# Ranged-Op Tree

Terence An

October 12, 2022

**Abstract**

The paper gives theoretical and implementation details for a data structure that allows cheap queries of ranged sums (or other associative binary operation) and cheap insertion and deletion of new range values. Given a set $S$ of pairs $(l, r)$ where $l$ is a member of the monoid being summed and $r$ is itself a pair indicating a one-dimensional integer range, this tree is able to quickly query $min_{t \in R}[\sum_{s \in S : t \in s.r} s.l]$ for any given integer range $R$. The intention for designing such a data structure is to cheaply track AMM liquidity at every tick to provide the minimum liquidity available in a range. The implementation details will include portions specific to smart contract development.

## 1 High Level

Consider a sequence $a_i thatrepresentstheamountofaresourceavailableatagivenindex\mathrm{i}, wewillinterchangeablym$
we have to sum the values in all the nodes on the path from the $i$th leaf node to the root. This lets us query $a_i$ in logarithmic time, but that is not the purpose of the tree. The purpose of the tree is to also answer range based queries in logarithmic time. For example, users can query the total amount of liquidity in a range, or the minimum, or the average, etc.

This tree can only compute functions that are additive. What I mean by this is that $f$ is additive if it is composed of functions $g$ where $g$ follows the property that $\exists h$ s.t. $g(x_0 + k, x_1 + k, ..., x_n + k) = h(g(x_0, x_1, ..., x_n), k)$. This property allows the value of $f$ to be computed by storing the values of $g$ at each node for its subtree and if any node modifies its value we can propogate the new values of $f$ up to the root.

For example we can compute the minimum liquidity available in any range. This is accomplished by storing two values at every node, the amount of resources available for precisely the range represented by the node and the subtree minimum. The subtree minimum answers the query if the query's range exactly matches the node's range. If the ranges do not exactly match, the queried range is broken up into partitions that do exactly match individual nodes' ranges in the tree. Then we take the minimum over those nodes' subtree minimums to answer the query.

In order to update the tree with resources for a given range, the range is once again broken up into the smallest number of node ranges whose union matches the given range, and that resource value is added to the sum in those nodes. Then starting from each node, we walk up the tree propogating the new subtree minimum.

Certain interesting properties make the walk cheap to accomplish. In fact insertions, removals, and queries all take $O(log(n))$ where $n$ is the size of the entire range covered by the tree.

## 2 Theoretical Design

The design starts similarly to an augmented tree where each node stores a value $L$ and the minimum subtree value $M$, both of which are members of the monoid being summed. Each node represents a range and has two children each represeting half their range except for the leaf nodes which have no children. Leaf nodes are those representing a single integer ranage. The top node represents the full range. Note that not all ranges are in the tree. We constrain ourselves to finite integer ranges and WLOG we assume the finite rate is centered around 0 with a total range width that is a power of two. Then the splits always divide the ranges in half which trivially makes all ranges start with a power of two and have a width that is a power of two.

In order to query the tree with an range $R$ that is contained within our total finite range, we decompose $R$ into $N_R = R_0, R_1, ...$ where $N_R$ is a set of ranges possessed by exactly one node in the tree. These ranges do not overlap, and $\bigcup N_R = R$. We will show the properties that $N_R$ must follow. From now on, we assume $N_R$ is the smallest of such sets.

But first, we'll define left-adjacent and right-adjacent.

**Definition 2.1** (Left-adjacent Right-adjacent). *A node $p$ is left-adjacent from $q$ if the path from either node's parent to the other node follows one left branch first and then one or more right branches and the node closer to the root is a right child. I.e. $lr+$ in regex notation where $l$ is a taken left branch and $r$ is a taken right branch. Likewise, right-adjacent is defined by a path with one right branch and then any number of left branches where the lower depth node is a left child.*

**Definition 2.2** (Consecutive nodes)**.** *We also define two nodes to be consecutive if the union of their ranges is continguous, i.e. for all $a, b$ in the union of their ranges, any $c$ such that $a < c < b$ must also be in the union.*

The main property will be building up to is this:

**Theorem 2.1.** *The nodes in $N_R$ follow a trapezoidal shape. Starting from the left-most node, there is a (possibly empty) consecutive series of left adjacent nodes of strict-monotonically lower depth (lower distance from the root), and starting from the right-most node there is another (possibly empty) consecutive series of right adjacent nodes of strict-monotonically lower depth. This left partition of nodes and right partition of nodes comprises the whole of $N_R$.*

**Lemma 2.2.** *There is a unique $N_R$ that is minimal.*

Assume there are two such $N_R$'s, $A$ and $B$. WLOG there exists a range $a$ in $A$ that is not in $B$. We know ranges don't overlap and the unions are identical thus there exists a set of ranges whose union overlaps with $a$ identically. Replacing them with $a$ in $B$ produces a smaller $N_R$.

