7** will look like :



Segment Tree



All this we will get clear by taking an example. You can given an array **A** of size **N** and some queries. There are two types of queries:

1. **Update:** add a value of an element to **val** in the array **A** at a particular position **idx**.
2. **Query:** return the value of **A[l] + A[l+1] + A[l+2] + ... + A[r-1] + A[r]** such that **0 <= l <= r < N**

Naive Algorithm:
This is the most basic approach. For query simple run a loop from **l to r** and calculate the sum of all the elements. So query will take **O(N)**. **A[idx] += val** will update the value of the element. Update will take **O(1)**. This algorithm is good if the number of update operations are very large and very few query operations.

Another Naive Algorithm:

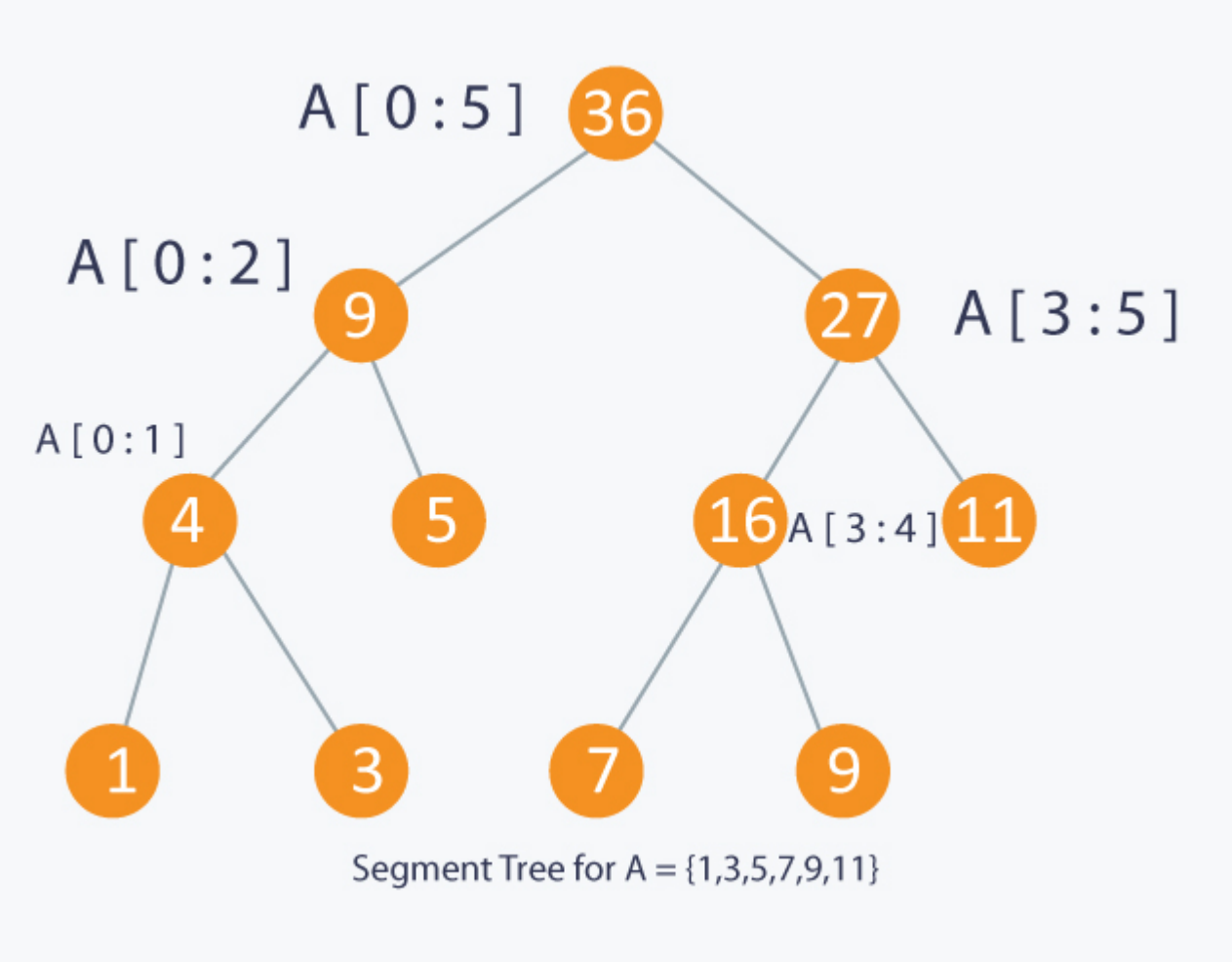
In this approach we will preprocess and store the cumulative sum of the elements of the array in an array **sum**. For each query simply return **(sum[r] - sum[l-1])**. Now query operation will take **O(1)**. But to update, we need to run a loop and change the value of all the **sum[i]** such that **l <= i <= r**. So update operation will take **O(N)**. This algorithm is good if the number of query operations are very large and very few update operations.

