# Splay tree potential function: why sum the logs of the sizes?

I'm teaching a course on data structures and will be covering splay trees early next week. I've read the paper on splay trees many times and am familiar with the analysis and intuition behind the data structure. However, I cannot seem to find a solid intuition for the potential function that Sleator and Tarjan use in their analysis.

The analysis works by assigning each element in the tree an arbitrary weight $w_i$, then setting the size $s(x)$ of a node to be the sum of the weights of the nodes in the subtree rooted at $x$. They then take the log of this value to get the rank $r(x)$ of the node, so $r(x) = \log s(x)$. Finally, the potential function of the tree is defined as the sum of the ranks of all the nodes.

I understand that this potential function works correctly and I can follow the analysis, but I don't see why they would choose this potential. The idea of assigning a size to each node makes sense to me, since if you sum up the sizes, you get the weighted path length of the tree. However, I can't figure out why they decided to take the logs of the weights and then sum those up instead - I don't see any natural property of the tree that this corresponds to.

Does the potential function of the splay tree correspond to some natural property of the tree? Is there a particular reason, other than "it works," that they would choose this potential? (I'm specifically curious because this set of course notes mentions that the "analysis is black magic. [N]o idea how discovered")

Thanks!

ds.data-structures intuition amortized-analysis

edited Apr 25 '14 at 20:54

asked Apr 25 '14 at 20:33
templatetypedef
832 • 5 • 20

I've heard this "it's black magic" explanation before as well. Did you try emailing the Sleator and Tarjan? – jbapple Apr 26 '14 at 0:44

@jbapple I haven't emailed them yet, since I was hoping that "the community at large" would be able to help out. I also figured that pinging someone about work they did 30 years ago might not necessarily elicit a response. :-) – templatetypedef Apr 26 '14 at 0:47

I'm not very familiar with it, but I just look at the paper very roughly, I think the only reason is that they want to have a small changes by potential functions after splay operation, e.g if we replace $\log$ with $\log\log$ or with $log^*$ seems still everything works fine (I didn't prove it but seems just needs a simple mathwork). But if we leave it as $s$, then that amortize analyse while is correct, but it's not a good upperbound (something like $(m+n)^2$ ). It's not related to any property of tree at all. – Saeed Apr 26 '14 at 12:39

I've always black-boxed the rank of x as "the depth of the ideal binary search tree containing the descendants of x", but that's more a mnemonic than useful intuition. – Jeffε Apr 26 '14 at 12:41

1 Answer

## How to come up with sum-of-logs potential

Let's consider the BST algorithm $A$ that for each access for element $x$, it rearranges only elements in the search path $P$ of $x$ called before-path, into some tree called after-tree. For any element $a$, let $s(a)$ and $s'(a)$ be the size of subtree rooted at $a$ before and after the rearrangement respectively. So $s(a)$ and $s'(a)$ may differ iff $a \in P$.

Moreover, $A$ only rearranges constantly many elements in the search path at any moment. Let's call this type of algorithm "local" algorithm. For example, splay tree is local. It rearranges only at most 3 elements at a time by zig, zigzig and zigzag.

Now, any local algorithm that creates "many" leaves in the after-tree, like splay tree, has the following nice property.

> We can create a mapping $f : P \to P$ such that
>
> 1. There are linearly many $a \in P$, where $s'(f(a)) \leq s(a)/2$.
> 2. There are constantly many $a \in P$, where $s'(f(a))$ can be large but trivially at most $n$.
> 3. Other elements $a \in P$, $s'(f(a)) \leq s(a)$.

We can see this by unfolding the change of search path. The mapping is actually quite natural. This paper, A Global Geometric View of Splaying, precisely shows the details how to see the above observation.

After knowing this fact, it is very natural to choose sum-of-logs potential. Because we can use the potential change of the type-1 elements to pay for the whole rearrangement. Moreover, for other-type elements, we have to pay for the potential change by at most logarithmic. Hence, we can derive logarithm amortized cost.

I think the reason why people think this is "black magic" is that the previous analysis do not "unfold" the overall change of the search path, and see what really happens in one step. Instead, they show the change in potential for each "local transformation", and then show that these potential changes can be magically telescoped.

P.S. The paper even shows some limitation of sum-of-logs potential. That is, one can prove satisfiability of *access lemma* via sum-of-logs potential to only the local algorithm.

## Interpretation of sum-of-logs potential

There is another way to define the potential of BST in Georgakopoulos and McClurkin's paper which is essentially the same as sum-of-logs potential in Sleator Tarjan's paper. But this gives good intuition to me.

Now I switch to the notation of the paper. We assign a weight $w(u)$ to every node $u$. Let $W(u)$ be the sum of the weight of $u$'s subtree. (This is just the size of $u$'s subtree when the weight of every node is one.)

Now, instead of defining the rank on the nodes, we define the rank to the edges, which they called **progress factor**.

$$pf(e) = \log(W(u)/W(v)).$$

And the potential of tree $S$ is

$$\Phi(S) = \sum_{e \in S} pf(e).$$

> This potential has a natural interpretation: if during a search we traverse an edge $(u, v)$ we reduce the search space from the descendants of $u$ to the descendants of $v$, a frantional reduction of $W(u)/W(v)$. Our progress factor is a logarithmic measure of this 'progress', hence its name. [From Section 2.4]

Observe that this is almost equal Sleator Tarjan's potential, and it is additive on paths.

**edit:** It turns out that this alternative definition and the intuition behind it was described long ago by Kurt Mehlhorn. See his book "Data Structures and Algorithms" Volume I, Section III. 6.1.2 Splay Trees, pages 263 - 274.

edited Mar 24 '15 at 10:11

answered Feb 10 '15 at 17:10

Thatchaphol

850 • 7 • 17

That's really great! Thanks for sharing! – templatetypedef Feb 10 '15 at 18:40