



Basic Research in Computer Science

BRICS RS-95-45

Frandsen & Skyum: Dynamic Maintenance of Majority Information

Dynamic Maintenance of Majority Information in Constant Time per Update

Gudmund Skovbjerg Frandsen
Sven Skyum

BRICS Report Series

RS-95-45

ISSN 0909-0878

August 1995

**Copyright © 1995, BRICS, Department of Computer Science
University of Aarhus. All rights reserved.**

**Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.**

**See back inner page for a list of recent publications in the BRICS
Report Series. Copies may be obtained by contacting:**

**BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk**

**BRICS publications are in general accessible through WWW and
anonymous FTP:**

**`http://www.brics.dk/
ftp ftp.brics.dk (cd pub/BRICS)`**

Dynamic Maintenance of Majority Information in Constant Time per Update*

Gudmund Skovbjerg Frandsen
Sven Skyum

BRICS[†]
Department of Computer Science
University of Aarhus
Ny Munkegade
DK-8000 Aarhus C, Denmark

Abstract

We show how to maintain information about the existence of a majority colour in a set of elements under insertion and deletion of single elements using $O(1)$ time and at most 4 equality tests on colours per update. No ordering information is used.

1 Introduction

We consider the problem of maintaining information about the existence of a majority in a set of elements under insertion and deletion of single elements. The notion of majority is formalised by considering each element to have a colour. If strictly more than half the elements have the same colour, this colour is a majority colour.

*This research was supported by the ESPRIT II BRA Programme of the EC under contract # 7141 (ALCOM II) and by CCI-Europe.

[†]**Basic Research in Computer Science**,
Centre of the Danish National Research Foundation.

Off-line, the existence of a majority colour may be decided in time $O(n \log n)$ by sorting, and several people [1, 3] independently found a linear upper bound and determined that precisely $\lfloor \frac{3}{2}(n-1) \rfloor$ equality tests on colours are needed in the worst case to determine the existence of a majority colour (without the use of any ordering information).

We are not aware of any similar results for the dynamic problem, prior to this paper. For the dynamic problem one may obtain a solution using $O(\log n)$ time per update and $O(1)$ time per query by using ordering information. In this paper we describe a data structure for the optimal bound of $\Theta(1)$ per update and query. We use equality tests on colours, but no ordering information.

1.1 Problem Definition

We let the dynamic majority maintenance problem consist in implementing the following data type.

- memory:
 - S : the set of elements. Initially, $S = \emptyset$.
- operations:
 - Init** : $S \leftarrow \emptyset$;
 - Insert**(e) : $S \leftarrow S \cup \{e\}$;
 - Delete**(e) : $S \leftarrow S - \{e\}$;
 - Query?** : return (yes, c), if there are at least $\lceil \frac{|S|+1}{2} \rceil$ elements in S of some single colour c ;
and return (no) otherwise.

As our model of computation, we use an ordinary unit cost RAM with $O(\log n)$ word size. Colours cannot be used as addresses (otherwise, we would have a trivial $O(1)$ solution). In fact the only allowed way to extract colour information is by making an equality test on two colours.

1.2 Results

We present a solution for the dynamic majority problems that uses time $\Theta(1)$. This solution uses at most 4 equality tests on colours per delete, at

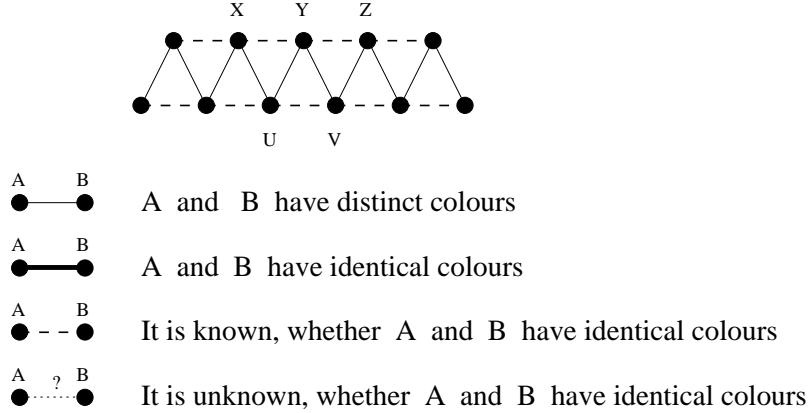


Figure 1: General tri-ladder and symbol explanation

most 3 equality tests per insert and no equality tests on a query. The lower bound of $\lfloor \frac{3}{2}(n-1) \rfloor$ equality tests for the off-line majority problem [1, 3] implies that a single insert requires at least 2 equality tests in the worst case. We present another solution for the dynamic problem that needs at most 2 equality tests per update but requires $O(\log n)$ time in the worst case.