Using Segment Tree:

Let us see how to use segment tree and what we will store in the segment tree in this problem. As we know that each node of the segtree will represent an interval or segment. In this problem we need to find the sum of all the elements in the given range. So in each node we will store the sum of all the elements of the interval represented by the node. How do we do that? We will build a segment tree using recursion (bottom-up approach) as explained above. Each leaf will have a single element. All the internal nodes will have the sum of both of its children.

```
void build(int node, int start, int end)
{
    if(start == end)
    {
        // Leaf node will have a single element
        tree[node] = A[start];
    }
    else
    {
        int mid = (start + end) / 2;
        // Recurse on the left child
        build(2*node, start, mid);
        // Recurse on the right child
        build(2*node+1, mid+1, end);
        // Internal node will have the sum of both of its children
        tree[node] = tree[2*node] + tree[2*node+1];
    }
}
```

In the above code we will start from the root and recurse on the left and the right child until we reach the leaves. From the leaves we will go back to the root and update all the nodes in the path. **node** represent the current node we are processing. Since segment tree is a binary tree. **2*node** will represent the left node and **2*node + 1** represent the right node. **start** and **end** represents the interval represented by the node. Complexity of build() is **O(N)**.



To update an element we need to look at the interval in which the element is and recurse accordingly on the left or the right child.

```
void update(int node, int start, int end, int idx, int val)
{
    if(start == end)
    {
        // Leaf node
        A[idx] += val;
        tree[node] += val;
    }
    else
    {
        int mid = (start + end) / 2;
        if(start <= idx and idx <= mid)
        {
            // If idx is in the left child, recurse on the left child
            update(2*node, start, mid, idx, val);
        }
        else
```

```
        {
            // if idx is in the right child, recurse on the right child
            update(2*node+1, mid+1, end, idx, val);
        }
        // Internal node will have the sum of both of its children
        tree[node] = tree[2*node] + tree[2*node+1];
    }
}
```

Complexity of update will be **O(logN)**.

To query on a given range, we need to check 3 conditions.

1. range represented by a node is completely inside the given range
2. range represented by a node is completely outside the given range
3. range represented by a node is partially inside and partially outside the given range

If the range represented by a node is completely outside the given range, we will simply return 0. If the range represented by a node is completely inside the given range, we will return the value of the node which is the sum of all the elements in the range represented by the node. And if the range represented by a node is partially inside and partially outside the given range, we will return sum of the left child and the right child. Complexity of query will be **O(logN)**.

```
int query(int node, int start, int end, int l, int r)
{
    if(r < start or end < l)
    {
        // range represented by a node is completely outside the given range
        return 0;
    }
    if(l <= start and end <= r)
    {
        // range represented by a node is completely inside the given range
        return tree[node];
    }
    // range represented by a node is partially inside and partially outside
the given range
    int mid = (start + end) / 2;
    int p1 = query(2*node, start, mid, l, r);
    int p2 = query(2*node+1, mid+1, end, l, r);
    return (p1 + p2);
}
```

Updating an interval (Lazy Propagation):

Sometimes problems will ask you to update an interval from **l to r**, instead of a single element. One solution is to update all the elements one by one. Complexity of this approach will be **O(N)** per operation since where are **N** elements in the array and updating a single element will take **O(logN)** time.

