

# Space-Efficient Finger Search on Degree-Balanced Search Trees\*

Guy E. Blelloch      Bruce M. Maggs      Shan Leung Maverick Woo

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213

{guyb, bmm, maverick}@cs.cmu.edu

## Abstract

We show how to support the finger search operation on degree-balanced search trees in a space-efficient manner that retains a worst-case time bound of  $O(\log d)$ , where  $d$  is the difference in rank between successive search targets. While most existing tree-based designs allocate linear extra storage in the nodes (e.g., for side links and parent pointers), our design maintains a compact auxiliary data structure called the “hand” during the lifetime of the tree and imposes *no* other storage requirement within the tree.

The hand requires  $O(\log n)$  space for an  $n$ -node tree and has a relatively simple structure. It can be updated synchronously during insertions and deletions with time proportional to the number of structural changes in the tree. The auxiliary nature of the hand also makes it possible to introduce finger searches into any existing implementation without modifying the underlying data representation (e.g., any implementation of Red-Black trees can be used). Together these factors make finger searches more appealing in practice.

Our design also yields a simple yet optimal in-order walk algorithm with *worst-case*  $O(1)$  work per increment (again without any extra storage requirement in the nodes), and we believe our algorithm can be used in database applications when the overall performance is very sensitive to retrieval latency.

## 1 Introduction

The problem of maintaining a sorted list of unique, totally-ordered elements is ubiquitous in computer science. When efficient element *access* (insert, delete, or search) is needed, one of the most common solutions is to use some form of balanced search trees to represent the list. Over the years many forms of balanced search trees have been devised, analyzed and implemented.

Balanced search trees are very versatile representa-

tions of sorted lists. In addition to providing element access in logarithmic time, certain forms also allow efficient aggregated operations like set intersection and union. For example, Brown and Tarjan [6] have shown a merging algorithm using AVL trees [1] with an optimal  $O(m \log \frac{n}{m})$  time bound, where  $m$  and  $n$  are the sizes of the two lists with  $m \leq n$ .

Their merging algorithm is, however, “not obvious and the time bound requires an involved proof” (p. 613, [7]). As such, in their subsequent paper, Brown and Tarjan [7] proposed a new structure by introducing extra pointers to a 2-3 tree [2] and called it a *level-linked 2-3 tree*. The merging algorithm on level-linked 2-3 trees is simple and intuitive and it uses the idea of finger searches, which we will define shortly. But there is a trade-off in this design. Each node in a level-linked 2-3 tree contains not only a key and two child pointers, but also a parent pointer and two side links. Considering this relatively high space requirement and the elegance of their simple yet optimal merging algorithm, it is natural to wonder if finger searches can be supported in a more *space-efficient* manner on any existing balanced search trees such as 2-3 trees. This is the motivation of our work.

**Finger search.** Consider a sorted list  $A$  of elements  $a_1, \dots, a_n$  represented by a search structure. Let the *rank* of an element be its position in the list and let  $\delta_A(a_i, a_j)$  be  $|i - j|$ , i.e., the difference in the ranks between  $a_i$  and  $a_j$  w.r.t. the ordering in  $A$ . We say that the search structure has the *finger search property* if searching for  $a_j$  takes  $O(\log \delta_A(a_i, a_j))$  time, where  $a_i$  is the most recently found element. The time bound can be worst-case or amortized and we will distinguish the two explicitly when needed. (As usual we let  $\log x$  denote  $\log_2 \max(2, x)$  and we will simply say  $O(\log d)$  when the elements  $a_i$  and  $a_j$  are not made explicit.)

A *finger* is a reference to an element in the list and historically it is often realized by a simple pointer to an element. (Indeed some papers mandate this representation in their definitions, e.g., see [5].) Typically, we maintain the invariant that the finger is on the most recently found element and we refer to this element as

---

\*This work was supported in part by the National Science Foundation under grants CCR-0205523 and CCR-9900304 and also through the Aladdin Center (www.aladdin.cs.cmu.edu) under grants CCR-0085982 and CCR-0122581. The second author is also affiliated with Akamai Technologies.

