

Segmented Winner Trees¹

By Andrew Lim and Sartaj Sahni

Abstract: A new data structure, *segmented winner tree*, is introduced. This is useful when one needs to represent data that are partitioned into segments. The segment operations that are efficiently supported are: initialize a unit length segment, find the element with least value in any given segment, update an element in any segment, merge two adjacent segments, and split a segment.

Key Words and Phrases: Data structures, winner trees, segmented data

1. Introduction

Suppose that we have n elements $1, 2, \dots, n$ that have been partitioned into some number of segments such that the elements in each segment are contiguous. Figure 1 shows a possible partitioning of the eleven elements $1, 2, \dots, 11$ into three segments. A value, $v[i]$, is associated with each element i .

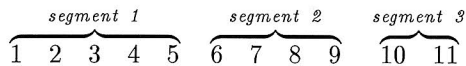


Figure 1: Eleven elements partitioned into three segments

The operations we wish to perform on these segments are:

1. *Initialize*(l_1): Initialize the one element segment $l_1 : r_1$ where $r_1 = l_1$.
2. *FindMin*(l_1, r_1, x): Find the smallest value in the segment comprised of elements $l_1 : r_1$. This value is returned in variable x .
3. *Update*(l_1, r_1, i, y): Change the value of element i to y . Element i is a member of the segment that is comprised of elements $l_1 : r_1$. Note that $l_1 \leq i \leq r_1$.
4. *Merge*(l_1, r_1, r_2): Merge the adjacent segments $l_1 : r_1$ and $r_1 + 1 : r_2$ into a single segment $l_1 : r_2$.
5. *Add*(l_1, r_1, y): This is a special case of merge in which the right segment $r_1 + 1 : r_2$ consists of a single element with value y . I.e., $r_1 + 1 = r_2$ and $v[r_2] = y$.

¹This research was supported, in part, by the National Science Foundation under grant MIP91-03379.

6. *Split*(l_1, r_1, q): The segment $l_1 : r_1$ is split into two adjacent segments $l_1 : q$ and $q + 1 : r_1$.

Two applications for a data structure that efficiently supports these operations are:

1. *Geographic Service Environment*: In this, the n elements represent n clients who are located adjacent to one another on the same side of a street. The clients are served by servers who, for efficiency reasons, serve a contiguous set of clients. So, each segment represents a set of clients served by the same server. In the ideal situation, we have one server per client. In this case all segments are of size one. If a server is disabled, then the set of clients served is assigned to one of the servers associated with an adjacent server (segment merging). Alternatively, the client set could be partitioned (segment split) with the left (right) partition being assigned to the left (right) server (segment merging). If a new server becomes available, then a client set for this server is created by partitioning off clients from at most two adjacent client sets (segment splitting). A server selects a client from its client set, for service, based on the clients value. We assume priority is given to the client with the least value. This uses the *FindMin* operation. The value (or priority) of a client may change in time. This requires one to update the clients value.
2. *Cell Joining*: When stretching two adjacent cells of a VLSI design so as to minimize the total wire length subject to a minimum area constraint[1], the pins on one side of a cell represent the n elements of our data structure. The pins are maintained as segments of adjacent pins and the distance between pins is the pin value. In the course of the algorithm, adjacent pin segments may coalesce; it is necessary to obtain the pin with the least value in any segment; and it is necessary to update the pin values. To implement this algorithm efficiently, we need an efficient data structure for the segment operations (except split) defined above.

The data structure we propose, *segmented winner trees*, represents each segment as a collection of winner trees [2] with each winner tree representing a number of elements that is a power of 2. All winner trees for all segments are compactly represented within an array of size $2n - 1$. Because of this, explicit pointers are not used and one can move up and down a tree by performing simple arithmetic operations on the current location within a tree [2].