To avoid multiple call to update function, we can modify the update function to work on an interval.

```
void updateRange(int node, int start, int end, int l, int r, int val)
{
    // out of range
    if (start > end or start > r or end < l)
        return;

    // Current node is a leaf node
    if (start == end)
    {
        // Add the difference to current node
        tree[node] += val;
        return;
    }

    // If not a leaf node, recur for children.
    int mid = (start + end) / 2;
    updateRange(node*2, start, mid, l, r, val);
    updateRange(node*2 + 1, mid + 1, end, l, r, val);

    // Use the result of children calls to update this node
    tree[node] = tree[node*2] + tree[node*2+1];
}
```

Let's be **Lazy** i.e., do work only when needed. How ? When we need to update an interval, we will update a node and mark its child that it needs to be updated and update it when needed. For this we need an array **lazy[]** of the same size as that of segment tree. Initially all the elements of the **lazy[]** array will be **0** representing that there is no pending update. If there is non-zero element **lazy[k]** then this element needs to update node **k** in the segment tree before making any query operation.

To update an interval we will keep 3 things in mind.

1. If current segment tree node has any pending update, then first add that pending update to current node.

2. If the interval represented by current node lies completely in the interval to update, then update the current node and update the **lazy[]** array for children nodes.
3. If the interval represented by current node overlaps with the interval to update, then update the nodes as the earlier update function

Since we have changed the update function to postpone the update operation, we will have to change the query function also. The only change we need to make is to check if there is any pending update operation on that node. If there is a pending update operation, first update the node and then work same as the earlier query function.

```
void updateRange(int node, int start, int end, int l, int r, int val)
{
    if(lazy[node] != 0)
    {
        // This node needs to be updated
        tree[node] += (end - start + 1) * lazy[node];    // Update it
        if(start != end)
        {
            lazy[node*2] += lazy[node];                   // Mark child as
lazy
            lazy[node*2+1] += lazy[node];                 // Mark child as
lazy
        }
        lazy[node] = 0;                                   // Reset it
    }
    if(start > end or start > r or end < l)               // Current segment
is not within range [l, r]
        return;
    if(start >= l and end <= r)
    {
        // Segment is fully within range
        tree[node] += (end - start + 1) * val;
        if(start != end)
        {
            // Not leaf node
            lazy[node*2] += val;
            lazy[node*2+1] += val;
        }
        return;
    }
    int mid = (start + end) / 2;
    updateRange(node*2, start, mid, l, r, val);         // Updating left child
```

```
    updateRange(node*2 + 1, mid + 1, end, l, r, val);    // Updating right
child
    tree[node] = tree[node*2] + tree[node*2+1];         // Updating root with
max value
}

int queryRange(int node, int start, int end, int l, int r)
{
    if(start > end or start > r or end < l)
        return 0;          // Out of range
    if(lazy[node] != 0)
    {
        // This node needs to be updated
        tree[node] += (end - start + 1) * lazy[node];             // Update
it
        if(start != end)
        {
            lazy[node*2] += lazy[node];          // Mark child as lazy
            lazy[node*2+1] += lazy[node];     // Mark child as lazy
        }
        lazy[node] = 0;                   // Reset it
    }
    if(start >= l and end <= r)               // Current segment is totally
within range [l, r]
        return tree[node];
    int mid = (start + end) / 2;
    int p1 = queryRange(node*2, start, mid, l, r);          // Query left
child
    int p2 = queryRange(node*2 + 1, mid + 1, end, l, r); // Query right child
    return (p1 + p2);
}
```

Practice Problems:

1. Help Ashu
2. Roy And Coin Boxes
3. Comrades-III

Solve Problems

COMMENTS (100) SORT BY: Relevance

Join Discussion...

Cancel Post

Rajeev Kumar 2 years ago

There is a little mistake in the line "So total number of nodes are 2*N - 1", since this is not true in every case. To be sure, that your recursive call does not cross the size limit, you should declare your segment tree array of size 4*N.

6 votes ● Reply ● Message ● Permalink

Gaurav Sen 2 years ago

I don't think so. The tree size is 1+2+4+...+N. This a geometric progression with sum =a*(lastTerm*r-1)/(r-1). Here r=2 and a=N and lastTerm=N. So we have sum =2*N-1.

3 votes ● Reply ● Message ● Permalink

Jatin Khurana 2 years ago

In case when A size is 6. How 2*N-1 is full filling the parent child relationship. It is going out of array when we take parent 6 and calculating it's child.

1 vote ● Reply ● Message ● Permalink

Gaurav Sen 2 years ago

No. The N you take in this case is the smallest power of 2 greater than A->size. That is equal to 8 in your case. Total size of segment tree will be 16 then.

