MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

Jakob Truelsen
Aarhus University

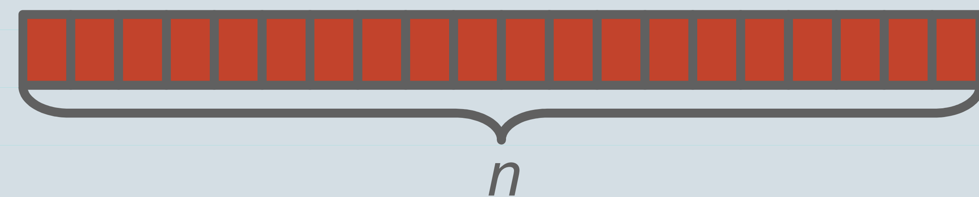
Casper Kejlberg-Rasmussen
Aarhus University

Danmarks Grundforskningsfond
Danish National Research Foundation

A Cache-Oblivious Implicit Dictionary with the Working Set Property

The Implicit Model

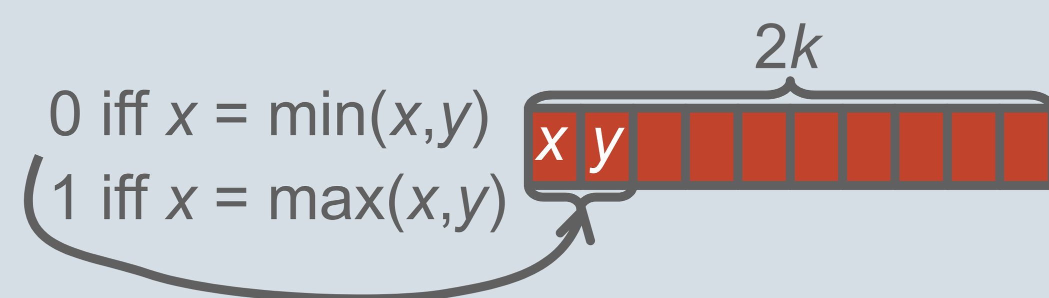
The implicit model is the RAM model but with some restrictions: you are only allowed to store exactly the n elements that you are to build the data structure over, nothing more. That means that the n elements are laid out in an array of n elements. So we cannot e.g. leave some of the entries empty, as they would still occupy space.



While performing operations you are allowed to store $O(1)$ words of memory, i.e. $O(\log n)$ bits of memory. When an operation finishes we forget the $O(1)$ words of memory we could use during the operation.

The Fundamental Trick

The fundamental trick to do anything in this models is to encode bits using permutations of elements. That is say we want to encode k bits, then we can encode them into $2k$ elements where we for the first two elements x and y encode a 0 iff $x = \min(x,y)$ and a 1 iff $x = \max(x,y)$. We encode the remaining k-1 bits the same way using the other k-1 pairs of elements.



The Working Set Property

The working set distance l of a key x is the number of keys we have accessed since we last searched for x. A dictionary has the working set property if the search time for a key x is expressesed in terms of l, in our result the search cost is $O(\log l)$.

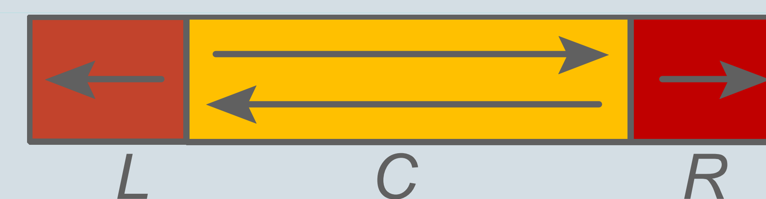
Previous Work

Our result is the first implicit dictionary with the working set property, and it is in fact also cache-oblivious, which is a property we inherent from the dictionary of [FG 2003] which we use as a black box.

