Project Report

Implementation of Finger Search:
An Extended Feature of Some Data Structures

Minh Thang Cao

24 February 2021

1 Introduction

Finger Search is an extended feature, which is suitable for some popular data structures, that improves the running time of search operations as well as other ones that depend on searching. Finger Search uses a pointer to an element, called *finger*, in the data structure to reduce the length of search path. Instead of searching from a normal start point, Finger Search starts at the *finger*. Let d be the difference between ranks[1] of the *finger* and the target value, the running time of Finger Search is proportional to d.

2 Finger Search on Treaps

Briefly, Treap is a Randomized Binary Search Tree which is a combination of Binary Search Tree and Heap. Each nodes in Treap holds a randomized priority which is used to maintain the Heap properties. The nodes order in a Treap satisfies ascending values in an inorder traversal while their priority satisfies min-heap (or max-heap) property, in which priority of a node is always smaller (or larger) than its children. This randomized combination creates an property of Treap that the *expected* height of a Treap is $O(logn)$ (not always), with n is the total number of nodes. Therefore, it supports searching and some associated operations in expected $O(logn)$ time which is standard for most balanced trees. Since Finger Search produces an expected running time, Treap is a very suitable randomized data structure for Finger Search.

In this section, Finger Search on Treaps will be introduced along with my implementation. Due to R. Seidel and C.R. Aragon (see [1]), there are several ways to achieve the running time proportional to the distance d such as using multiple pointers, multiple keys, or even without anything by relying on the shortness of the search path, and these approaches all give us the $O(logd)$ expected running time. Since if we do Finger Search without any assistance of other elements, the excess path in some cases may be much longer compared to the path that

[1]Denoted i if x is the i^{th} smallest element in the data structure (consistent with only smallest or largest for all elements)

we are supposed to search on. To minimize the probability that we get these cases, multiple extra pointers are definitely useful, which is used in my implementation.

Based on what R. Seidel and C.R. Aragon found (see Section 5.4 of [1]), denote a *left parent* or a *right parent* of a node u is the first ancestor of u on the path from u to the *root* node that satisfies u is on the ancestor's right or left subtree, respectively (see Figure 1).

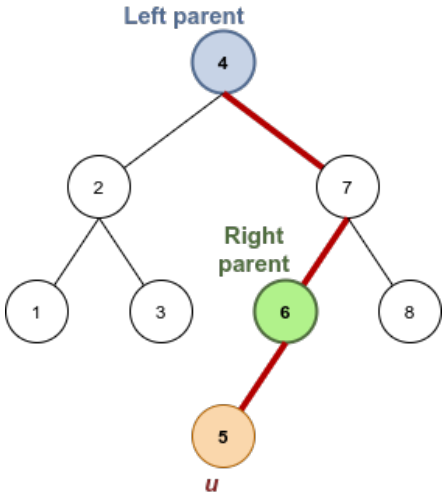

Figure 1: Visualization of *left parent* and *right parent* of a node u in a Binary Tree.

References

[1] Seidel, R., Aragon, C.R. Randomized search trees. *Algorithmica* 16, 464–497 (1996). https://doi.org/10.1007/BF01940876