LAZY PROPAGATION

0

i understood segmented tree...but nowher i can see clear explanations of "LAZY PROPAGATION"....i searched topcoder,geeksforgeeks and all possible blogs and forums....Is ther any book on advanced data structures??...i want to clearly understand this lazy propagation....pls help..thanks...

2

help

asked 20 Feb, 03:47
chaseme
546●9●23●43
accept rate: 0%

2 Answers:

oldest newest most voted

21

Ok, here I go.

Consider a standard interval tree (usually called segment tree, but I use a transliteration from the Slovak term), which supports operations "set the value of node *i* to *v*" and "return the maximum value from the interval *[u,v)*" (semi-closed intervals are comfortable to use). This tree keeps the following information in a node: the interval *[u_i,v_i)*, the maximum *value* from this interval and at most two sons of this node.

Updates are done recursively: after the value in one of a node's sons is updated, the value in the node itself is updated. Queries are answered this way: you always query on (node,interval), where *interval* is the subinterval of the one *[v_i,u_i)* in the *node*. If *interval* is empty, the answer's trivial; if it's equal to *[v_i,u_i)*, it's just *value* of the *node* and otherwise, you compute it recursively.

Now, what does lazy propagation (lazy-loading is the Slovak transliteration) do? You don't update "one node to value *v*", but "interval *[x,y)* to all values equal to *v*". You also add one more value to each node: the *modifier*, and a single function *modify(node)*. The *modifier* holds the following information: "all elements in this interval are actually equal to *x*" or "ignore this" (if there's no fitting *x*, for example during initialization). The *modify()* function updates all sons' *modifier*s to this one (we need to make sure that the *modifier* of an ancestor has higher priority than the one of a son), sets the *value* in the node equal to the *modifier* and changes *modifier* to "ignore", to avoid collisions with later updates. This basically processes all updates in the given node, but not in the underlying subtree.

Before any call of functions *update()* or *query()*, you call *modify()* on this node. That way, you'll be certain that earlier updates in this node have been processed and there can't be any collisions.

Notice that *query()* hardly changes. You still do the same, just with calling *modify()* beforehand on the node on which you call *query()*, and time complexity stays the same: *O(log N)*.

Updating changes a bit, because it affects the modifier. Firstly, *update(node,interval)* is also called with *interval* being a subinterval of the one in the *node*. It's called in a similar way to *update()*: if *interval* is empty, just return from the function call; if it's the whole interval stored in the node, all elements are updated to the same value, which means you only change the *modifier* and return; the rest of the work would be done when any function is called on this node later on. And in the last other case, with *interval* being a non-trivial one, you call *update()* on the sons (if the *interval* is non-trivial, both exist), set *value* to their maximum and return. Just remember: don't forget to call *modify(node)* before *update(node,interval)* or *query(node,interval)*.

We can see that any function call only takes *O(1)* time. The trick is in the width of recursion. It can be shown that if both recursive calls (whether of *update()* or *query()*) are non-trivial, then it won't happen again, which means no further recursive branching to depth larger than 1, and therefore *O(tree depth)=O(log N)* calls (or time).

There are many other variants of lazy-loaded interval trees, which depend on the types of updates and queries, and on the number of them - it's possible to have two types of updates or of queries, etc., but you need to handle specific commands in specific order to avoid collisions, and it's quite complicated.

link | award points

answered 20 Feb, 06:05
xellos0
3.7k●5●28●50
accept rate: 10%

Can you please explain this sentence: all elements in this interval are actually equal to x" or "ignore this" (if there's no fitting x, for example during initialization). From what I get, the x should be v instead.

rajbsk (25 Sep, 05:49)

@xello..reali thanks for the explanation...also i got a aother source to leaern this concept here

-2 http://sportcoder.com/segment-trees-lazy-updates/

link | award points

answered **20 Feb, 11:54**
chaseme
546●9●23●43
accept rate: 0%

1 dead link...

betlista ◆◆ (04 Nov, 03:05)

Your answer

[hide preview]

community wiki

Post Your Answer