Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 161 (2006) 175–183

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Decision Trees of Algorithms and a Semivaluation to Measure Their Distance

M. O' Keeffe [1], H. Pajoohesh [2], M. Schellekens [3]

Centre for Efficiency Oriented Languages (CEOL) [4]
Department of Computer Science, University College Cork, Ireland

Abstract

We use the set T_n of binary trees with n leaves to study decision trees of algorithms. The set T_n of binary trees with n leaves can be ordered by the so called "imbalance" order, where two trees are related in the order iff the second is less "balanced" than the first. This order forms a lattice. We show that this lattice is nonmodular and extend the imbalance lattice with an algebraic operation. The operation corresponds to the extension of a binary tree with new binary trees at the leafs, which reflects the effect of recursive calls in an algorithm on the decision tree and we will characterize as an illustration the decision tree of the insertion sort algorithm.
We investigate the semivaluations on the binary trees which is related to the running time of the algorithm.

Keywords: Decision trees, sorting algorithms, level of balance, time complexity, semivaluations

1 Introduction

Parker and Ram introduced the (im)balance lattice of binary trees in [6]. They exploit the lattice structure to show that a cost-function for file compression is submodular and through this approach obtain a new proof of the optimality of the use of Huffman codes for optimal file compression.

We are interested here in studying the particular case of binary decision trees in the context of the imbalance lattice. We motivate this approach below.

In [1] it is remarked, in the context of Divide and Conquer algorithms, that algorithms which are "more balanced" than others will have better speed. However this intuitive notion of "more balanced" thus far has not been put on a formal basis.

[1] Funded by Science Foundation Ireland, Award 02/IN.1/181 and by Enterprise Ireland Research Scholarship BR/2001/141

[2] Funded by Boole Centre for Research in Informatics (BCRI), University College Cork

[3] Funded by Science Foundation Ireland, Award 02/IN.1/181

[4] Funded by Science Foundation Ireland, Investigator Award 02/IN.1/181

doi:10.1016/j.entcs.2006.04.032

A typical example of balance in relation to Divide and Conquer algorithms, considered in [1], is Mergesort in comparison with Insertion sort. Mergesort uses a Divide and Conquer approach by splitting lists in two (nearly) equal parts. Insertion sort, though typically not regarded as a Divide and Conquer algorithm, can be viewed as a Divide and Conquer algorithm which splits a list up in two unequal parts: a sorted list and a single element to be inserted in the sorted list. Since insertion sort is "less balanced" in its approach to Divide and Conquer than mergesort, it is argued in [1] that this implies that the speed of the Mergesort is better than the speed of Insertion Sort.

This has motivated our study of the (im)balance lattice in the context of binary decision trees. Each comparison-based algorithm [5] allows for the representation of its running time via a binary decision tree (cf. [1]). Hence it is natural to study the level of balance of such algorithms in terms of the level of balance of decision trees and to study the implications w.r.t. running time behaviour. This has been the topic of [5] where it is shown that the fact that Mergesort has a more balanced decision tree than insertion sort, in the sense of Parker-Ram's (im)balance lattice, implies that the speed of the first algorithm is better than the speed of the second. Here we continue to study algebraic operations which reflect the effect of recursive calls of algorithms on decision trees.

1.1 Preliminaries

We recall some definitions from [6]:

Definition 1.1 Binary trees are reversed trees with a root node, in which every internal node has exactly two children. The order of the leaves is insignificant, so a tree is determined (up to permutation on the leaves) by the lengths of the paths from the root node to each leaf (the distance of the leaf from the root). Thus we can represent equivalence classes of the rooted binary trees with n leaves by increasing sequences of n non-negative integers, which give the path-length of each leaf.

Remark 1.2 If we denote the set of binary trees with n leaves by T_n, $< x_1, ..., x_n > \in T_n$ if and only if $\sum_{i=1}^{n} 1/2^{x_i} = 1$.

