A Tight Threshold Bound for Search Trees with 2-Way Comparisons

Sunny Atalig and Marek Chrobak(✉)

University of California at Riverside, Riverside, USA
marek@cs.ucr.edu

Abstract. We study search trees with 2-way comparisons (2WCST's), which involve separate less-than and equal-to tests in their nodes, each test having two possible outcomes, yes and no. These trees have a much subtler structure than standard search trees with 3-way comparisons (3WCST's) and are still not well understood, hampering progress towards designing an efficient algorithm for computing minimum-cost trees. One question that attracted attention in the past is whether there is an easy way to determine which type of comparison should be applied at any step of the search. Anderson, Kannan, Karloff and Ladner studied this in terms of the ratio between the maximum and total key weight, and defined two threshold values: λ^- is the largest ratio that forces the less-than test, and λ^+ is the smallest ratio that forces the equal-to test. They determined that $\lambda^- = \frac{1}{4}$, but for the higher threshold they only showed that $\lambda^+ \in [\frac{3}{7}, \frac{4}{9}]$. We give the tight bound for the higher threshold, by proving that in fact $\lambda^+ = \frac{3}{7}$.

1 Introduction

Search trees are decision-tree based data structures used for identifying a query value within some specified set $\mathcal{K}$ of keys, by applying some simple tests on the query. When $\mathcal{K}$ is a linearly ordered set, these tests can be comparisons between the query and a key from $\mathcal{K}$. In the classical model of 3-way comparison trees (3WCST's), each comparison has three outcomes: the query can be smaller, equal, or greater than the key associated with this node. In the less studied model of search trees with 2-way comparisons (2WCST's), proposed by Knuth [7, §6.2.2, Example 33], we have separate less-than and equal-to tests, with each test having two outcomes, yes or no. In both models, the search starts at the root node of the tree and proceeds down the tree, following the branches corresponding to these outcomes, until the query ends up in the leaf representing the key equal to the query value[1].

[1] Here we assume the scenario when the query is in $\mathcal{K}$, often called the *successful-query* model. If arbitrary queries are allowed, the tree also needs to have leaves representing inter-key intervals. Algorithms for the successful-query model typically extend naturally to this general mode without increasing their running time.

[2] There is of course vast amount of research on the dynamic case, when the goal is to have the tree to adapt to the input sequence, but it is not relevant to this paper.

Research supported by NSF grant CCF-2153723.

X. Chen and B. Li (Eds.): TAMC 2024, LNCS 14637, pp. 99–110, 2024.
https://doi.org/10.1007/978-981-97-2340-9_9

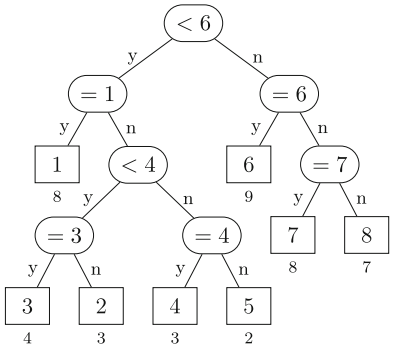


Fig. 1. An example of a 2WCST. This tree handles keys 1, 2, 3, 4, 5, 6, 7, 8 with respective weights 8, 3, 4, 3, 2, 9, 8, 7. Computing the cost in terms of leaves, we get $8 \cdot 2 + 3 \cdot 4 + 4 \cdot 4 + 3 \cdot 4 + 2 \cdot 4 + 9 \cdot 2 + 8 \cdot 3 + 7 \cdot 3 = 127$.

The focus of this paper is on the static scenario, when the key set $\mathcal{K}$ does not change over time[2]. Each key k is assigned a non-negative weight w_k, representing the frequency of k, or the probability of it appearing on input. Given these weights, the objective is to compute a tree T that minimizes its cost, defined by $\mathrm{cost}(T) = \sum_{k \in \mathcal{K}} w_k \cdot \mathrm{depth}(k)$, where $\mathrm{depth}(k)$ is the depth of the leaf representing key k in T. This concept naturally captures the expected cost of a random query (Fig. 1).

Using the now standard dynamic programming algorithm, optimal 3WCST's can be computed in time $O(n^3)$, where $n = |\mathcal{K}|$ denotes the number of keys. As shown already by Knuth [6] in 1971 and later by Yao [10] in 1980, using a more general approach, the running time can be improved to $O(n^2)$. This improvement leverages the property of 3WCST's called the quadrangle inequality (which is essentially equivalent to the so-called Monge property or total monotonicity—see [3]). In contrast, the first (correct[3]) polynomial-time algorithm for finding optimal 2WCST's was developed only in 2002 by Anderson, Kannan, Karloff and Ladner [1]. Its running time is $O(n^4)$. A simpler and slightly more general $O(n^4)$-time algorithm was recently given in [4].