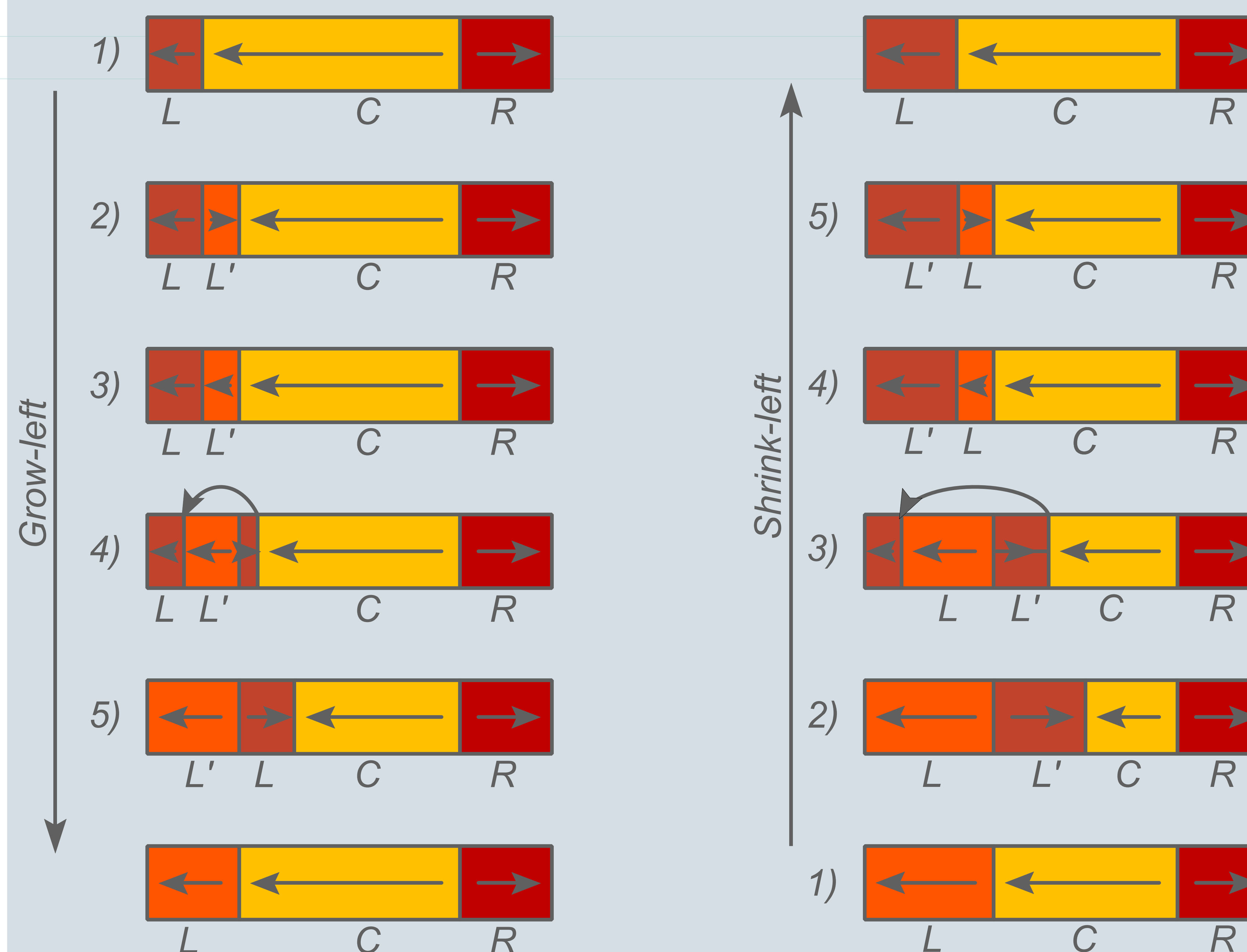
Reference	Insert / Delete	Search	Predecessor / Successor	Additional space (words)
[I 2001]	$O(\log n)$	$O(\log l)$	$O(\log l)$	$O(n)$
[FG 2003]	$O(\log n)$	$O(\log n)$	$O(\log n)$	None
[BHM 2009]	$O(\log n)$	$O(\log l)$ *exp.*	$O(\log n)$	$O(\log\log n)$
[BHM 2009]	$O(\log n)$	$O(\log l)$ *exp.*	$O(\log l)$ *exp.*	$O(\sqrt{n})$
[BKT 2010]	$O(\log n)$	$O(\log l)$	$O(\log n)$	None

A Moveable Dictionary

To make the working set dictionary we will need a moveable version of the dictionary (a FG dict) from [FG 2003], i.e. insertions and deletions can happen at both ends of the array. We obtain it using at most 4 instances of the FG dict. In its basic form the moveable dictionary consists of FG dicts L, C and R, growing to the left, left/right and right, respectively. C can be growing in either way depending which time we look at it.