0 votes ● Reply ● Message ● Permalink

Jatin Khurana 2 years ago

yes... in worst case size of segment tree is O(4*N)...

4 votes ● Reply ● Message ● Permalink

Utkarsh Kumar 2 years ago

What is exactly mean by (end - start + 1) in lazy propagation?

3 votes ● Reply ● Message ● Permalink

Sidhanta Choudhury 2 years ago

suppose there is update for arr[start] to arr[end] adding v so total added is (end-start+1)*v .As in a segment tree a node (tree[node])gives sum of array values it covers ,so we updated tree node as tree[node]+=(end-start+1)*v ,later we will update left and right node

10 votes ● Reply ● Message ● Permalink

Utkarsh Kumar 2 years ago

understood. Thanks!

0 votes ● Reply ● Message ● Permalink

Punit Bhatt Edited a year ago

in updateRange why dont we first check whether the segment is in the given interval (start>end or start>r or end<l). Only if this condition is not satisfied should the lazy[node]!=0 condition be checked.
Initially i thought the order wouldnt matter. But my solution for a particular question in Codechef was not getting accepted (WA) but when i reversed the order it got accepted

(lazy[node] condition first).
Can anyone tell me the significance of this order of conditions ?
▲ 7 votes ● Reply ● Message ● Permalink

Ankur dhir a year ago
same question
if someone knows please help..!!
▲ 1 vote ● Reply ● Message ● Permalink

Prashank Mehta a year ago
You should change the order. Author has used the same order as you suggested for query range.
▲ 0 votes ● Reply ● Message ● Permalink

Kaneki Ken a year ago
irrespective of the range, the update must be applied so that further updates happen over the current update so as to get the results of the queries correctly. The interval should not be considered as a factor, just apply the updates so further updates make sense. Correct me @author Akash if I am wrong.
▲ 0 votes ● Reply ● Message ● Permalink

kallu420 a year ago
The point is that even if that is done what you are suggesting there must not be any effect as the intervals will be out of range for the node, and this just increases one recursive function call and that doesn't seems to be good..!!
Have a look at my comment below
▲ 0 votes ● Reply ● Message ● Permalink

kallu420 ✎ Edited a year ago
Another cool example and SOME TROUBLE, HELP NEEDED and APPRECIATED :
https://www.hackerearth.com/code-monk-segment-tree-and-lazy-propagation/algorithm/monk-and-otakuland-1/
the above problem is solvable using lazy propagation and that is what has been done here :
http://ideone.com/FfagL8
but to get the above code accepted you need to activate the comment block in /* */, and deactivate the
un-comment // game starts here and // game ends here in the updateRange and query methods..
I am not able to understand this behavior....
▲ 1 vote ● Reply ● Message ● Permalink

SARTHAK MITTAL a year ago
I had the same doubt. But see, if suppose you check for range first, in case of partial overlap you recurse to the left and right child. Now the value of "node" is changed. The child in lazy[node] can be zero even when the parent was not. So now you need to check the zero for parent at (i-1)/2. I think this complicates the code more than the existing one. Think about it.
▲ 0 votes ● Reply ● Message ● Permalink

Shubham Singh 9 months ago
I also faced the same.My all test cases except the last one ,failed due to ignoring this order. But it seems very clear now. Note the last 3 statements of updateRange :
updateRange(node*2, start, mid, l, r, val); // Updating left child
updateRange(node*2 + 1, mid + 1, end, l, r, val); // Updating right child
tree[node] = tree[node*2] + tree[node*2+1]; // Updating root with max value
Note that you are calling the updateRange for both the left and right subtrees. Now suppose that left and right nodes are lazy but still any of these does not fall in the current

required range to be updated. Now just think if you ignore the updation of these nodes , (just because they do not lie in the current range), how can the last line : i.e.
tree[node] = tree[node*2] + tree[node*2+1];
will get executed correctly . This last line uses the fresh values of the left and right nodes , therefore, we must update the left and right nodes irrespective of the fact that it lies in current required range or not.
Hope it helps! :)

▲ 2 votes ● Reply ● Message ● Permalink

Sushant Gupta 5 months ago

Thanks. It helped

▲ 0 votes ● Reply ● Message ● Permalink

Shubham Singh 5 months ago

Pleasure! :)

▲ 0 votes ● Reply ● Message ● Permalink

Deepayan Bardhan a year ago

In updateRange funtion (before lazy propagation) why aren't we incrementing the the value of a[start]+=val ??