The reason for this disparity in the running times lies in the internal structure of 2WCST's, which is much more intricate than that of 3WCST's. Roughly, while the optimal cost function of 3WCST's has a dynamic-programming formulation where all sub-problems are represented by key intervals, this is not the case for 2WCST's. For 2WCST's, a similar approach produces intervals with holes (corresponding to earlier failed equal-to tests), leading to exponentially many sub-problems. As shown in [1] (and conjectured earlier by Spuler [8,9]), this can be reduced to $O(n^3)$ sub-problems using the so-called heaviest-first key property.

One other challenge in designing a faster algorithm is that, for any sub-problem, it is not known a priori whether the root should use the less-than test or the equal-to test. The intuition is that when some key is sufficiently heavy then the optimum tree must start with the equal-to test (to this key) at the root. On the other hand, if all weights are

[3] Anderson et al. [1] reference an earlier $O(n^5)$-time algorithm by Spuler [8,9]. However, as shown in [5], Spuler's proof is not correct.

roughly equal (and there are sufficiently many keys), then the tree should start with a less-than test, to break the instance into two parts with roughly the same total weight. Addressing this, Anderson et al. [1] introduced two threshold values for the maximum key weight. Denoting by W the total weight of the keys in the instance, these values are defined as follows:

- λ^- is the largest λ such that if all key weights are smaller than λW then there is no optimal tree with an equal-to test at the root.
- λ^+ is the smallest λ such that if any key has weight at least λW then there is an optimal tree with an equal-to test at the root.

In their paper[4], they proved that $\lambda^- = \frac{1}{4}$ and $\lambda^+ \in [\frac{3}{7}, \frac{4}{9}]$. These thresholds played a role in their $O(n^4)$-time algorithm for computing optimal 2WCST's. The more recent $O(n^4)$-time algorithm in [4] uses a somewhat different approach and does not rely on any threshold bounds on key weights.

Nevertheless, breaking the $O(n^4)$ barrier will require deeper understanding of the structure of optimal 2WCST's; in particular, more accurate criteria for determining which of the two tests should be applied first are likely to be useful. Even if not improving the asymptotic complexity, such criteria reduce computational overhead by limiting the number of keys to be considered for less-than tests.

With these motivations in mind, in this work we give a tight bound on the higher weight threshold, by proving that the lower bound of $\frac{3}{7}$ for λ^+ in [1] is in fact tight.

Theorem 1. *For all $n \geq 2$, if an instance of n keys has total weight W and a maximum key weight at least $\frac{3}{7}W$ then there exists an optimal* 2WCST *rooted at an equal-to test. In other words, $\lambda^+ \leq \frac{3}{7}$.*

The proof is given in Sect. 3, after we introduce the necessary definitions and notation, and review fundamental properties of 2WCST's, in Sect. 2.

Note: For interested readers, other structural properties of 2WCST's were recently studied in the companion paper [2]. In particular, that paper provides other types of threshold bounds, including one that involves two heaviest keys, as well as examples showing that the speed-up techniques for dynamic programming, including the quadrangle inequality, do not work for 2WCST's.

2 Preliminaries

Notation. Without any loss of generality we can assume that set of keys is $\mathcal{K} = \{1, 2, ..., n\}$, and their corresponding weights are denoted $w_1, w_2, \ldots, w_n$. Throughout the paper, we will typically use letters $i, j, k, ...$ to represent keys. The total weight of the instance is denoted by $W = \sum_{k \in \mathcal{K}} w_k$. For a tree T by $w(T)$ we denote the total weight of keys in its leaves, calling it the *weight of* T. For a node v in T, $w(v)$ denotes the weight of the sub-tree of T rooted at v.

[4] In [1] the notation for the threshold values λ^- and λ^+ was, respectively, λ and μ. We changed the notation to make it more intuitive and consistent with [2].

Each internal node is either an equal-to test to a key k, denoted by $\langle = k\rangle$, or a less-than test to k, denoted by $\langle < k\rangle$. Conventionally, the left and right branches of the tree at a node are labelled by "yes" and "no" outcomes, but in our proof we will often depart from this notation and use relation symbols "$=$", "$\neq$", "$<$", etc., instead. For some nodes only the comparison key k will be specified, but not the test type. Such nodes will be denoted $\langle * k\rangle$, and their outcome branches will be labeled "$=/\geq$" and "$\neq/<$". The interpretation of these is natural: If $\langle * k\rangle$ is $\langle = k\rangle$ then the first branch is taken on they "yes" answer, otherwise the second branch is taken. If $\langle * k\rangle$ is $\langle < k\rangle$ then the second branch is taken on the "yes" answer and otherwise the first branch is taken. This convention will be very useful in reducing the case complexity.

A branch of a node is called *redundant* if there is no query value $q \in \mathcal{K}$ that traverses this branch during search. A 2WCST T is called *irreducible* if it does not contain any redundant branches. Each tree can be converted into an irreducible tree by "splicing out" redundant branches (linking the sibling branch directly to the grandparent). This does not increase cost. So throughout the paper we tacitly assume that any given tree is irreducible. In particular, note that any key k appearing in a node $\langle * k\rangle$ of an irreducible tree must satisfy all outcomes of the tests on the path from the root to $\langle * k\rangle$. (The only non-trivial case is when $\langle * k\rangle$ is $\langle < k\rangle$ and there is a node $\langle = k\rangle$ along this path. In this case, $\langle < k\rangle$ can be replaced by $\langle < k+1\rangle$, as in this case k cannot be n if T is irreducible.)