Definition 1.3 Balancing exchanges transform a path-length sequence from $< \dots p \dots (q+1)\,(q+1) \dots >$ to $< \dots (p+1)\,(p+1) \dots q \dots >$. Minimal balancing exchanges are a special type of balancing exchange where we require that $(p+1) = q$.

Definition 1.4 Given two sequences $x, y \in T_n$, we define x to be at least as balanced as y, $x \sqsubseteq y$, if there are sequences $l_1, ..., l_m (m \geq 1) \in T_n$ such that $y = l_1, x = l_m$ and for each i, $1 \leq i < m$, there is a balancing exchange from l_i to l_{i+1}.

Definition 1.5 If $x =< x_1, ..., x_n >, y =< y_1, ..., y_n > \in T_n$ then we say that x is more balanced than y if $\sum_{i=1}^{n} x_i \leq \sum_{i=1}^{n} y_i$, and we say the level of balance of x is

[5] i.e. an algorithm for which every action is ultimately based on a comparison between two elements.

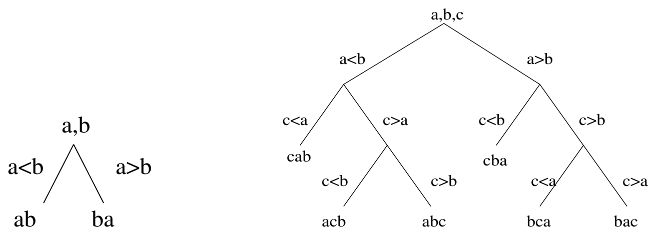

Fig. 1. Decision trees for insertion sort on lists of size 2 and 3 respectively

$(n-1)(n+2)/2 - \sum_{i=1}^{n} x_i$ and denote it by $l(x)$.

Proposition 1.6 *T_n with the above order is a lattice (See [6]).*

Theorem 1.7 *For $n \geq 8$, T_n is not a modular lattice and hence is not distributive.*

Proof. It is enough to show that, when $n \geq 8$ then T_n has a copy of N_5. Consider the sequence $A =< 1...(n-7)(n-5)(n-5)(n-4)(n-4)(n-4)(n-3)(n-3) >$. Thus by minimal ternary exchanges on A we get the following sequences: $D =< 1...(n-7)(n-5)(n-4)(n-4)(n-4)(n-4)(n-4)(n-4) >$ and $B =< 1...(n-6)(n-6)(n-6)(n-4)(n-4)(n-4)(n-3)(n-3) >$. Actually there are only two minimal ternary exchanges on A. Now we have a minimal ternary exchange on B, which gives the following sequence: $C =< 1...(n-6)(n-6)(n-5)(n-5)(n-5)(n-4)(n-3)(n-3) >$. Now we show that $C \wedge D$ is the following sequence: $E =< 1...(n-6)(n-6)(n-5)(n-5)(n-4)(n-4)(n-4)(n-4) >$. First of all since with ternary exchanges we come from C, D to E, $E \sqsubseteq C$ and $E \sqsubseteq D$. By looking at C, D, we see that the difference between C and D is that C has one copy of $(n-4)$ but D has 6 copies of this. So what is the form of E?

The only element $K \in T_n$ that we can get from both C, D via ternary exchanges is what we get by reducing $(n-4)$ in D to 4 copies and increasing it in C to 4 copies and the only K that we get is E, so $E = C \wedge D$. □

2 The structure of the decision tree for insertion sort

The algorithm insertion sort is so called because on the i^{th} pass it "inserts" the i^{th} element $L[i]$ into its rightful place among $(L[1], L[2], \ldots, L[i-1])$, which were previously placed in sorted order [1]. So essentially, insertion sort is always inserting one element into a sorted sublist which continues to grow in size with each iteration of the algorithm.