▲ 1 vote ● Reply ● Message ● Permalink

Akash Sharma Author a year ago

We can update it if we will use it. But in most case we don't need it as we have already stored its information into the segtree and using segtree to answer the queries.

▲ 1 vote ● Reply ● Message ● Permalink

Deepayan Bardhan a year ago

Then why is that done for only a index update ??

▲ 0 votes ● Reply ● Message ● Permalink

Akash Sharma a year ago

It wont affect the answer unless you are using the array itself.

▲ 0 votes ● Reply ● Permalink

Jayant Jain a month ago

yes bro you dont need add just replace it with a[start]=val;
tree[node]=val;
publisher has not properly explained it val he has used as difference between new value and current value :)

▲ 0 votes ● Reply ● Message ● Permalink

Saurabh Suman 2 years ago

Shouldn't the time complexity for build be O(n). Value of every node is calculated only once.

▲ 1 vote ● Reply ● Message ● Permalink

Anubhav Gupta 2 years ago

I agree. Recurrence for build is: T(n) = 2T(n/2) + O(1) = O(n)
So, time complexity of build should be O(n).

▲ 0 votes ● Reply ● Message ● Permalink

Okeke Ugochukwu 2 years ago

I don't think so. It should be O(lgn). Note that the recurrence relation you wrote is the same as that of binary search.

If a merge step is O(1) and you are at each iteration doing n/2 subproblems, then the height of the abstract binary tree is lgn. And since we are not spending time to do merging then everything is still lgn.

0 votes ● Reply ● Message ● Permalink

Anubhav Gupta 2 years ago

Hi Okeke,
recurrence for binary search is T(n) = T(n/2) + O(1)
but here it is T(n) = 2T(n/2) + O(1)
In binary search, we make only one recursive call on array of half the size, but here we are making two calls on arrays of half the sizes. So, according to Master's theorem it should be O(n).

0 votes ● Reply ● Message ● Permalink

Okeke Ugochukwu 2 years ago

Agreed!!! My bad.

0 votes ● Reply ● Message ● Permalink

Akash Sharma Author 2 years ago

Thanks for pointing it out. Fixed :)

0 votes ● Reply ● Message ● Permalink

Murat Kurnaz 2 years ago

Is it really needed to write start > end in if statement. Isn't that impossible ?

0 votes ● Reply ● Message ● Permalink

Shubham Marathia 2 years ago

yes it is needed in order to break your recursion when it low wants to go out of bounds , i.e. , greater then high

0 votes ● Reply ● Message ● Permalink

Manohar Reddy Poreddy a year ago

looks, it's more to do with mid+1

1 vote ● Reply ● Message ● Permalink

Madhuri Batchu 6 months ago

I still cant understand in which case start>end would be possible,could u pls elaborate ur explanation

0 votes ● Reply ● Message ● Permalink

Manohar Reddy Poreddy 6 months ago

sorry I don't remember anything about seg-tree anymore

0 votes ● Reply ● Message ● Permalink

prateekbehera a year ago

In the query function shouldn't it be if(l >= start and end >= r) instead of if(l <= start and end <= r) in the range represented by a node is completely inside the given range if statement

1 vote ● Reply ● Message ● Permalink

Deepayan Bardhan a year ago

No...because we have invoked the funtion initially with start=0 & end=n-1 .... And it gets divided into 2 halves ! when the condition if(l <= start and end <= r) gets satisfied , there's another half working simultaneously so that thing will be taken care of !

0 votes ● Reply ● Message ● Permalink

Manohar Reddy Poreddy Edited a year ago

Good question, my initial reaction was the same as you felt, then I worked out an example, [L:R] being partly left side, and partly right side of root node, then it was more clear. Also [L:R] being only on left or right side is also good simple example.
In simple, L & R are fixed, start and end are changing, getting smaller, at some time they

will be equal or less interval than L & R interval, which is what is (l <= start and end <= r). Hope that helped.

