Relaxed Radix Balanced Trees

Elias Khoury

2016

Background

- Invented by researchers at EPFL in Lausanne[Bagwell and Rompf, 2011]

Background

- Invented by researchers at EPFL in Lausanne[Bagwell and Rompf, 2011]
- RRB-Trees are part of the ongoing development of the Scala Programming Language

Background

- Invented by researchers at EPFL in Lausanne[Bagwell and Rompf, 2011]
- RRB-Trees are part of the ongoing development of the Scala Programming Language
- The goal is to present a tree 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[Stucki et al., 2015]
- An application for efficient vector concatenation can be used to generate SQL queries, or even to dynamically generate XML code or HTML code for websites. [Hull]

Main Concept

- To represent a vector as a tree structure (an alternative example being a cons list)

Main Concept

- To represent a vector as a tree structure (an alternative example being a cons list)
- Multiple variables to control:
  - Branching factor
  - Balance of the tree

Main Concept

- To represent a vector as a tree structure (an alternative example being a cons list)
- 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

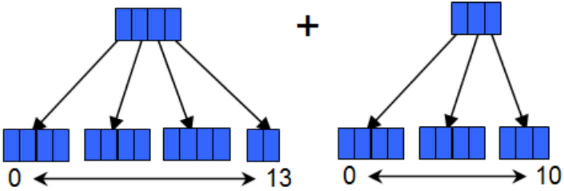
- Given two trees of branching factor m, a naive approach will simply be a copy of linear cost from one tree to another.

The Problem

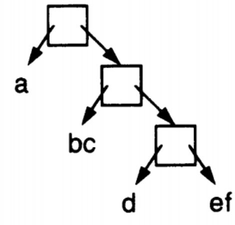
- Given two trees of branching factor m, a naive approach will simply be a copy of linear cost from one tree to another.
- Likewise a naive O(1) solution is possible by creating a new root and simply attaching it to the two trees. However, this is at the cost of degenerating the datastructure into a linked list[Boehm et al., 1995]

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 by creating a new root and simply attaching it to the two trees. However, this is at the cost of degenerating the datastructure into a linked list[Boehm et al., 1995]



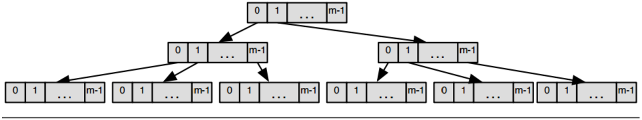
Concatenating Two Vectors



Examples of Ropes

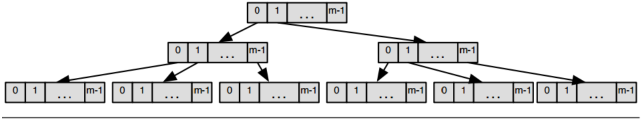
Radix Search

- 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 $\lfloor i/(m^{h-1}) \rfloor$ [Angle et al., 1999]



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 $\lfloor i/(m^{h-1}) \rfloor$ [Angle et al., 1999]



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.

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[Hinze and Paterson, 2006].

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[Hinze and Paterson, 2006].
- 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.

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[Hinze and Paterson, 2006].
- 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

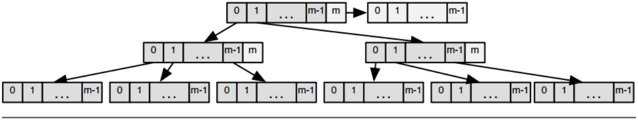
- For a tree of branching factor m, the height h is $log_m(n)$. Therefore, given two possible values of m. the ratio of the heights is $\frac{log_{m_{min}}}{log_{m_{max}}}$

Branching Factor and Balance

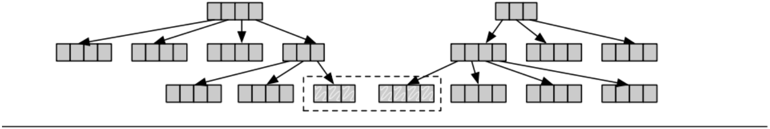
- For a tree of branching factor m, the height h is $log_m(n)$. Therefore, given two possible values of m. the ratio of the heights is $\frac{log_{m_{min}}}{log_{m_{max}}}$
- The closer to 1 this ratio is the more balanced the tree is. Choosing $m_{max} = m_{min} + 1$ is a good compromise.

Branching Factor and Balance

- 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.

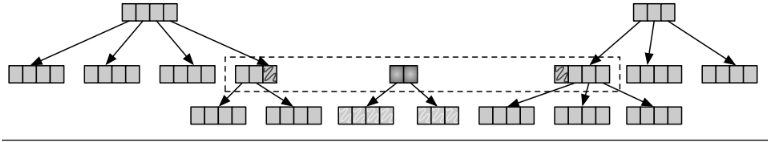


Concatenation Process



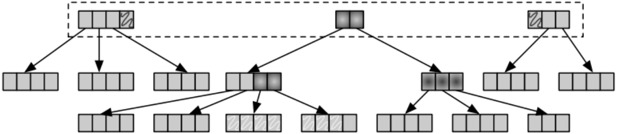
merge the leaf nodes of each tree to create a new tree of height 1

Concatenation Process

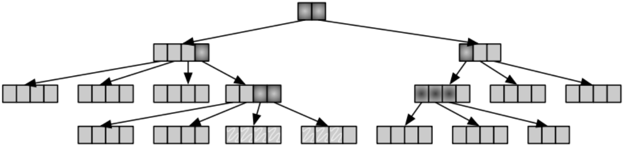


propagate the merge up the tree. Note: At each stage of the merge the tree needs to be rebalanced

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

References

All diagrams are sourced from Bagwell and Rompf [2011] and Stucki et al. [2015]

R. L. Angle, E. S. Harriman Jr, and G. B. Ladwig. Radix tree search logic, Feb. 16 1999. US Patent 5,873,078.

P. Bagwell and T. Rompf. Rrb-trees: Efficient immutable vectors. Technical report, 2011.

H.-J. Boehm, R. Atkinson, and M. Plass. Ropes: an alternative to strings. *Software: Practice and Experience*, 25(12):1315–1330, 1995.

R. Hinze and R. Paterson. Finger trees: a simple general-purpose data structure. *Journal of Functional Programming*, 16(02):197–217, 2006.

M. Hull. Balancing simplicity and efficiency in web applications.

N. Stucki, T. Rompf, V. Ureche, and P. Bagwell. Rrb vector: a practical general purpose immutable sequence. In *ACM SIGPLAN Notices*, volume 50, pages 342–354. ACM, 2015.