In order to generalise the structure of the decision tree for insertion sort, we first need some intuition on how the decision tree for a list of size $n-1$, $DT_I(n-1)$, can be used to generate the decision tree for a list of size n, $DT_I(n)$. The decision trees for insertion sort on lists of size 2 and 3 are given in Figure 1.

If we consider insertion sort on a list of size 4 as an example we will be able to see that $DT_I(4)$ is made up of the decision trees $DT_I(2)$ and $DT_I(3)$. We let our input list L contain the elements $\{a, b, c, d\}$. In order to produce the decision tree $DT_I(4)$, we firstly produce the decision tree $DT_I(2)$ shown in Figure 1 by comparing the first two list elements a and b. We then insert c into the sorted sublist represented

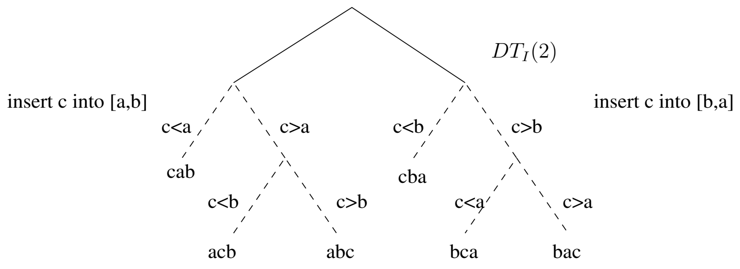

Fig. 2. Decision tree for insertion sort on list of size 3

by $DT_I(2)$ producing the decision tree of size 3, $DT_I(3)$. We can see from Figure 2 that $DT_I(3)$ consists of $DT_I(2)$ with some new structure attached to its leafs. When we insert the final element d into the sorted sublist represented by $DT_I(3)$ we obtain $DT_I(4)$ which, as expected, consists of the previous decision tree, $DT_I(3)$, with a new structure attached to its leafs as shown in Figure 3. So it appears that in general we can produce the decision tree of size n by adding a new structure onto every leaf of the decision tree for size $n-1$. So we really have that:

$$\begin{aligned} DT_I(4) &= DT_I(3) \oplus \text{ new structure} \\ &= (DT_I(2) \oplus \text{ new structure}) \oplus \text{ new structure} \end{aligned}$$

In order to produce the decision tree of size n, we first produce the decision tree of size $n-1$. At this point in the algorithm, we are trying to insert the last remaining element into a sorted sublist of size $n-1$. There are n different possibilities for the value of this last element to be inserted. Depending on its value, it will take between 1 and $n-1$ comparisons to insert it into its correct position. For example, if it is the smallest element in the original list then it will take one comparison to insert. If it is the second smallest element it will take two comparisons to insert. If it is the the second largest or largest element it will take $n-1$ comparisons to insert as it will need to be compared against each element of the sublist. So the values of the comparisons made at the final stage of the algorithm are $1, 2, \ldots, n-3, n-2, n-1, n-1$. Note that these values actually represent the path-length sequence of the most imbalanced tree of size n which we will call $MI(n)$.

So, the decision tree for insertion sort on a list of size n can be defined recursively as:

$$DT_I(n) = DT_I(n-1) \oplus MI(n)$$

where $\oplus$ represents an operation which adds a subtree to each leaf of an existing tree.

We can generate the path-length sequence for this decision tree by simply adding to each leaf of $DT_I(n-1)$ the most imbalanced sequence of size n, $MI(n)$, as follows:

$$\begin{aligned} DT_I(2) &= <1\,1> \\ DT_I(3) &= DT_I(2) \oplus MI(3) \end{aligned}$$