Side Weights. We use some concepts and auxiliary lemmas developed by Anderson et al. [1]. In particular, the concept of side-weights is useful. The *side-weight* of a node v in a 2WCST T is defined by

$$sw(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf} \\ w_k & \text{if } v \text{ is an equal-to test } \langle = k\rangle \\ \min\{w(L), w(R)\} & \text{if } v \text{ is a less-than test with sub-trees } L, R \end{cases}$$

Lemma 1. [1] *Let T be an optimal* 2WCST. *Then $sw(u) \geq sw(v)$ if u is a parent of v.*

The lemma, while far from obvious, can be proved by applying so-called "rotations" to the tree, which are local rearrangements that swap some adjacent nodes. As a simple example, if $u = \langle = k\rangle$ is a child of $v = \langle = l\rangle$ and $sw(u) > sw(v)$ then exchanging these two comparisons would reduce the tree cost, contradicting optimality. For the complete proof, see [1].

Lemma 1 implies immediately the following:

Lemma 2. [1] *For $n > 2$, if an optimal* 2WCST *for an instance of n keys is rooted at an equal-to test on i, then i is a key of maximum-weight.*

We remark that in case of ties between maximum-weight keys a subtlety arises that led to some complications in the algorithm in [1]. It was shown later in [4] that this issue can be circumvented. This issue does not arise in our paper, and the above statement of Lemma 2 is sufficient for our argument.

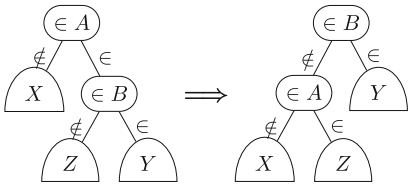
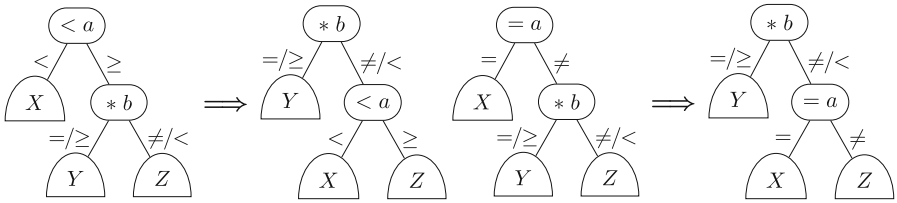


Fig. 2. A tree rotation using general queries.



(a) Valid if tree is irreducible.

(b) Valid if $a < b$ or $\langle * \; b\rangle = \langle = b\rangle$.

Fig. 3. Rotating a node with double outcomes.

More About Rotations. We will extend the concept of rotations to involve nodes of the form $\langle * \; k\rangle$, with unspecified tests. This makes the rotations somewhat non-trivial, since such nodes represent two different possible subtrees, and we need to justify that the tree obtained from the rotation is correct in both cases.

To give a simple condition for ensuring valid rotations, we generalize 2WCSTs by considering arbitrary types of tests, not merely equal-to or less-than tests. Any binary test can be identified by the set of keys that satisfy the "yes" outcome. We can thus represent such a test using the set element relation, denoted $\langle \in A\rangle$, with branches labelled "$\in$" and "$\notin$". An equal-to test $\langle = k\rangle$ can be identified by the singleton set $\{k\}$ or its complement $\mathcal{K} - \{k\}$, and a less-than test $\langle < k\rangle$ can be identified by $(-\infty, k)$ or $[k, \infty)$.

Consider the rotation in Fig. 2, where we denote the sub-trees and the keys they contain by X, Y, and Z. Comparing the trees on the left and the right, we must have $Y = A \cap B = B$, which implies that for the rotation to be correct we need $B \subseteq A$. (If the tree on the left is irreducible, the containment must in fact be strict.) On the other hand, if $B \subseteq A$ then in both trees we have $Z = A \setminus B$ and $X = \mathcal{K} - A$. This gives us the following property:

Containment Property: The rotation shown in Fig. 2 is valid if and only if $B \subseteq A$.

We remark that other rotations, such as when $\langle \in B\rangle$ is in the $\notin$-branch of $\langle \in A\rangle$, can be accounted for by the above containment property, by replacing $\langle \in A\rangle$ with $\langle \in \bar{A}\rangle$, where $\bar{A}$ is the complement of A.

Consider Fig. 3a, with tests $\langle < a\rangle$ and $\langle * \; b\rangle$. Assuming the original tree is irreducible, we have $b \geq a$, with strict inequality if $\langle * \; b\rangle$ is a less-than test. We identify $\langle < a\rangle$ with set $A = [a, \infty)$ and $\langle * \; b\rangle$ with $B = \{b\}$ or $[b, \infty)$ (second option corresponds to $\langle < b\rangle$). Then by our inequalities, it is clear that $B \subset A$ and the rotation is

valid. By a similar reasoning, the rotation shown in Fig. 3b is also valid, assuming either $a < b$ or $\langle * \; b\rangle$ is an equal-to test. For proving Theorem 1, we need only consider these two rotations (or the corresponding reverse rotations).