**Lemma 2.3.** *If one node's range is a subset of another node, then that node is a descendent of the other node.*

This follows trivially from the construction of the tree.

**Lemma 2.4.** *There cannot be two nodes of the same depth that overlap in range.*

By construction nodes of the same depth have the same range width and their starting indices are all multiples of that width. If two node ranges overlapped the start index of one of them cannot be a multiple of the width.

**Lemma 2.5.** *$N_R$ nodes do not overlap in range.*

Node ranges start with a power of two. If a node starts within the range of another node, then that node's range is entirely contain within the other node's range or there is a subsequent section that isn't. In the former case, it means the node is not necessary and $N_R$ is not minimal. In the latter case these two nodes cannot be of the same depth by the previous lemma. But the deeper of the two nodes must have a parent at the same depth at the other node which is not the other node (or else it'd be fully contained) and that parent's range must also overlap which cannot be true by the previous lemma.

**Corollary 2.5.1.** *There is a well ordering of $N_R$ nodes by ranges where every end index of one range precedes the next smallest starting index whose corresponding end index is the next smallest end index.*

**Lemma 2.6.** *When $N_R$ are well-ordered, all nodes are consecutive with their neighbors in the ordering.*

If they were not, then $\bigcup N_R \neq R$ since $R$ is continuous.

**Definition 2.3** (Far-adjacent)**.** *Let $p$ be lowest common ancestor of two nodes. These two nodes are far-adjacent if the right child of $p$ is left-adjacent to one node and the left child of $p$ is right-adjacent to the other.*

**Theorem 2.7.** *Two nodes are consecutive if and only if they are either siblings, left-adjacent, right-adjacent, or far-adjacent.*

**Lemma 2.8.** *If $a, b, c$ are consecutive nodes and $a$ and $b$ are left-adjacent, then $b$ and $c$ are either left-adjacent or far-adjacent.*

**Lemma 2.9.** *If $a, b, c$ are consecutive nodes and $b$ and $c$ are right-adjacent, then $a$ and $b$ are either right-adjacent or far-adjacent.*

**Lemma 2.10.** *The left-most node is a right child and the right-most node is a left child.*

**Theorem 2.11.** *$N_R$ when well ordered, is a sequence of left-adjacent nodes, a pair of far-adjacent nodes, and then a sequence of right-adjacent nodes.*

**Lemma 2.12.** *Left-adjacent nodes cannot be of the same depth. Likewise for right-adjacent nodes.*

Show they decrease in depth.

We will call the lowest depth node of a set (the node closest to root) the tallest node.

The tallest nodes of the two sides are far-adjacent.

Theorem proved.

This provides us with the first property of the tree, the range partition.

Then we use the range partition to add liquidity to and walk up the tree.

Prove this update to minimim will also give the subtree minimum value for any contained index.

This naturally holds for removing liquidity.

Also holds for tracking maximums.

**Definition 2.4.** *The peak of our breakdown is the highest node in the smallest subtree containing our breakdown trapezoid.*

**Theorem 2.13.** *The xor of the bit representations of the two range bounds is the bit representation of the peak's base, and the least significant bit of the base is the peak's range.*

*Proof.* First, this procedure produces a parent of both the low and high bounds. The produced key's range spans both the low and high since it spans all possible values in that range, which includes those two by the definition of xor. Since all nodes at a single tier have mutually exclusive ranges this is the only possible parent at that tier.

Second, the bit below the lsb is a 0 in the low's bit representation and a 1 in the high's representation or else the xor would have produced that below or lower as its lsb. Thus no range smaller than the one we've found is sufficient to cover the whole range. Therefore lowest common ancestor must be at this tier.

Thus this procedure produces the lowest common ancestor. $\square$