2 A simple constant time construction

2.1 Tri-ladder data structure

Our solution uses a special pointer structure, which we have called a *tri-ladder* for storing the elements of the set (see Figure 1). The tri-ladder stores information about identity and distinctness of colours. The tri-ladder organises the elements in two opposing lists, such that an element Y has four adjacent elements, two opposing elements U, V in the other list, and two neighbours X, Z in the same list. An element may have fewer adjacent elements when placed at the end of one list or beyond the end of the other list. We maintain the following invariant:

- For any pair of adjacent elements it is known whether colours are equal or distinct.
- All pairs of opposing elements have distinct colours.
- The left ends of the two lists are opposed to each other.

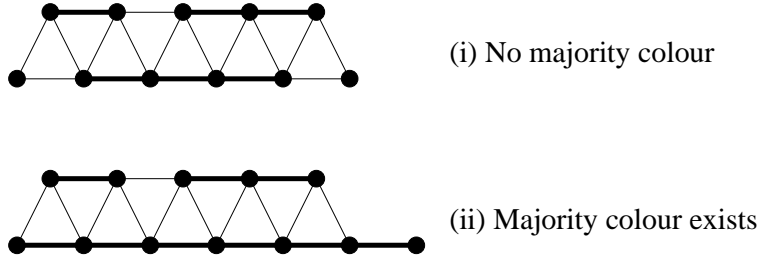


Figure 2: Typical sets with and without a majority colour

- If there is no majority colour, then the length of the two lists differ by at most 1 (see typical tri-ladder in Figure 2).
- If there is a majority colour, then the two list are unequal in length and all the elements in the longer list have the majority colour (see typical tri-ladder in Figure 2).

Any maximal length run of identically coloured elements in one of the two lists is called a *block* (it is possible that a block consists of a single element). A block has pointers between the two endpoints, allowing to go from one endpoint of a block to the other endpoint in constant time.

2.2 The query operation

According to the invariant, there exists a majority colour precisely when

- one list is longer than the other, and
- the elements of the longer list forms a single block.

These criteria can be checked in time $\Theta(1)$ using the tri-ladder representation.

2.3 The insert operation

We shall show how to maintain the tri-ladder structure when inserting a new element. We will assume that there is no majority colour in the present set. The reader should find no difficulty in modifying our construction to the other case, when there is a majority colour.

Figure 3 illustrates the insertion of a new element into a typical tri-ladder. Initially, we compare the colour of the new element Z with the colour of the

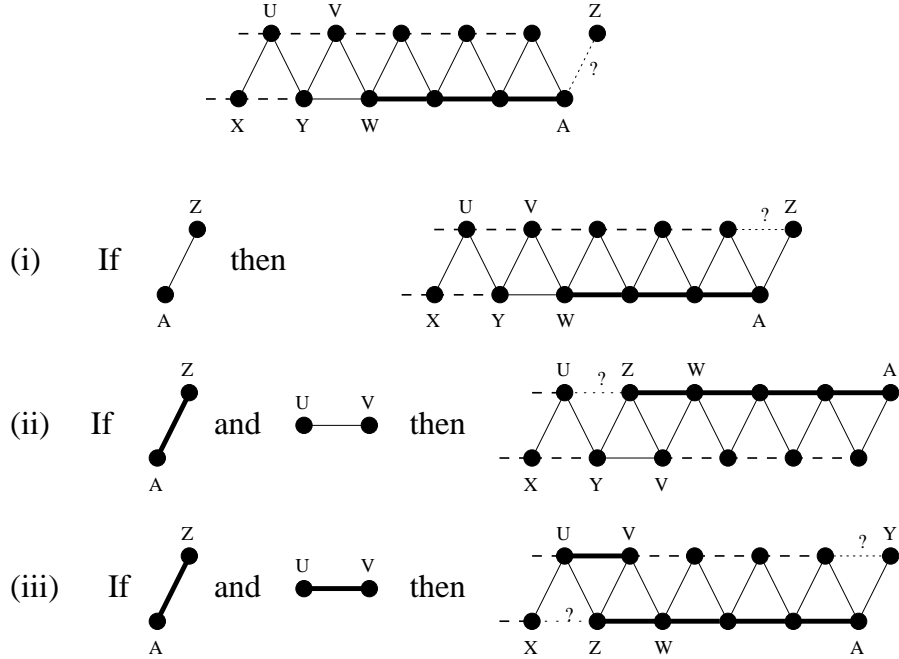


Figure 3: Insertion of a new element Z .