▲ 0 votes ● Reply ● Message ● Permalink

Vivek Singh Edited 5 months ago

last line of update should be tree[node]+=val*(min(end,r)-max(start,l)+1);
as tree[node*2] and tree[node*2+1] are still in process
try this problem it will be clear: http://www.spoj.com/problems/HORRIBLE/

▲ 1 vote ● Reply ● Message ● Permalink

Piyush Kumar a month ago

Thanks it helped me!!

▲ 0 votes ● Reply ● Message ● Permalink

Hitesh Garg 2 years ago

In Lazy Propagation , you have called two functions update_tree() and query_tree() without defining them , how this function is working and where is its body????

▲ 0 votes ● Reply ● Message ● Permalink

Akash Sharma Author 2 years ago

Thanks for point it out. Fixed. :)

▲ 0 votes ● Reply ● Message ● Permalink

Shivasurya S 2 years ago

Really useful :) thanks

▲ 0 votes ● Reply ● Message ● Permalink

Prateek Singh Chauhan 2 years ago

suppose if i have to find min in array using lazy propagation then y don't we do this (end-start+1) although we directly update value like SegTree[pos]+=LazyTree[pos];
plz help me out

▲ 0 votes ● Reply ● Message ● Permalink

Akash Sharma Author 2 years ago

Answer is quite simple. In RMQ, each node contain only a single element i.e. the minimum element in the given range while in RSQ, each node contain the sum of all the elements in the given range.

▲ 0 votes ● Reply ● Message ● Permalink

Prateek Singh Chauhan 2 years ago

got it... Thanks man

▲ 0 votes ● Reply ● Message ● Permalink

Swapnil Lohani a year ago

can we also use lazy propogation in case queries are not for adding some number in the range but some other type like dividing by some number?

▲ 0 votes ● Reply ● Message ● Permalink

Mojtaba Khooryani 2 years ago

very good
can you give examples of using methods?

▲ 0 votes ● Reply ● Message ● Permalink

Okeke Ugochukwu 2 years ago

Poster, nice tutorial. But what i would like to know more is how to qualify a problem as a candidate for segment tree and i want to know more about the "split and merge" part of segment tree.

If one can conquer the split and merge part of the problem then that is a big step.
▲ 0 votes ● Reply ● Message ● Permalink

Alex 2 years ago
Are there two different "segment trees"? According to wiki "segment tree is a tree data structure for storing intervals, or segments. It allows querying which of the stored segments contain a given point". Seems it slightly different from this one, used for range queries
▲ 0 votes ● Reply ● Message ● Permalink

~(BzzzKzzzB)~ 2 years ago
what are values passed to function call ------> void build(int node, int start, int end) in main function
▲ 0 votes ● Reply ● Message ● Permalink

Akash Sharma Author 2 years ago
build(1, 0, N-1);
▲ 0 votes ● Reply ● Message ● Permalink

~(BzzzKzzzB)~ 2 years ago
Thanks for quick reply. But why first parameter is 1 shouldn't it be 0.
▲ 0 votes ● Reply ● Message ● Permalink

Akash Sharma Author 2 years ago
No..it will be 1. Look at the first diagram for better understanding. We will basically pass the root node in the build function.
▲ 0 votes ● Reply ● Message ● Permalink

Ashwani Gautam 2 years ago
Yes you are right it should be 0, as it is root node thus it is the first element of the array tree
▲ 0 votes ● Reply ● Message ● Permalink

Akash Sharma Author 2 years ago
No. In binary tree we consider the indexing from 1. So that the left child will be 2*i and right child will be 2*i+1. If we consider 0-indexing then left child will be 2*0 = 0 which is actually the root.
▲ 0 votes ● Reply ● Message ● Permalink

Ashwani Gautam 2 years ago
Technically you are correct, but as you wrote we are implementing Binary tree as a linear array which are 0 Based indexed, so i still think it should be 0
▲ 0 votes ● Reply ● Message ● Permalink

Akash Sharma Author 2 years ago
That depends on us how we start indexing. But if you call the build with 0 it will end up in an infinite loop.
▲ 0 votes ● Reply ● Message ● Permalink

Ashwani Gautam 2 years ago
Obvioulsy if you use 0 over there then you have to change things from 2*i, 2*i+1 to 2*i + 1, 2*i + 2
▲ 0 votes ● Reply ● Message ● Permalink

Bhagwat kumar Singh 2 years ago
Hi friends Help needed.