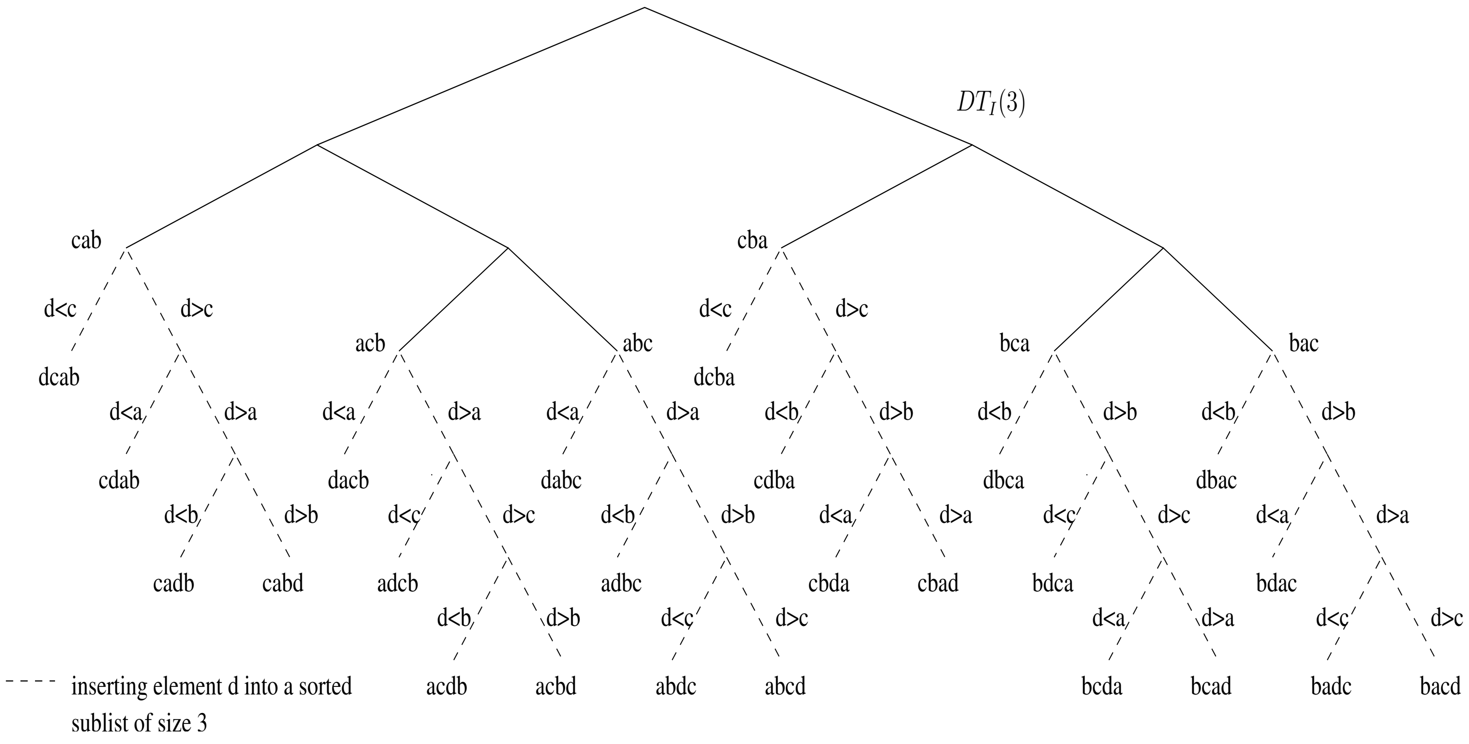

Fig. 3. Decision tree for insertion sort on list of size 4

$$\begin{aligned}
&= <1\,1> \oplus <1\,2\,2>\\
&= <(1+1)\,(1+2)\,(1+2)\,(1+1)\,(1+2)\,(1+2)>\\
&= <2\,3\,3\,2\,3\,3>\\
DT_I(4) &= DT_I(3) \oplus MI(4)\\
&= <2\,3\,3\,2\,3\,3> \oplus <1\,2\,3\,3>\\
&= <(2+1)\,(2+2)\,(2+3)\,(2+3)\,(3+1)\,\ldots\,(3+3)>\\
DT_I(4) &= <3\,4\,5\,5\,4\,5\,6\,6\,4\,5\,6\,6\,3\,4\,5\,5\,4\,5\,6\,6\,4\,5\,6\,6>\\
&\vdots\\
DT_I(n) &= DT_I(n-1) \oplus MI(n)
\end{aligned}$$

where $\oplus$ adds to each element of $DT_I(n-1)$ the elements of $MI(n)$.

