A correction to Andersson's fusion tree construction

Željko Vrba, Pål Halvorsen, Carsten Griwodz

We revisit the fusion tree [1] data structure as presented by Andersson et al. [2]. We show that Andersson's ranking algorithm has a subtle flaw and we present a revised algorithm.

1 Introduction

Fusion tree, designed by Fredman and Willard [1], is a variant of B-tree which stores n elements in $O(n)$ space and uses $O(\frac{\log n}{\log\log n})$ (amortized) time for search, insert and delete operations. Their algorithm uses multiplication to manipulate individual bits of a computer word, but a multiplication circuit is not in the AC^0 complexity class. At the end of their paper, they asked whether fusion trees can be implemented without multiplication, using only operations in the AC^0 class.

Their question was answered positively by Andersson et al. in [2], where they explicitly described an alternate way of constructing and querying nodes. Thorup [3] has further discussed how Andersson's construction can be implemented on real-world processors (concretely, using SIMD features of Pentium 4).

In this paper, we describe and correct a subtle flaw in Andersson's construction, which we have discovered by attempting to implement it using Thorup's suggestions. For completeness, we will first shortly describe Andersson's variant of fusion trees. Then, we demonstrate that the algorithm has a flaw by giving an explicit example for which it returns a wrong result. Finally, we explain the underlying cause of the flaw and describe how it can be corrected.

2 Fusion tree

A fusion tree may be understood as a B-tree containing w-bit machine *words* as keys such that: 1) internal node degree d does not exceed $\sqrt{w}$, which we assume to be a power of two; 2) the tree leaves are roots of balanced binary trees of size $\Theta(d)$. If we can find the rank of value X within an internal node consisting of $n \le d$ *keys* $\Upsilon = \{Y^0 < Y^1 < \ldots < Y^{n-1}\}$ in constant time, we will achieve sublogarithmic searches.[1] The second condition guarantees sublogarithmic amortized cost of inserts and deletes.

Obviously, the challenging part is constant-time rank computation, which Fredman and Willard have achieved by inventive use of multiplication. Their algorithm, though, has constant factors that they themselves admit are impractically high: node construction takes $O(n^4)$ time; a word and node can hold up to $\lfloor w^{1/6} \rfloor$ elements; each compressed key requires up to n^4 bits; a lookup table using $O(n^2)$ space is stored in the node along with the keys.

Andersson et al. have later described fusion trees in a more accessible manner [2], simultaneously improving it in two ways: 1) the bound on node degree n is increased to $\sqrt{w}$, and 2) their rank computation (which they got subtly wrong) does not need a lookup table. Except for the change of indexing conventions, the presentation of algorithms in this section closely follows Andersson's [2].

[1]The superscripts here are just indexes.

2.1 Definitions and conventions

Indexing. Indexing is 0-based, and if there exists an implied order, the 0th element is the least element.

Words, fields and bits. Given a *word* X, $X[i]$ $(0 \le i < w)$ denotes the i'th bit, 0 being the least significant bit. We partition words into *fields* of length $b = \sqrt{w}$ bits, so there are b fields in a word, with bits $X[0 \ldots b-1]$ comprising field 0. By X_i $(0 \le i < b)$, we denote the i'th field of X. Values of words and fields are interpreted as *unsigned* binary numbers.

Words as sorted sets. Fields of X may be naturally viewed as a set. In our case, fields are sorted in increasing order from right to left, i.e., $X_0 \le X_1 \le X_{k-1}$. Some fields may be unused $(k < b)$, and we are not concerned with their contents. The number of used fields is stored externally to the word.

Operators. $u \oplus v$, $u \& v$ and $u \ll v$ denote respectively bitwise XOR, AND, and left shift (v is shift amount in bits). $|\Upsilon|$ denotes cardinality of Υ. $\nu(X)$ denotes the index of the most significant set bit (MSB) in a word; we define $\nu(0) = -1$. For example, $\nu(13) = \nu(1101_2) = 3$.

Rank. Given Υ and X, the *rank* of X in Υ $(\rho_\Upsilon(X))$ is defined as the number of elements in Υ that are less than or equal to X. For example, let $\Upsilon = \{3, 7, 8, 10\}$. Then, $\rho_\Upsilon(1) = 0$, $\rho_\Upsilon(3) = \rho_\Upsilon(4) = 1$, $\rho_\Upsilon(10) = \rho_\Upsilon(11) = 4$. We define analogously $\rho_Y(x)$ for a sorted set represented by the fields of word Y and value x that fits into a field.

Select. Given a list of (not necessarily distinct) bit indices K, $\Sigma_K(X)$ denotes a binary number formed by selecting bits from X that are in K and "compacting" them. For example, $\Sigma_{1,2,5}(37_{10}) = \Sigma_{1,2,5}(\underline{1}00\underline{10}1_2) = 110_2 = 6$.

2.2 Node construction

Fusion tree node may be understood as a compressed representation of binary trie containing the set of keys Υ. A node contains the following data:

- The set of *significant bit positions* packed into a word K where $K_i = \nu(Y^i \oplus Y^{i+1})$ for $0 \le i < n-1$. In other words, K contains indices of the most significant bit in which two *adjacent* full-length keys differ.
- The set of *compressed keys* packed into a word Y where $Y_i = \Sigma_K(Y^i)$. In other words, Y_i is obtained from Y^i by extracting bits at positions in K obtained in previous step.
- Additionally, we store the set Υ containing full-length keys and $n = |\Upsilon|$.