I have tried to solve this question by Segment tree by lazy propagation (not needed but for practice) but only able to get 10 marks and getting run time error as well as time limit exceeded in different cases.

my code is :
https://code.hackerearth.com/9f0941N

0 votes ● Reply ● Message ● Permalink

Bhagwat kumar Singh 2 years ago

Question 2 : Roy and coin boxes

0 votes ● Reply ● Message ● Permalink

Saswat Kumar Mishra 2 years ago

On a completely different subject, shouldn't "end" variable be highlighted in black. Because, it's just a variable like start, node etc. I think you should check the syntax highlighter it may be taking "end" as a keyword.

0 votes ● Reply ● Message ● Permalink

Akash Sharma Author 2 years ago

its because end is a keyword :P :D

0 votes ● Reply ● Message ● Permalink

Sandeep Ravindra 2 years ago

Great tutorial :D

0 votes ● Reply ● Message ● Permalink

Shubham Marathia 2 years ago

Thanks mate for such nicely explained tutorial. Hoping to see more like this from you in near future :P

Can you do one more like this on Maximum sum in range using segtree please ? It would be totally awesome learning experience for all of us.

0 votes ● Reply ● Message ● Permalink

Sughosh Kaushik 2 years ago

In the update function of the segment tree(without lazy propagation), shouldn't the if condition be if(start == end && start == idx) ?

0 votes ● Reply ● Message ● Permalink

Aditya Agarwal 2 years ago

Great source to learn (y)

0 votes ● Reply ● Message ● Permalink

Mayank Agarwal 2 years ago

How can i generate all possible subsets in the given range (l,r) ?

0 votes ● Reply ● Message ● Permalink

Sourabh Khandelwal 9 months ago

Use Backtracking. The problem of generating all subsets of a subset is similar to generating all the permutations of a string.

0 votes ● Reply ● Message ● Permalink

Omar Yasser a year ago

Thank you ver much :)

0 votes ● Reply ● Message ● Permalink

Harshal Garg a year ago

in the pseudocode for update, shouldn't this be outside else in the main body
tree[node] = tree[2*node] + tree[2*node+1]; ?
please explain

0 votes ● Reply ● Message ● Permalink

Akash Sharma Author a year ago

No. if start == end that means we are at leaf node. else we are at some interior node and tree[node] = tree[2*node] + tree[2*node+1]; is only for interior nodes which have a left and right child.

0 votes ● Reply ● Message ● Permalink

Harshal Garg a year ago

got that.. thanks.. :)

0 votes ● Reply ● Message ● Permalink

Helper a year ago

Nice tutorial..!! Can you please help me solve the "Roy an Coin Boxes Problem" using segment trees.?

0 votes ● Reply ● Message ● Permalink

SHUBHAM GUPTA a year ago

can anyone tell the what is wrong with my solution...https://www.hackerearth.com/submission/4521850/

0 votes ● Reply ● Message ● Permalink

Bihan Sen a year ago

I have a query about the function updateRange using Lazy Propagation.
Suppose i want to update the whole array by a value,
then the function call would be updateRange(1,0,n-1,0,n-1,v)
In the first level call of the function it enters to the 3rd if block, updates the root and returns.
I guess the stored lazy values are adjusted during queries, right?

0 votes ● Reply ● Message ● Permalink

Saeem Ahamed a year ago

Shouldn't the tree array start with index of Zero0, and so that the left child will be (2*node+1) and the right child will be (2*node+2)

0 votes ● Reply ● Message ● Permalink

Manohar Reddy Poreddy a year ago

see this: https://www.hackerearth.com/messages/compose/r3gz3n/

0 votes ● Reply ● Message ● Permalink

Saeem Ahamed a year ago

this is not directing to any valid info. its opening up a messaging page. :-(

0 votes ● Reply ● Message ● Permalink

Manohar Reddy Poreddy a year ago

In the comments, above
look for a discussion that is started by "bzzzkzzzb" user, it ends up what you said.
the comment thread above says:
both are possible
1-based and 0-based
If 0-based index then as you said it will be 2i+1 and 2i+2
if 1-based index then, then it will be 2i & 2i+1, most probably 1st element (A[0]) is not used in array,