2.1 Formalising the decision tree of insertion sort

We will now analyse and formalise the operation $\oplus$. We take the following definitions from [2].

Definition 2.1 A po-groupoid (or m-poset) is a poset M with a binary multiplication which satisfies the isotonicity condition

(1) $$a \leq b \text{ implies } xa \leq xb \text{ and } ax \leq bx$$

for all $a, b, x \in M$. When multiplication is commutative or associative, M is called a commutative po-groupoid or po-semigroup, respectively. A po-semigroup with identity 1, such that $x1 = 1x = x$ for all $x \in M$ is called a partially ordered monoid, or po-monoid. A po-group is $(G, +, \leq)$ such that $(G, +)$ is a group which satisfies (1). An l-group is a po-group such that $(G, \leq)$ is a lattice.

Proposition 2.2 *If X is a m-poset (po-semigroup, po-monoid, po-group or l-group). Then*

(i) *X^n with coordinate multiplication and coordinate order is a m-poset (po-semigroup, po-monoid, po-group or l-group).*

(ii) *In a po-monoid, $x, y \geq 1$ implies $xy \geq 1$.*

Remark 2.3 In the m-poset X, for $(z_1, ..., z_n) \in X^n$ where X is totally ordered, $[(z_1, ..., z_n)]$ means that it is ordered.

If $(X, +)$ is a po-semigroup, then we define $+_{m,n} : X^m \times X^n \to X^{mn}$, $m, n \in \{1, 2, ...\}$ such that $(x_1, ..., x_m) +_{m,n} (y_1, ..., y_n) = [(z_{ij})]$, $z_{ij} = x_i + y_j$, where $i = 1, ..., m$ and $j = 1, ..., n$.

Theorem 2.4 *Let X be a totally ordered semigroup. Then*

(i) $(x_1, ..., x_m) +_{m,np} ((y_1, ..., y_n) + (z_1, ..., z_p)) = ((x_1, ..., x_m) +_{m,n} (y_1, ..., y_n))$ $+_{mn,p} (z_1, ..., z_p)$

(ii) *If X is commutative then* $(x_1, ..., x_m) +_{m,n} (y_1, ..., y_n) = (y_1, ..., y_n)$ $+_{n,m} (x_1, ..., x_m)$

(iii) *If* $(x_1, ..., x_m) \leq (t_1, ..., t_m)$ *then* $(x_1, ..., x_m) +_{m,n} (y_1, ..., y_n) \leq (t_1, ..., t_m) +_{m,n} (y_1, ..., y_n)$

(iv) *If* $(r_1, ..., r_n) \leq (y_1, ..., y_n)$ *then* $(x_1, ..., x_m) +_{m,n} (r_1, ..., r_n) \leq (x_1, ..., x_m) +_{m,n} (y_1, ..., y_n)$

Proof. We prove only the first part. Notice that $(x_1, ..., x_m) +_{m,np} ((y_1, ..., y_n) + (z_1, ..., z_p)) = [(x_1+s_{11}, ..., x_1+s_{np}, ..., x_m+s_{11}, ..., x_m+s_{np})]$, such that $s_{ij} = y_i+z_j$, where $i = 1, ..., n$, $j = 1, ..., p$, and $((x_1, ..., x_m) +_{m,n} (y_1, ..., y_n))$ $+_{mn,p} (z_1, ..., z_p) = [(t_{11} + z_1, ..., t_{mn} + z_1, ..., t_{11} + z_p, ..., t_{mn} + z_p)]$ such that $t_{ij} = x_i + y_j$, where $i = 1, ..., m$, $j = 1, ..., n$.