Two of the already known data structures that could be used to represent our segments are *Leftist trees* and *Fibonacci heaps* [2, 3]. Table 1 compares the complexity of these data structures and that of segmented winner trees. s is the size of the segment involved (in the case of a merge, it is the size of the resulting segment). The complexities are amortized complexities for the case of Fibonacci heaps and worst case per operation complexities for leftist trees and segmented winner trees.

We note that if the updates are restricted to those that only reduce a value, then the amortized update cost for Fibonacci heaps is $O(1)$.

Segmented winner trees are defined in Section 2. Algorithms for the segmented winner tree operations are provided in Section 3. Experimental data comparing the performance of leftist trees, Fibonacci heaps, and segmented winner trees is provided in Section 4.

Operation	Leftist Trees	Fibonacci Heaps	Segmented Winner Trees
Initialize	$O(1)$	$O(1)$	$O(1)$
FindMin	$O(1)$	$O(1)$	$O(\log s)$
Update	$O(s)$	$O(s)$	$O(\log s)$
Merge	$O(\log s)$	$O(1)$	$O(\log s)$
Add	$O(\log s)$	$O(1)$	$O(\log s)$
Split	$O(s)$	$O(s)$	$O(1)$

Table 1: Data structure complexities

2. Segmented Winner Trees

A *winner tree* is a tree where each node represents the smaller of its children [2]. A *segmented winner tree* is a complete binary tree [2]. I.e., each node has degree 0 or 2; the leaves are on at most two adjacent levels; if the leaves are on two adjacent levels, then all leaves on the next to last level are at the right end of that level. Figure 2 shows a complete binary tree with 11 leaves and 21 nodes. Node numbers are inside the nodes. The leaves have been numbered left to right. The leaf numbers are given outside the leaves. Note that for every m , $m \geq 0$, there is a unique m node complete binary tree. Further, for every n , $n \geq 0$, there is a unique n leaf complete binary tree. The number, m , of nodes in this complete binary tree is $2n - 1$ when $n \geq 1$ and 0 when $n = 0$.

A complete binary tree with m nodes is efficiently stored in an array $t[1..m]$ with node i of the tree represented in position i of the array. The parent of node i is in position $i \text{ div } 2$, its left child (if it exists) is in position $2i$, and its right child (if it exists) is in position $2i + 1$ [2].

To represent a collection of segments with n elements, we use a complete binary tree with n leaves. This has $m = 2n - 1$ nodes. Element i is represented by leaf i , where the leaves are numbered 1 through n left to right (see Figure 2). The position of element i in the array $t[1..m]$ is easily obtained using the function *Position* of Figure 3. In this, k is a power of 2 such that $\frac{k}{2} < n \leq k$. So, when $8 < n \leq 16$, $k = 16$. I.e. $k = 2^{\lceil \log_2 n \rceil}$, $n > 0$. k gives the position of element 1. Hence, we expect to find element i in position $j = k + i - 1$ unless $j > n$. In this case, only $m - k + 1$ leaves are at the lowest level and $i > m - k + 1$. The first leaf on the preceding level is at position $\lfloor \frac{m}{2} \rfloor + 1$. So, the i 'th leaf is in position $\lfloor \frac{m}{2} \rfloor + 1 + i - (m - k + 1) - 1 = \lfloor \frac{2n-1}{2} \rfloor - (2n - 1) + j = j - n$.

The non leaf nodes of a segmented winner tree are used to maintain winner trees. When we have only one segment, we get a winner tree as defined in [2]. However, when there is more than one segment, several winner trees are embedded in $t[...]$. Consider the eleven element example of Figure 1. Segment 1 is represented by two winner trees comprised of the nodes labeled a and b in Figure 4. The nodes labeled a form one tree and the node labeled b forms the other. The nodes labeled c , d , and e , respectively, form the three winner trees that represent segment 2 while the nodes labeled f form the single winner tree that represents segment 3. Note that each winner tree embedded in a segmented winner tree is a full binary tree.