# Space-Efficient Finger Search on Degree-Balanced Search Trees\*

Guy E. Blelloch    Bruce M. Maggs    Shan Leung Maverick Woo  
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213  
{guyb,bmm,maverick}@cs.cmu.edu

## Abstract

We show how to support the finger search operation on degree-balanced search trees in a space-efficient manner that retains a worst-case time bound of  $O(\log d)$ , where  $d$  is the difference in rank between successive search targets. While most existing tree-based designs allocate linear extra storage in the nodes (e.g., for side links and parent pointers), our design maintains a compact auxiliary data structure called the “hand” during the lifetime of the tree and imposes *no* other storage requirement within the tree.

The hand requires  $O(\log n)$  space for an  $n$ -node tree and has a relatively simple structure. It can be updated synchronously during insertions and deletions with time proportional to the number of structural changes in the tree. The auxiliary nature of the hand also makes it possible to introduce finger searches into any existing implementation without modifying the underlying data representation (e.g., any implementation of Red-Black trees can be used). Together these factors make finger searches more appealing in practice.

Our design also yields a simple yet optimal in-order walk algorithm with *worst-case*  $O(1)$  work per increment (again without any extra storage requirement in the nodes), and we believe our algorithm can be used in database applications when the overall performance is very sensitive to retrieval latency.

## 1 Introduction

The problem of maintaining a sorted list of unique, totally-ordered elements is ubiquitous in computer science. When efficient element *access* (insert, delete, or search) is needed, one of the most common solutions is to use some form of balanced search trees to represent the list. Over the years many forms of balanced search trees have been devised, analyzed and implemented.

Balanced search trees are very versatile representa-

tions of sorted lists. In addition to providing element access in logarithmic time, certain forms also allow efficient aggregated operations like set intersection and union. For example, Brown and Tarjan [6] have shown a merging algorithm using AVL trees [1] with an optimal  $O(m \log \frac{n}{m})$  time bound, where  $m$  and  $n$  are the sizes of the two lists with  $m \leq n$ .

Their merging algorithm is, however, “not obvious and the time bound requires an involved proof” (p. 613, [7]). As such, in their subsequent paper, Brown and Tarjan [7] proposed a new structure by introducing extra pointers to a 2-3 tree [2] and called it a *level-linked 2-3 tree*. The merging algorithm on level-linked 2-3 trees is simple and intuitive and it uses the idea of finger searches, which we will define shortly. But there is a trade-off in this design. Each node in a level-linked 2-3 tree contains not only a key and two child pointers, but also a parent pointer and two side links. Considering this relatively high space requirement and the elegance of their simple yet optimal merging algorithm, it is natural to wonder if finger searches can be supported in a more *space-efficient* manner on any existing balanced search trees such as 2-3 trees. This is the motivation of our work.

**Finger search.** Consider a sorted list  $A$  of elements  $a_1, \dots, a_n$  represented by a search structure. Let the *rank* of an element be its position in the list and let  $\delta_A(a_i, a_j)$  be  $|i - j|$ , i.e., the difference in the ranks between  $a_i$  and  $a_j$  w.r.t. the ordering in  $A$ . We say that the search structure has the *finger search property* if searching for  $a_j$  takes  $O(\log \delta_A(a_i, a_j))$  time, where  $a_i$  is the most recently found element. The time bound can be worst-case or amortized and we will distinguish the two explicitly when needed. (As usual we let  $x$  denote  $\log_2 \max(2, x)$  and we will simply say  $O(\log d)$  when the elements  $a_i$  and  $a_j$  are not made explicit.)

A *finger* is a reference to an element in the list and historically it is often realized by a simple pointer to an element. (Indeed some papers mandate this representation in their definitions, e.g., see [5].) Typically, we maintain the invariant that the finger is on the most recently found element and we refer to this element as

\*This work was supported in part by the National Science Foundation under grants CCR-0205523 and CCR-9900304 and also through the Aladdin Center (www.aladdin.cs.cmu.edu) under grants CCR-0085982 and CCR-0122581. The second author is also affiliated with Akamai Technologies.