Now one can easily show that the equality holds.

□

Theorem 2.5 *Let X be a m-poset and let* $(a_1, ..., a_m)+_{m,n}(b_1, ..., b_n) = (z_1, ..., z_{mn})$. *Then* $\sum_{i=1,j=1}^{m,\, n} z_{ij} = n\sum_{i=1}^{m} a_i + m\sum_{i=1}^{n} b_i$.

Proof. $\sum_{i=1,j=1}^{m,\, n} z_{ij} = \sum_{i=1}^{m}\sum_{j=1}^{n}(a_i + b_j) = \sum_{i=1}^{m}(\sum_{j=1}^{n}(a_i + b_j)) = \sum_{i=1}^{m}(na_i + \sum_{j=1}^{n} b_j) = n\sum_{i=1}^{m} a_i + \sum_{i=1}^{m}\sum_{j=1}^{n} b_j = n\sum_{i=1}^{m} a_i + m\sum_{i=1}^{n} b_i$.

□

Remark 2.6 The path-length sequence for the decision tree of insertion sort on a list of size 2 is $< 1\,1 >$. In order to obtain the path-length sequence for the decision tree of size 3 we must add the sequence $< 1\,2\,2 >$ to every leaf of the decision tree of size 2. So we get $(< 1\,1 > +_{2,3} < 1\,2\,2 >) =< 2\,2\,3\,3\,3\,3 >$ which is the decision tree of size 3. In order to obtain the decision tree for size 4 we add $< 1\,2\,3\,3 >$ to every leaf of the decision tree of size 3 and we get the following: $(< 2\,2\,3\,3\,3\,3 > +_{6,4} < 1\,2\,3\,3 >) =< 3\,3\,4\,4\,4\,4\,4\,4\,5\,5\,5\,5\,5\,5\,5\,5\,6\,6\,6\,6\,6\,6\,6\,6 >$. From these we obtain the following corollary.

Lemma 2.7 *The path-length sequence of the decision tree for insertion sort on an input list of size n is* $< 1\,1 > +_{2,3} \cdots +_{(n-1)!,n} < 1\,2 ... (n-1)\,(n-1) >$. *This sequence begins with* $n - 1$ *which repeats twice and terminates with* $n(n - 1)/2$.

Proof. We sketch the proof and proceed by induction. The result holds because when we attach a binary tree y with n leaves to the binary tree x with m leaves the obtained tree is $x +_{m,n} y$. □

Theorem 2.8 *The level of balance of the decision tree for insertion sort on an input list of size n is* $(n! - 1)(n! + 2)/2 - \sum z_i$, *where* $(z_i) =< 1\,1 > +_{2,3} \cdots +_{(n-1)!,n} < 1\,2 ... (n-1)\,(n-1) >$.

Proof. By the previous corollary and definition of level of balance, one can easily see that the equality is true. □

3 Semivaluations

We can interpret a binary tree as a ∨-semilattice and conversely define a binary tree as a semilattice with some additional properties. This leads to a characterization of binary trees in terms of semilattices. Our interest in binary trees stems from the fact that in order to carry out the running time analysis of so-called *comparison-based algorithms*, i.e., algorithms in which every action is ultimately based on a prior comparison between two elements, the notion of a decision tree is a fundamental tool [1]. Typical examples are so-called *Sorting Algorithms* which sort a given list of numbers in increasing order. Decision trees are binary trees representing the comparisons carried out during the computation of a comparison-based algorithm. The distance from the root (*input*) to a leaf (*output*) gives the comparison time for the algorithm (i.e., the total number of comparisons) to compute the output corresponding to the given leaf.