Not all tree modifications in our proof are rotations. Some modifications also *insert* a new comparison test into the tree. This modification has the effect of making a tree reducible, though it can be converted into a irreducible tree, as explained earlier in this section. Insertions are used by Anderson et al. [1] in proving the tight bound for λ^- and will also be used in proving Theorem 1.

3 Proof of Theorem 1

In this section we prove Theorem 1, namely that the lower bound of $\frac{3}{7}$ on λ^+ in [1] is tight.

Before proceeding with the proof, we remind the reader that $\langle * \; k\rangle$ denotes an unspecified comparison test on key k, and that its outcomes are specified with labels"$=/\geq$" and "$\neq/<$".

The proof is by induction on the number of nodes n. The cases $n = 2, 3$ are trivial so we'll move on to the inductive step. Assume that $n \geq 4$ and that Theorem 1 holds for all instances where the number of keys is in the range $\{2, \ldots, n-1\}$.

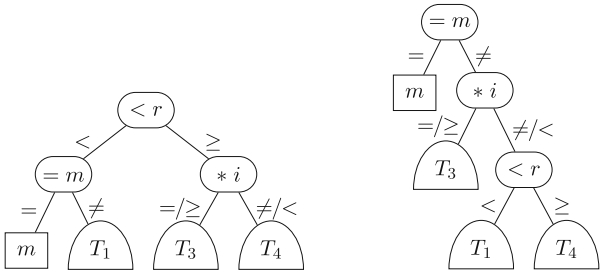
To show that the theorem holds for any n-key instance, consider a tree T for an instance with n keys and whose maximum-weight key m satisfies $w_m \geq \frac{3}{7}w(T)$, and suppose that the root of T is a less-than test $\langle < r\rangle$. We show that we can then find another tree T' that is rooted at an equal-to test node and has cost not larger than T.

If $r \in \{2, n\}$, then one of the children of $\langle < r\rangle$ is a leaf, and we can simply replace $\langle < r\rangle$ by an appropriate equal-to test. So we can assume that $2 < r \leq n-1$, in which case each sub-tree of $\langle < r\rangle$ must have between 2 and $n-1$ nodes. By symmetry, we also can assume that a heaviest-weight key m is in the left sub-tree L of $\langle < r\rangle$. Since $w_m \geq \frac{3}{7}w(T) \geq \frac{3}{7}w(L)$, we can replace L with a tree rooted at $\langle = m\rangle$ without increasing cost, by our inductive assumption. (A careful reader might notice that, if a tie arises, the inductive assumption only guarantees that the root of this left subtree will be an equal-to test to a key of the same weight as m. For simplicity, we assume that this key is m, for otherwise we can just use the other key in the rest of the proof.) Let T_1 be the $\neq$-branch of $\langle = m\rangle$ and T_2 be the $\geq$-branch of $\langle < r\rangle$.

By applying Lemma 1 to node $\langle = m\rangle$ and its parent $\langle < r\rangle$, we have that $w(T_2) \geq w_m \geq \frac{3}{7}w(T)$, which in turn implies that $w(T_1) \leq \frac{1}{7}w(T)$. Since T_2 is not a leaf, let its root be $\langle * \; i\rangle$ with sub-trees T_3 and T_4, where T_3 is the $=/\geq$-branch. The structure of T is illustrated in Fig. 4a.

To modify the tree, we break into two cases based on T_3's weight:

Case 1: $w(T_3) \geq \frac{1}{3}w(T_2)$ To obtain T', rotate $\langle = m\rangle$ to the root of the tree, then rotate $\langle * \; i\rangle$ so that it is the $\neq$-branch of $\langle = m\rangle$. (The irreducibility of T implies that the Containment Property in Sect. 2 holds for both rotations.) Node m goes up by 1 in depth and subtrees T_1 and T_4 both go down by 1 (see Fig. 4b). So the total change in cost is



(a) Original tree T (b) Modified tree T'

Fig. 4. On the left, tree T in the proof of Theorem 1. On the right, the modification for Case 1. Notice that T_3 is either the leaf i or contains all keys greater than or equal to i, depending on the comparison test $\langle * \, i \rangle$.

$$\begin{aligned}
\text{cost}(T') - \text{cost}(T) &= w(T_1) + w(T_4) - w_m \\
&= w(T) - 2w_m - w(T_3) && w_m + w(T_1) + w(T_3) + w(T_4) = w(T) \\
&\leq w(T) - 2w_m - \tfrac{1}{3}w_m && w(T_3) \geq \tfrac{1}{3}w(T_2), w(T_2) \geq w_m \\
&= w(T) - \tfrac{7}{3}w_m \\
&\leq 0 && w_m \geq \tfrac{3}{7}w(T)
\end{aligned}$$

Case 2: $w(T_3) < \frac{1}{3}w(T_2)$ Note that in this case, $w(T_4) > \frac{2}{3}w(T_2)$. We'll first handle some trivial cases. If T_4 is a leaf j, we can replace $\langle * \, i \rangle$ with $\langle = j \rangle$ and do the same rotations as in Case 1 (swapping the roles of T_4 and T_3). If T_4 contains only two leaves, say k and j, applying Lemma 1 we obtain that $w_k, w_j \leq w(T_3)$, which in turn implies that $w(T_2) \leq 3w(T_3) < w(T_2)$—a contradiction.