# Space-Efficient Finger Search on

Guy E. Blelloch      Bruce M. Magdon  
Computer Science Department, Carnegie Mellon University  
{guyb, bmm, maver}

## Abstract

We show how to support the finger search operation on degree-balanced search trees in a space-efficient manner that retains a worst-case time bound of  $O(\log d)$ , where  $d$  is the difference in rank between successive search targets. While most existing tree-based designs allocate linear extra storage in the nodes (e.g., for side links and parent pointers), our design maintains a compact auxiliary data structure called the “hand” during the lifetime of the tree and imposes *no* other storage requirement within the tree.

The hand requires  $O(\log n)$  space for an  $n$ -node tree and has a relatively simple structure. It can be updated synchronously during insertions and deletions with time proportional to the number of structural changes in the tree. The auxiliary nature of the hand also makes it possible to introduce finger searches into any existing implementation without modifying the underlying data representation (e.g., any implementation of Red-Black trees can be used). Together these factors make finger searches more appealing in practice.

Our design also yields a simple yet optimal in-order walk algorithm with *worst-case*  $O(1)$  work per increment (again without any extra storage requirement in the nodes), and we believe our algorithm can be used in database applications when the overall performance is very sensitive to retrieval latency.

## 1 Introduction

The problem of maintaining a sorted list of unique, totally-ordered elements is ubiquitous in computer science. When efficient element *access* (insert, delete, or search) is needed, one of the most common solutions is to use some form of balanced search trees to represent the list. Over the years many forms of balanced search trees have been devised, analyzed and implemented.

Balanced search trees are very versatile representa-

---

\*This work was supported in part by the National Science Foundation under grants CCR-0205523 and CCR-9900304 and also through the Aladdin Center ([www.aladdin.cs.cmu.edu](http://www.aladdin.cs.cmu.edu)) under grants CCR-0085982 and CCR-0122581. The second author is also affiliated with Akamai Technologies.

# 1 Degree-Balanced Search Trees\*

Shan Leung Maverick Woo  
Mellon University, Pittsburgh, PA 15213  
mwo@cs.cmu.edu

tions of sorted lists. In addition to providing element access in logarithmic time, certain forms also allow efficient aggregated operations like set intersection and union. For example, Brown and Tarjan [6] have shown a merging algorithm using AVL trees [1] with an optimal  $O(m \log \frac{n}{m})$  time bound, where  $m$  and  $n$  are the sizes of the two lists with  $m \leq n$ .

Their merging algorithm is, however, “not obvious and the time bound requires an involved proof” (p. 613, [7]). As such, in their subsequent paper, Brown and Tarjan [7] proposed a new structure by introducing extra pointers to a 2-3 tree [2] and called it a *level-linked 2-3 tree*. The merging algorithm on level-linked 2-3 trees is simple and intuitive and it uses the idea of finger searches, which we will define shortly. But there is a trade-off in this design. Each node in a level-linked 2-3 tree contains not only a key and two child pointers, but also a parent pointer and two side links. Considering this relatively high space requirement and the elegance of their simple yet optimal merging algorithm, it is natural to wonder if finger searches can be supported in a more *space-efficient* manner on any existing balanced search trees such as 2-3 trees. This is the motivation of our work.

**Finger search.** Consider a sorted list  $A$  of elements  $a_1, \dots, a_n$  represented by a search structure. Let the *rank* of an element be its position in the list and let  $\delta_A(a_i, a_j)$  be  $|i - j|$ , i.e., the difference in the ranks between  $a_i$  and  $a_j$  w.r.t. the ordering in  $A$ . We say that the search structure has the *finger search property* if searching for  $a_j$  takes  $O(\log \delta_A(a_i, a_j))$  time, where  $a_i$  is the most recently found element. The time bound can be worst-case or amortized and we will distinguish the two explicitly when needed. (As usual we let  $\log x$  denote  $\log_2 \max(2, x)$  and we will simply say  $O(\log d)$  when the elements  $a_i$  and  $a_j$  are not made explicit.)

A *finger* is a reference to an element in the list and historically it is often realized by a simple pointer to an element. (Indeed some papers mandate this representation in their definitions, e.g., see [5].) Typically, we maintain the invariant that the finger is on the most recently found element and we refer to this element as