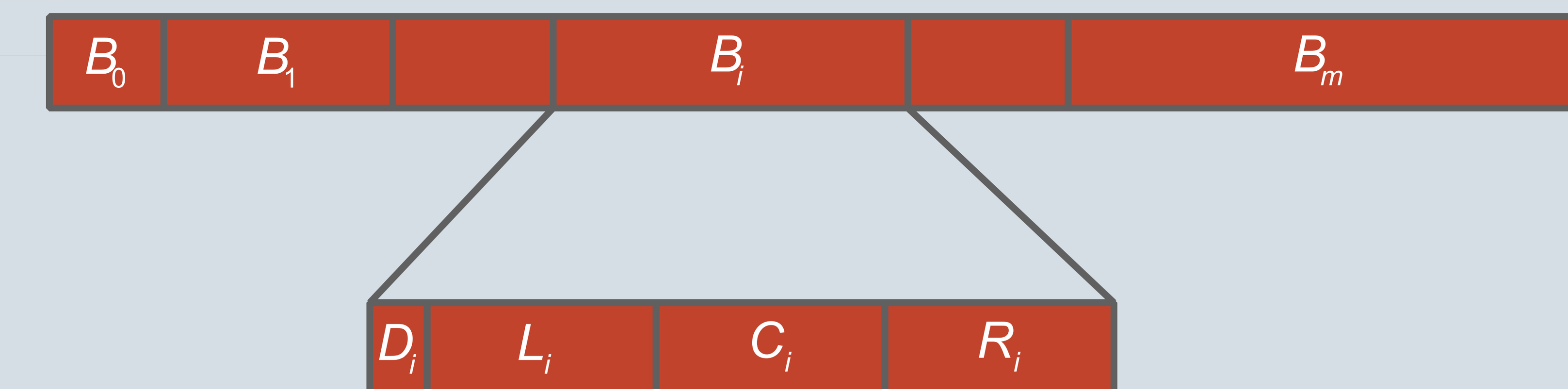
It is easy to move this dictionary one element to the left by taking an element from R and putting it into L, likewise it is simple to move it right. The trouble comes if L or R becomes empty or too large compared to C, hence we trigger a job to either grow or shrink L or R.



A job simply performs its task by executing a constant number of steps during each insert, delete, search or predecessor/successor operation. To grow L when it becomes to small we use the simple trick of address-mapping to split one of the FG dicts into two parts – see L in grow-left and L' in shrink-left – that way we can take elements from C and put into L' while we shrink L further, and when L' grows enough it takes over the role of L. We do simlar for shrink-left, see the right of the above figure.

A Working Set Dictionary

The working set dictionary is now constructed from moveable dictionaries of double exponentially increasing size. The working set dictionary is divided into blocks where the i'th block is of size $\Theta(2^{2^i})$. The i'th block further consists of a set of elements D_i, and three moveable dictionaries L_i, C_i and R_i. Here elements in R_i are ready to move to D_{i+1} or L_{i+1} in block B_{i+1} and elements in C_i are waiting to be promoted to be in R_i.



It is now easy to make a search in $O(\log l)$ time: for an element, we first search in B_0 then B_1 and so until we find the element we are searching for or nothing. In each block B_i we search in D_i by a scan and in L_i, C_i and R_i using the search function of the moveable dictionary. We then remove the found element e of block B_j, where $j = \log l$, and insert it into block B_0 and move a element from R_i to L_{i+1} for $i = 0,\ldots,j$-1. If R_i runs out of elements we rename C_i to R_i, L_i to C_i and make a new empty L_i. This way we spend time

$$\sum_{i=0}^{\log l} \log\left(2^{2^i}\right) = \sum_{i=0}^{\log l} 2^i = O(\log l)$$

Future / Current Work

We are currently working on doing the predecessor and successor search in the working set time also, that is we get search and predecessor/successor in $O(\log l)$ time and insert and delete in $O(\log n)$ time, using no space. This will still be cache-oblivious because we again use the moveable dictionary that again uses the FG dict.

References

[I 2001] Iacono - *Alternatives to splay trees with O(log(n)) worst-case access times.* SODA 2001.

[FG 2003] Franceschini, Grossi - *Optimal worst-case operations for implicit cache-oblivious search trees.* WADS 2003.

[BHM 2009] Bose, Howat, Morin - *A distribution-sensitive dictionary with low space overhead.* WADS 2009.

[BKT 2010] Brodal, Kejlberg-Rasmussen, Truelsen - *A Cache-Oblivious Implicit Dictionary with the Working Set Property.* ISAAC 2010.