Definition 3.1 If $(X, \preceq)$ is a join semilattice then a function $f\colon (X, \preceq) \to \mathcal{R}_0^+$ is a *join valuation* iff

$$\forall x, y, z \in X.\, f(x \sqcup z) \leq f(x \sqcup y) + f(y \sqcup z) - f(y)$$

and f is *join co-valuation* iff

$$\forall x, y, z \in X.\, f(x \sqcup z) \geq f(x \sqcup y) + f(y \sqcup z) - f(y).$$

Remark 3.2 We can consider every binary tree as a join semilattice.

Proposition 3.3 *The distance from any node of a binary tree to the root is a join co-valuation.*

Proof. We sketch the proof. We proceed by induction on the number of leaves. It suffices to show that if $T \in T_n$, where T_n is the set of binary trees with n leaves, and $x, y, z \in T$ then

$$l(x \vee z) \geq l(x \vee y) + l(y \vee z) - l(y).(*)$$

Here $l(x)$, the level of x, is the distance from x to the root.

We show that $(*)$ holds for leaves. The result for general nodes then follows. This is true for $n = 2$. We assume that this is true for every k less than n and we have to show it still holds for n. Notice that if the level of none of them is the maximum level then we can remove this maximum level and by induction (*) is true for x, y, z.

Now if we have some leaves at levels other than the levels of x, y and z, we can remove the leaves by use of induction. Thus the problem is reduced to the case where there are no other levels except $l(x), l(y)$ and $l(z)$.

On the other hand notice that if $x \vee y \vee z$ is not the root then again we can reduce the problem by induction and prove that (*) is true. So we assume that this join is the root. We have to investigate 3 cases:

1) Three of them are at the same level.

2) Two of them are at the same level and the last one is on a different level.

3) They are on 3 pairwise different levels.

We prove (*) in the first case and for the other cases the proof is similar.

Since three of them are at the same level they have to be in the last level. Notice that the tree divides via two branches from the root in two subtrees. Since the join of the three elements is the root, three of them can't be in the same half, so two of them are in the same half and the other is the other half. Now we have to investigate two cases, x, y are in the same half and z is in the other half and the second case where x, z are in the same half and y is in another half.

If x, y are in the same half and z is in the other half then both $x \vee z$ and $y \vee z$ are the root and so $l(x \vee z) = l(y \vee z) = 0$ and since l is order reversing $l(x \vee y) \geq l(y)$ and (*) trivially holds.

Also if x, z be in the same half and y in another half then the argument is similar and the proof is complete. □

References

[1] Aho, A., J. Hopcroft and J. Ullman, "Data Structures and Algorithms", Addison-Wesley Series in Computer Science and Information Processing, Addison-Wesley, 1983.

[2] Birkhoff, G., "Lattice Theory", American Mathematical Society Colloquium Publications, Volume XXV, 1967.

[3] Knuth, D., "The Art of Computer Programming", vol. **1**, Addison-Wesley, 1973.

[4] Knuth, D., "The Art of Computer Programming", vol. **3**, Addison-Wesley, 1973.

[5] O' Keeffe, M., H. Pajoohesh and M. Schellekens, *The relationship between balance and the speed of algorithms*, Hadronic Journal, accepted for publication, to appear in Vol. **28** (2005).

[6] Parker, D. Stott and Prasad Ram, *The construction of Huffman codes is a submodular ("convex") optimization problem over a lattice of binary trees*, SIAM Journal of Computing, vol **28** (1999), No.5, pp.1875-1905.

[7] Romaguera, S., and M. Schellekens, *Duality and quasi-normability for complexity spaces*, Applied General Topology, vol **3** (2002), nr. 1.

[8] O'Keeffe, M., S. Romaguera and M. Schellekens, *Norm-weightable Riesz spaces and the dual complexity space*, ENTCS, volume **74** (2003), Elsevier, Proceedings MFCSIT2002.

[9] Romaguera, S., and M. Schellekens, *Partial metric monoids and semivaluation spaces*, Topology and its Applications, Elsevier, accepted for publication, to appear.

[10] Schellekens, M. P., *The correspondence between partial metrics and semivaluations*, Theoretical Computer Science, to appear.