This transformation has two important properties: 1) elements of K can be represented by $\lg w$ bits, and w-bit elements of Υ are compressed to $n-1$ bits; 2) the $\Sigma_K(X)$ mapping preserves ordering: $Y^i < Y^j \Rightarrow \Sigma_K(Y^i) < \Sigma_K(Y^j)$.

Example. Assume that $w = 64$, and let the set of keys be[2] $\Upsilon = \{0D_{16}, 11_{16}, 18_{16}, 1C_{16}, F0_{16}\}$, or, in binary, $\Upsilon = \{00001101_2, 00010001_2, 00011000_2, 00011100_2, 11110000_2\}$.

[2]Without loss of generality, we use 8-bit keys for simplicity of presentation; bits $8 \ldots 63$ are 0.

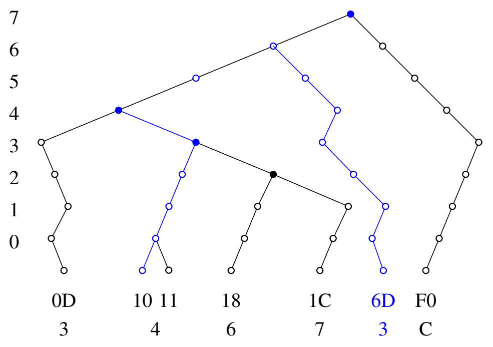

Figure 1: Binary trie over 5 keys (in black). Levels containing branching nodes (7, 4, 3, 2) are also positions of significant bits. The paths in blue (corresponding to 10_{16} and $6D_{16}$) are not part of the node; they just show how example queries of section 2.3 fit into the trie.

Figure 1 shows elements of Υ arranged in a binary trie. We first find that significant bits are at indices 4, 3, 2, 7 (in that order), which correspond exactly to the trie levels containing branching nodes. Now, we can extract significant bits from the keys to obtain the set of compressed keys $\{3_{16}, 4_{16}, 6_{16}, 7_{16}, \mathrm{C}_{16}\}$, or, in binary, $\{0011_2, 0100_2, 0110_2, 0111_2, 1100_2\}$. We have thus computed $K = 07040302_{16}$ and $Y = 0\mathrm{C}07060403_{16}$ (the values are zero-padded so that each occupies a whole field of 8 bits.)

2.3 Rank computation

Andersson gave the following procedure for computing $\rho_K(X)$:

1. Set $x \leftarrow \Sigma_K(X)$ and $h \leftarrow \rho_Y(x)$.
2. $p \leftarrow \min(\nu(X \oplus Y^{h-1}), \nu(X \oplus Y^h))$
3. Set $Z \leftarrow X$ and $Z[p \ldots 0] \leftarrow X[p]$.
4. Set $z \leftarrow \Sigma_K(Z)$ and $h \leftarrow \rho_Y(z)$.
5. If $Y^{h-1} \leq Z < Y^h$, return h; otherwise return $\rho_Y(z \oplus 1)$.

Step 2 determines the key sharing the longest common prefix with X, which is either Y^{h-1} or Y^h, and sets p to the index of the most significant differing bit. If $h = 0$ or $h = n$, the appropriate values in steps 2 and 5 are ignored.

Example. In table 1, we have worked out the steps of Andersson's algorithm for two different searches. Note that wrong result is returned for the first search; the correct result is 1.

The problem in Andersson's procedure lies in the last step, which is based on the proof of Lemma 5 in [2]. The proof overlooks the possibility that the longest prefix of X and one of the keys in Υ may extend past the lowest bit in K, i.e., that $p < \min K$, resulting in $z = y_{h-1}$ after steps 3 and 4. In this case, X may fall on either side of Y^{h-1} (refer

Step	$X = 10_{16}$	$X = 6\text{D}_{16}$
1	$x = 4$, $h = 2$	$x = 3$,$h = 1$
2	$p = 0$	$p = 6$
3	$Z = 10$	$Z = 7F$
4	$z = 4$, $h = 2$	$z = 7$, $h = 4$
5	$\rho_Y(z \oplus 1) = 2$	$h = 4$

Table 1: Values of key variables during execution of Andersson's ranking algorithm [2]. Note that the computed rank for 10 is *wrong*; the correct rank is 1. The condition of step 5 is false for the first query, so another packed rank is computed. The same condition is true for the second query.

again to figure 1), and the correct side may only be determined by comparing full-length keys. Step 5 should thus be modified to

$5'$ If $Y_{h-1} = z$ and $X < Y^{h-1}$, set $h \leftarrow h - 1$. Return h.

References

[1] M. L. Fredman, D. E. Willard, Surpassing the information theoretic bound with fusion trees, Journal of Computer and System Sciences 47 (3) (1993) 424 – 436. doi:10.1016/0022-0000(93)90040-4.

[2] A. Andersson, P. B. Miltersen, M. Thorup, Fusion trees can be implemented with AC0 instructions only, Theor. Comput. Sci. 215 (1999) 337–344. doi:http://dx.doi.org/10.1016/S0304-3975(98)00172-8.

[3] M. Thorup, On AC0 implementations of fusion trees and atomic heaps, in: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '03, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003, pp. 699–707.