Relaxed Radix Balanced Trees

Elias Khoury

2016

Background

- Invented by researchers at EPFL in Lausanne

Background

- Invented by researchers at EPFL in Lausanne
- RRB-Trees are part of the ongoing development of the Scala Programming Language

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

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

Main Concept

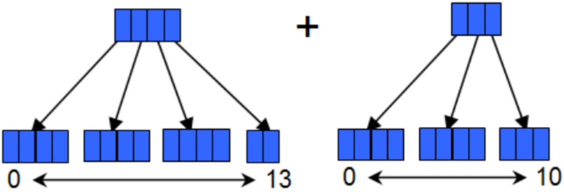
- To represent a vector as a tree structure
- Multiple variables to control:
  - Branching factor
  - Balance of the tree

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

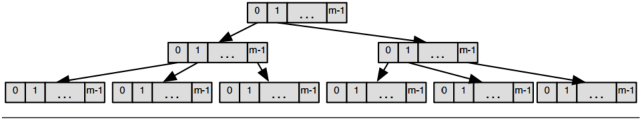
The Problem

Given two trees of branching factor m, a naive approach will simply be a copy of linear cost from one tree to another. Likewise a naive O(1) solution is possible, but at the cost of degenerating the datastructure into a linked list



Radix Search

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



eg. For m = 4. Finding the index 5 using the tree above yields the following calculations:

$$\lfloor 5/(4^{3-1}) \rfloor = 0 \quad \lfloor 5/(4^{1-1}) \rfloor = 5$$
$$\lfloor 5/(4^{2-1}) \rfloor = 1 \quad 5 \ \% \ 4 = 1$$

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

- Having a variable branching factor makes it more difficult to maintain a balanced structure.

Branching Factor and Balance

- Having a variable branching factor makes it more difficult to maintain a balanced structure.
- A good way of still being able to maintain the convenience of a balanced tree is to simply state the size of each unbalanced subtree.

Branching Factor and Balance

- Having a variable branching factor makes it more difficult to maintain a balanced structure.
- A good way of still being able to maintain the convenience of a balanced tree is to simply state the size of each unbalanced subtree.
- Furthermore, choosing the upper and lower bounds of the branching factor can help determine how unbalanced the tree could become.

Branching Factor and Balance

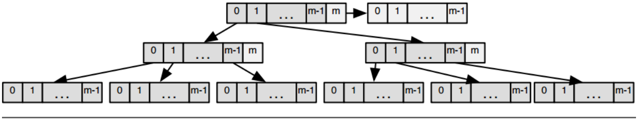
- Having a variable branching factor makes it more difficult to maintain a balanced structure.
- A good way of still being able to maintain the convenience of a balanced tree is to simply state the size of each unbalanced subtree.
- Furthermore, choosing the upper and lower bounds of the branching factor can help determine how unbalanced the tree could become.
- For a tree of branching factor m, the height h is $log_m(n)$. Therefore, given two possible values of m. the ratio of the heights is $\frac{log_{m_{min}}}{log_{m_{max}}}$

Branching Factor and Balance

- Having a variable branching factor makes it more difficult to maintain a balanced structure.
- A good way of still being able to maintain the convenience of a balanced tree is to simply state the size of each unbalanced subtree.
- Furthermore, choosing the upper and lower bounds of the branching factor can help determine how unbalanced the tree could become.
- For a tree of branching factor m, the height h is $log_m(n)$. Therefore, given two possible values of m. the ratio of the heights is $\frac{log_{m_{min}}}{log_{m_{max}}}$
- The closer to 1 this ratio is the more balanced the tree is. Choosing $m_{max} = m_{min} + 1$ is a good compromise.

Branching Factor and Balance

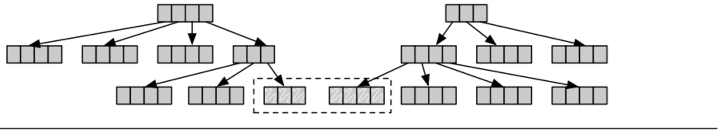
- Having a variable branching factor makes it more difficult to maintain a balanced structure.
- A good way of still being able to maintain the convenience of a balanced tree is to simply state the size of each unbalanced subtree.
- Furthermore, choosing the upper and lower bounds of the branching factor can help determine how unbalanced the tree could become.
- For a tree of branching factor m, the height h is $log_m(n)$. Therefore, given two possible values of m. the ratio of the heights is $\frac{log_{m_{min}}}{log_{m_{max}}}$
- The closer to 1 this ratio is the more balanced the tree is. Choosing $m_{max} = m_{min} + 1$ is a good compromise.



RRB-Trees

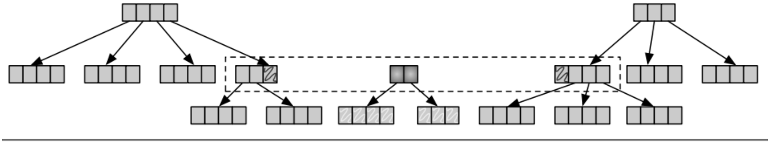
- The concatenation operation works by effectively merging the right side of the left tree, with the left side of the right tree.
- starting from the leaf nodes of both trees, a new tree is built from the bottom up containing the merged data of both trees.
- The concept behind this is to use the concept 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



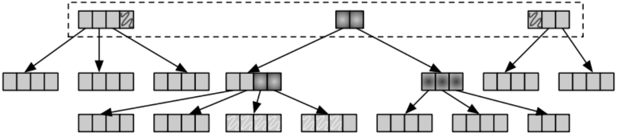
merge the leaf nodes of each tree to create a new tree of height 1

Concatenation Process

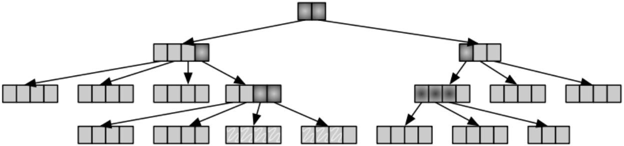


propagate the merge up the tree. Note: At each stage of the merge the tree needs to be rebalanced

Concatenation Process



Only the highlighted nodes need to be modified. The rest are simply shared.



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

	RRB-Vector	Finger Tree	Red-Black Tree
indexing	eC	Log_2	Log_2
update	eC	Log_2	Log_2
insert ends	aC	aC	Log_2
concat	L v.s. eC	Log_2	L
split	eC	Log_2	Log_2