Therefore, we can assume that T_4 contains at least 3 leaves and at least 2 comparison nodes. Let $\langle * \, j \rangle$ be the test in the root of T_4. If $\langle * \, j \rangle$ is a less-than test and any of its branches is a leaf, we can replace $\langle * \, j \rangle$ with an equal-to test. If $\langle * \, j \rangle$ is an equal-to test than its $\neq$-branch is not a leaf (because T_4 has at least 3 leaves). Thus we can assume that the $\neq/<$-branch of $\langle * \, j \rangle$ is not a leaf, and let $\langle * \, k \rangle$ be the root of this branch. This means that both $\langle * \, j \rangle$ and $\langle * \, k \rangle$ follow from an $\neq/<$-outcome. Let T_5 be the $=/\geq$-branch of $\langle * \, j \rangle$ and T_6 and T_7 be the branches of $\langle * \, k \rangle$. The structure of T is shown in Fig. 6a.

This idea behind the remaining argument is this: We now have that T_5, T_6 and T_7 together are relatively heavy (because T_4 is), while T_1 and T_3, which are at lower depth, are light. If this weight difference is sufficiently large, it should be thus possible to rebalance the tree and reduce the cost. We will accomplish this rebalancing by using keys i, j and k, although it may require introducing a new less-than test to one of these keys.

For convenience, re-label the keys $\{i, j, k\} = \{b_1, b_2, b_3\}$ such that $b_1 \leq b_2 \leq b_3$. The goal is to use $\langle < b_2 \rangle$ as a "central" cut-point to divide the tree, with $\langle < r \rangle$ and $\langle * \, b_1 \rangle$ in its $<$-branch. The $\geq$-branch will contain $\langle * \, b_3 \rangle$ and, if $\langle * \, b_2 \rangle$ is an equal-to test, also $\langle * \, b_2 \rangle$.

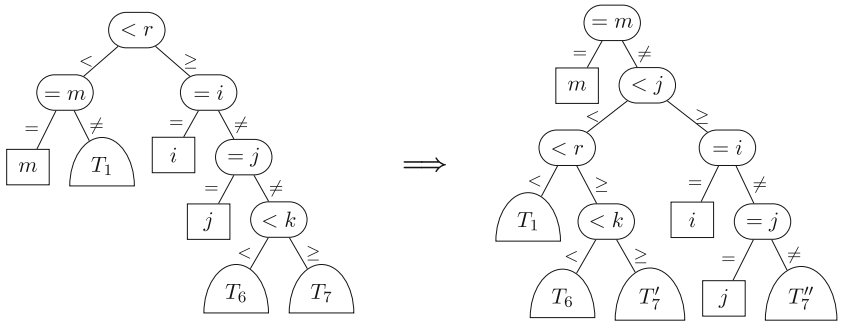


Fig. 5. An example conversion of T (on the left) into T' (on the right) in Case 2, where $k < j < i$. T'_7 and T''_7 are copies of T_7.

The idea is illustrated by the example in Fig. 5. In tree T' we have two copies of T_7, denoted T'_7 and T''_7. (Because of this T' is not irreducible. As explained in Sect. 2, T' can be then converted into an irreducible tree by splicing out redundant branches.) These copies are needed because the range of T_7 is partitioned by the $\langle < j \rangle$ test into two subsets, with query keys smaller than j following the $<$-branch and the other following the $\geq$-branch. We refer to this in text later as "fracturing" of T_7. However, since these subsets form a disjoint partition of the range of T_7, the total contribution of T'_7 and T''_7 to the cost of T' is the same as the contribution of T_7 to the cost of T. More generally, such fracturing does not affect the cost as long as the fractured copies of a subtree are at the same depth as the original subtree.

Ultimately, using the case assumption that $w(T_3) < \frac{1}{3}w(T_2)$, we want to show the following:

Claim 1: There exists a tree T' such that

$$\text{cost}(T') - \text{cost}(T) \;\leq\; w(T_1) - w_m + 2w(T_3).$$

In other words, we want to show that in the worst case scenario, we have a modified tree whose cost is at worst equivalent to moving key m up by one, T_1 down by one, and T_3 down by 2. This suffices to prove Case 2 as then

$$\begin{aligned}
\text{cost}(T') - \text{cost}(T) &\leq w(T_1) - w_m + 2w(T_3) \\
&= w(T) - 2w_m + w(T_3) - w(T_4) && w_m + w(T_1) + w(T_3) + w(T_4) = w(T) \\
&\leq w(T) - 2w_m + \tfrac{1}{3}w(T_2) - \tfrac{2}{3}w(T_2) && w(T_3) < \tfrac{1}{3}w(T_2), w(T_4) > \tfrac{2}{3}w(T_2) \\
&= w(T) - 2w_m - \tfrac{1}{3}w(T_2) \\
&\leq w(T) - 2w_m - \tfrac{1}{3}w_m && w(T_2) \geq w_m \\
&\leq 0
\end{aligned}$$

Before describing the construction of T', we'll first establish Claim 2 below.

