# Relaxed Radix Balanced Trees

Elias Khoury

2016

# Background

- Invented by researchers at EPFL in Lausanne
- RRB-Trees are part of the ongoing development of the Scala Programming Language
- The goal is to present a datastructure which can be concatenated in O(log n) time at no cost to other operations

# Application

- RRB-Trees can be used to implement an efficient vector datastructure
- As it is a purely functional datastructure the operations on it can be easily run in parallel
- An application for efficient vector concatenation is:

# Main Concept

- To represent a vector as a tree structure
- Multiple variables to control:
  - Branching factor
  - Balance of the tree
- The main operations are:
  - Indexing
  - Updating
  - Append to the front or back
  - Splitting

# The Problem

Given two trees of branching factor m, it is impossible to merge the trees faster than linear time without degenerating to a linked list

# Radix Search

- With a perfectly balanced tree of branching factor m, for any particular sub branch, there are exactly $m\hat{h}$-1 leaves. Therefore, an index i can be found using $Li/(m\hat{h}-1)L$

# Relaxed Radix Search

- However, for this particular application we need to relax the branching factor m in order to circumvent the linear concatenation cost.
- The concept of a variable branching factor has been applied to tree structures before, like 2-3 Trees and finger trees
- By relaxing the branching factor, we lose the guarantee that any leaf node can be found in O(1) time, this is due to the possibility of the tree being slightly unbalanced.
- The cost this adds is a slight linear lookup when arriving at a node that is unbalanced.

# Branching Factor and Balance

- BALANCE

# RRB-Trees

- The concatenation operation works by effectively merging the right side of the left tree, with the left side of the right tree.
- starting from the leaf nodes of both trees, a new tree is built from the bottom up containing the merged data of both trees.
- The concept behind this is to use the concept of sharing when building this new tree in order to minimise the cost of copying data
- For a tree of height, h. The complexity of this operation is O(Log(h)). However, there is a significant constant time cost when it comes to ensuring that the merged tree is balanced

# Concatenation Process

# A note on branching factor

- For simplicity of the diagrams a branching factor of 4 was chosen
- In practice the most efficient branching factor is actually 32
- Branching factor controls tree height which affects the runtime of the operations
- Need to minimise branching factor without degenerating into a linked list

| | RRB-Vector | With m = 32 |
|---|---|---|
| indexing | $log_m$ | eC |
| update | aC | eC |
| insert ends | $m \times log_m$ | aC |
| concat | $m^2 \times log_m$ | L v.s. eC |
| split | $m \times log_m$ | eC |

# Runtime Complexity In Relation To Similar Structures