0 votes ● Reply ● Message ● Permalink

Abhishek Bhatt a year ago

in query()
return tree[node] condition should have r>=end
or else segmentation error creeps in

0 votes ● Reply ● Message ● Permalink

Manohar Reddy Poreddy a year ago

looks like it is there or is updated now

0 votes ● Reply ● Message ● Permalink

Juan Fiorenzano a year ago

Great article+++++

0 votes ● Reply ● Message ● Permalink

Mohit Vachhani a year ago

What can we do if we have to update the values in the range with the different values.
For example
Suppose Array to be
1 4 2 3 4
and then it should be updated by its smallest divisor.
so update range is from 1 to 3
new array would be
1 2 2 3 4
What should we do in this case?

0 votes ● Reply ● Message ● Permalink

Jaskaran Singh a year ago

In Lazy Propagation,
What does lazy[k] store?
All the pending updates to the kth node?
You should elaborate this with an example.
The rest of the tutoarial is great and very easy to understand.

0 votes ● Reply ● Message ● Permalink

Sarthak Gupta a year ago

What will be the complexity of update function of segment tree (not lazy propagation) if we are updating from [L,R] .
Will it be (R-L)log(n) or log(n+(R-L))?

0 votes ● Reply ● Message ● Permalink

Uppinder Chugh a year ago

(R-L)logn

0 votes ● Reply ● Message ● Permalink

shikhar jindal a year ago

What will be the time complexity for the updateRange function without lazy propagation??

0 votes ● Reply ● Message ● Permalink

Sai Sharath Edited a year ago

O(nlogn) forr each update

0 votes ● Reply ● Message ● Permalink

alok gupta a year ago

why trere is end-start+1 ?

0 votes ● Reply ● Message ● Permalink

don't worry a year ago

"There are N leaves representing the N elements of the array. The number of internal nodes is N-1. So total number of nodes are 2*N - 1" whether it means that size of segment tree would be 2*n-1 ?

▲ 0 votes ● Reply ● Message ● Permalink

Manveer Singh a year ago

In that "Another naive solution:" heading, you may also write that ,the making of cumulative sum array is itself O(N) complexity.

▲ 0 votes ● Reply ● Message ● Permalink

Foyaz Akanda a year ago

awesome article

▲ 0 votes ● Reply ● Message ● Permalink

Subham 9 months ago

Thank you Hackerearth and Akash Sharma for this wonderful article. It made me understand segtree thoroughly. Just keep on doing your great work.

▲ 0 votes ● Reply ● Message ● Permalink

Zaid Alkhateeb 8 months ago

the complex of update range is O(n) because you reach to the son of all node

▲ 0 votes ● Reply ● Message ● Permalink

Bhushan Dhodi Edited 6 months ago

Build tree is wrong! Left sub tree is not built! I get this output 36 27 16 11 7 9 0 0 0 0 0 for given example..please help With what initial parameters should we call build function?

▲ 0 votes ● Reply ● Message ● Permalink

Hrudai pabbisetty 6 months ago

build(1,0,n-1);

▲ 0 votes ● Reply ● Message ● Permalink

Hrudai pabbisetty 6 months ago

Building the segment tree is top-down right?

▲ 0 votes ● Reply ● Message ● Permalink

Adarsh Kumar 5 months ago

The 2nd practice problem "Roy and Coin Boxes" mentioned above is nowhere related to Segment Tree. I can't understand why it is here..

▲ 0 votes ● Reply ● Message ● Permalink

Filipe Herculano 4 months ago

The complexity of build is more accurate if we say it is O(n * log n) ?

▲ 0 votes ● Reply ● Message ● Permalink

suman 16 days ago

Total no of node in segment tree will be 2*n-1
But we must have segment tree array of 4*n

▲ 0 votes ● Reply ● Message ● Permalink

AUTHOR

Akash Sharma
Problem Curator at Hacker...
Dehradun
7 notes

TRENDING NOTES

Python Diaries Chapter 3 Map | Filter | For-else | List Comprehension
written by Divyanshu Bansal

Bokeh | Interactive Visualization Library | Use Graph with Django Template
written by Prateek Kumar

Bokeh | Interactive Visualization Library | Graph Plotting
written by Prateek Kumar

Python Diaries chapter 2
written by Divyanshu Bansal

Python Diaries chapter 1
written by Divyanshu Bansal

more ...