Claim 2: $r < b_2$ (so that $\langle < r\rangle$ and $\langle < b_2\rangle$ are distinct tests).

Because keys i, j, and k are in the $\geq$-branch of $\langle < r\rangle$, we have that $r \leq b_1, b_2, b_3$. As any irreducible tree can perform at most two tests on the same key, if $b_i = r$ then b_i is distinct from the other two keys. Then $b_i = b_1$ must hold to preserve order, and thus $b_2 > b_1 = r$. This proves Claim 2.

The modified tree T' then has the following form: $\langle = m\rangle$ is at the root, $\langle < b_2\rangle$ is at the $\neq$-branch of $\langle = m\rangle$, and $\langle < r\rangle$ is at the $<$-branch of $\langle < b_2\rangle$. Notably, T_1 will still be in the $<$-branch of $\langle < r\rangle$ in T', which implies that m moves up by 1 and T_1 moves down by 1, thus matching the $w(T_1) - w_m$ terms in Claim 1. The right branch of $\langle < r\rangle$ leads to $\langle * b_1\rangle$. The rest of T' will be designed so that all subtrees T_3, T_5, T_6 and T_7 will have roots at depth at most 4, so in particular T_6 and T_7 will never move down.

Then it suffices to show that the new depths of T_3 and T_5 imply a cost increase no greater than $2w(T_3)$. More precisely, we will show that T' has one of the following properties: Compared to T, in T'

(j1) T_3 moves down by at most 2 and T_5's depth doesn't change, *or*
(j2) If $\langle * j\rangle$ is an equal-to test then T_5 and T_3 move down at most by 1 each. (This suffices because $w(T_5) = w_j \leq w(T_3)$ if $\langle * j\rangle$ is an equal-to test, by applying Lemma 1 to T.)

To describe the rest of our modification, we break into two sub-cases, depending on whether $\langle * b_2\rangle$ is and equal-to test or less-than test.

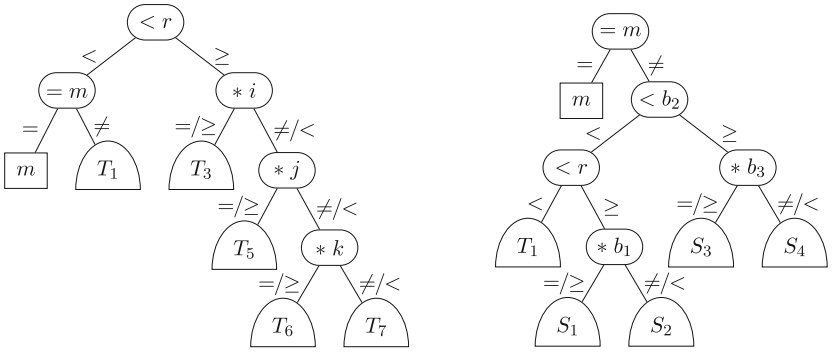
Case 2.1: $\langle * b_2\rangle = \langle < b_2\rangle$ In this case, $\langle * b_3\rangle$ is at the $\geq$-branch of $\langle < b_2\rangle$ in T'. We do not introduce any new comparison tests, obtaining T' shown in Fig. 6b. We now break into further cases based on which key b_2 is.

First, we observe that $b_2 \neq i$, as for $b_2 = i$ the structure of T would imply that $j, k < i$, meaning that i would be the largest key, instead of the middle one. Thus, $b_2 \in \{j, k\}$. This gives us two sub-cases.

Case 2.1.1: $b_2 = j$. Then we have $k < j$ by the structure of T, implying $k = b_1$, and $i \geq j$ since i must now be b_3. Then, in T', $\langle * b_1\rangle$ has branches T_6 and T_7, while $\langle * b_3\rangle$ has branches T_3 ($=/\geq$-branch) and T_5 ($\neq/<$-branch). In which case, only T_3 moves down by 1, satisfying (j1).

Case 2.1.2: $b_2 = k$. Then the structure of T implies that $k \neq \{i, j\}$, so $b_1 < k$, which in turn implies that $\langle * b_1\rangle$ is an equal-to test. We now have two further sub-cases. If $(i, j) = (b_1, b_3)$ then, in T', T_5 is in the $=/\geq$-branch of $\langle * b_3\rangle$ and leaf $T_3 = i$ is in the $=/\geq$-branch of $\langle * b_1\rangle$, implying T_5's depth doesn't change and T_3 moves down by 2, satisfying (j1). If $j = b_1$, then leaf $T_5 = j$ is below $\langle * b_1\rangle$ and T_3 is below $\langle * b_3\rangle$, both moving down by 1, satisfying (j2).

Case 2.2: $\langle * b_2\rangle = \langle = b_2\rangle$ and both tests $\langle * b_1\rangle$, $\langle * b_3\rangle$ are different from $\langle < b_2\rangle$. In this case, $\langle < b_2\rangle$ (the $\neq$-child of $\langle = m\rangle$ in T') is a newly introduced test. In T', we will have $\langle = b_2\rangle$ and $\langle * b_3\rangle$ in the $\geq$-branch of $\langle < b_2\rangle$. The order in which we perform $\langle = b_2\rangle$ and $\langle * b_3\rangle$ will be determined later.



(a) Original tree T (b) Modified tree T'

Fig. 6. Original and modified tree for Case 2.1. S_1, S_2, S_3, S_4 is simply some permutation of T_3, T_5, T_6, T_7.

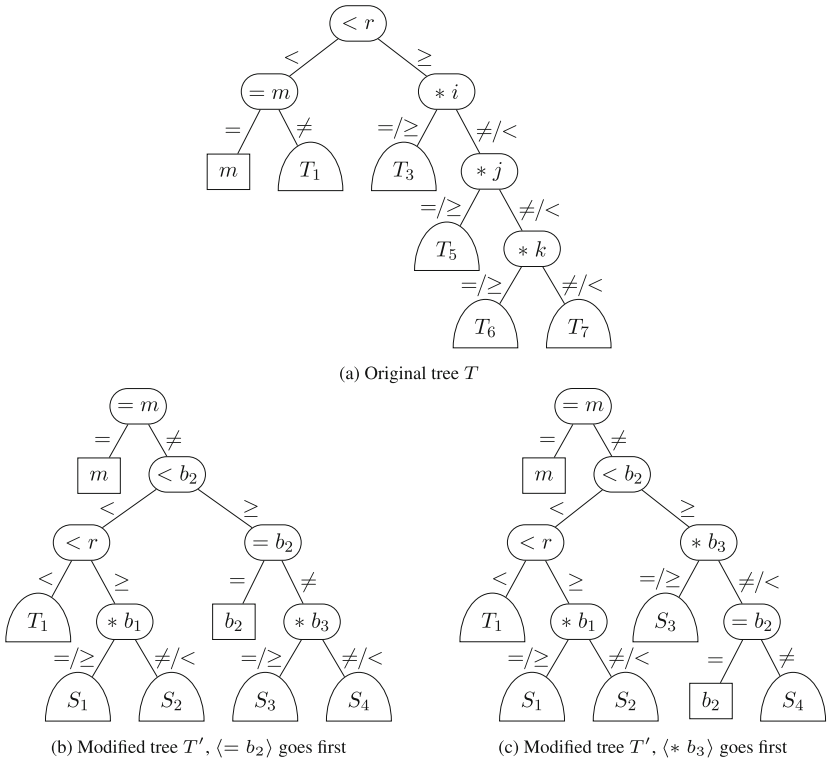
We will say $\langle = b_2 \rangle$ is "performed first" if it is the root of the $\geq$-branch of $\langle < b_2 \rangle$, in which case $\langle * b_3 \rangle$ is rooted at the $\neq$-branch of $\langle = b_2 \rangle$, or "performed second". Likewise if $\langle * b_3 \rangle$ is performed first, then $\langle = b_2 \rangle$ is performed second, rooted at the $\neq/<$-branch of $\langle * b_3 \rangle$. Figure 7 illustrates these two possible configurations. In most cases, $\langle = b_2 \rangle$ and $\langle * b_3 \rangle$ are performed in the same order as in the original tree T (i.e. $\langle * i \rangle$ comes before $\langle * j \rangle$, which comes before $\langle * k \rangle$), though one case (Case 2.2.2) requires going out-of-order, implying a rotation.

Because $\langle < b_2 \rangle$ is a new comparison, one of the sub-trees T_3, T_5, T_6, T_7 in T may fracture, meaning that some of its keys may satisfy this comparison and other may not. We will in fact show that only T_6 or T_7 can fracture, but their depths do not increase. So, as explained earlier, this fracturing will not increase cost. (As also explained before, the redundancies created by this fracturing, and other that can occur as a result of the conversion, can be eliminated by post-processing T' that iteratively splices out redundant branches.) We again break into further cases based on which key b_2 is.

Case 2.2.1: $b_2 = i$ and $\langle * j \rangle$ is an equal-to test. Then $T_3 = i$ and $T_5 = j$ are both leaves and can't fracture. Perform $\langle = b_2 \rangle$ first and $\langle * b_3 \rangle$ second (since $i = b_2$, this follows order of comparisons in the original tree). Then T_3 and T_5 both move down by 1, satisfying (j2), with $T_3 = i = b_2$ in the $=$-branch of $\langle = b_2 \rangle$ and $T_5 = j$ in the $=$-branch of either $\langle * b_1 \rangle$ or $\langle * b_3 \rangle$ (depending on whether j is b_1 or b_3).

Case 2.2.2: $b_2 = i$ and $\langle * j \rangle$ is a less-than test. Then $k < j$, so $j = b_3$ and $k = b_1$. By the case assumption, we have that $j > i$. Then in the $\geq$-branch of $\langle < b_2 \rangle$, we perform $\langle < j \rangle$ *before* $\langle = i \rangle$ (going out-of-order compared to the original tree), which will be in the $<$-branch of $\langle < j \rangle$. Then $T_3 = i$ will be below $\langle = i \rangle$ moving down twice and T_5 will be in the $\geq$-branch of $\langle < j \rangle$ staying at the same depth, satisfying (j1).

Case 2.2.3: $b_2 = j$. Then we may simply perform $\langle = b_2 \rangle$ and $\langle * b_3 \rangle$ in the same order as in the original tree. If $\langle * i \rangle$ is an equal-to test, then T_3 and T_5 are both leaves, and either both will go down by 1 (if $i = b_3$ and $k = b_1$), satisfying (j2), or only T_3 goes down by 2 (if $i = b_1$ and $k = b_3$), satisfying (j1). If $\langle * i \rangle$ is a less-than test,



(a) Original tree T

(b) Modified tree T', $\langle = b_2 \rangle$ goes first

(c) Modified tree T', $\langle * b_3 \rangle$ goes first

Fig. 7. Original and modified tree for Case 2.2. In this case, $T_5 = b_2$, and S_1, S_2, S_3, S_4 is some permutation of T_3, T_6, T_7, T'', where T'' is a copy of T_6 or T_7.

then $j, k < i$, so $k = b_1$ and $i = b_3$. $\langle < i \rangle$ will be performed first in the $\geq$-branch of $\langle < b_2 \rangle$ and T_3 will be in the $\geq$-branch of $\langle < i \rangle$, implying T_3 and T_5 both go down by 1, satisfying (j2).

Case 2.2.4: $b_2 = k$. The analysis is similar to Case 2.1.2. By the structure of T, $k \neq \{i, j\}$, so $b_1 < k$, implying $\langle * b_1 \rangle$ is an equal-to test. Then perform $\langle = b_2 \rangle$ and $\langle * b_3 \rangle$ in the same order as the original tree (that is, $\langle = k \rangle$ is always performed second). Thus $\langle * b_1 \rangle$ is in the $\geq$-branch of $\langle < r \rangle$ and $\langle * b_3 \rangle$ is in the $\geq$-branch of $\langle < b_2 \rangle$. If $(i, j) = (b_1, b_3)$, then T_3 (which is a leaf i) is at the $=/\geq$-branch of $\langle * b_1 \rangle$ and T_5 is at the $=/\geq$-branch of $\langle * b_3 \rangle$, implying T_5's depth stays the same and T_3 moves down by 2, satisfying (j1). If $(i, j) = (b_3, b_1)$, then leaf $T_5 = j$ is below $\langle * b_1 \rangle$ and T_3 is below $\langle * b_3 \rangle$, implying both trees move down by 1, satisfying (j2).

References

1. Anderson, R., Kannan, S., Karloff, H., Ladner, R.E.: Thresholds and optimal binary comparison search trees. J. Algorithms **44**, 338–358 (2002)
2. Atalig, S., Chrobak, M., Mousavian, E., Sgall, J., Vesely, P.: Structural properties of search trees with 2-way comparisons. CoRR, abs/2311.02224 (2023)

3. Bein, W., Golin, M.J., Larmore, L.L., Zhang, Y.: The Knuth-Yao quadrangle-inequality speedup is a consequence of total monotonicity. ACM Trans. Algorithms **6**(1), 1–17 (2009)
4. Chrobak, M., Golin, M., Munro, J.I., Young, N.E.: A simple algorithm for optimal search trees with two-way comparisons. ACM Trans. Algorithms **18**(1), 1–11 (2021)
5. Chrobak, M., Golin, M., Munro, J.I., Young, N.E.: On Huang and Wong's algorithm for generalized binary split trees. Acta Informatica **59**(6), 687–708 (2022)
6. Knuth, D.E.: Optimum binary search trees. Acta Informatica **1**, 14–25 (1971)
7. Knuth, D.E.: The Art of Computer Programming, Volume 3: Sorting and Searching, 2nd edn. Addison-Wesley Publishing Company, Redwood City (1998)
8. Spuler, D.: Optimal search trees using two-way key comparisons. Acta Informatica **31**(8), 729–740 (1994)
9. Spuler, D.A.: Optimal search trees using two-way key comparisons. PhD thesis, James Cook University (994)
10. Yao, F.F.: Efficient dynamic programming using quadrangle inequalities. In: Miller, R.E., Ginsburg, S., Burkhard, W.A., Lipton, R.J. (eds.) Proceedings of the 12th Annual ACM Symposium on Theory of Computing, 28–30 April 1980, Los Angeles, California, USA, pp. 429–435. ACM (1980)