rightmost element A in the current tri-ladder. Depending on the outcome of this comparison and internal colour relations in the old tri-ladder, there are three cases:

- (i) If A and Z have distinct colours, we can simply insert Z at the right end of the tri-ladder. One additional comparison is needed to maintain invariants.
- (ii) If A and Z have identical colours and U and V have distinct colours then we split the tri-ladder to the right of U and Y and turn the right part of the tri-ladder upside down. Z can then be inserted between U and W . This requires one additional comparison to maintain invariants.

In case (ii) the requirement that U and V have distinct colours guarantees that we do not split the tri-ladder in the middle of any block, and therefore the insert is handled in constant time in this case.

- (iii) If A and Z have identical colours and U and V also have identical colours (i.e. U and V could be in the middle of a large block for all

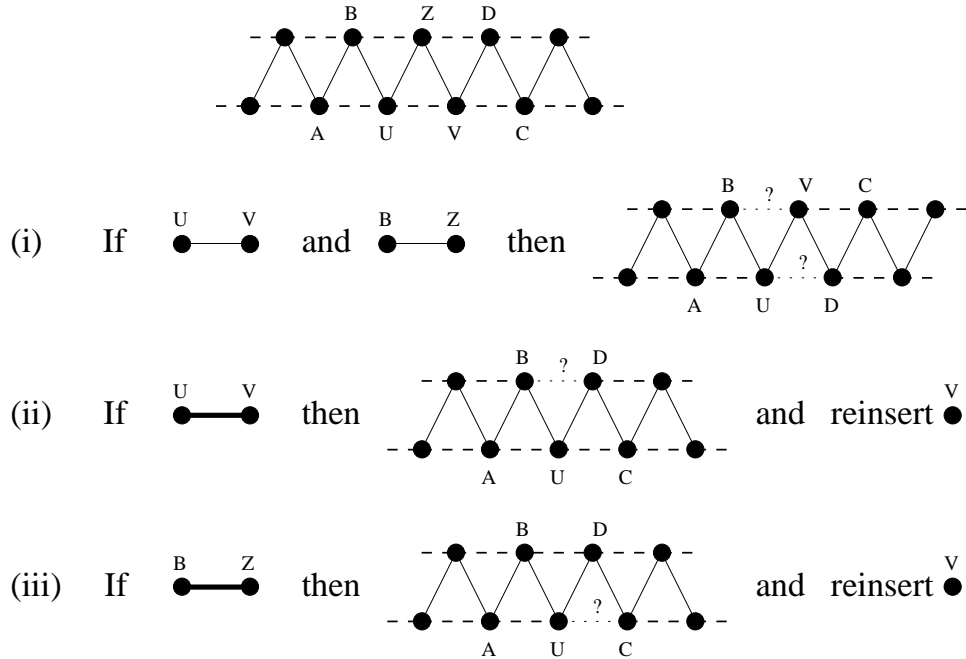


Figure 4: Deletion of an element Z .

we know) then we may replace the element Y next to A 's block with Z and reinsert Y at the right end of the tri-ladder. This requires two additional comparisons to maintain invariants.

It should be clear that a single insertion can be done in time $O(1)$ and uses at most 3 colour comparisons to maintain the tri-ladder structure.

2.4 The delete operation

We shall show how to maintain the tri-ladder structure when deleting an element. Figure 4 illustrates the deletion of an element Z from a typical tri-ladder. There are several cases depending on whether the elements named U and V in the figure have distinct colours or not, and depending on whether Z is at one end of its block.

- (i) If U and V have distinct colours and B and Z also have distinct colours then we split the tri-ladder to the right of B and U and turn the right part of the tri-ladder upside down. The two parts of the tri-ladder are

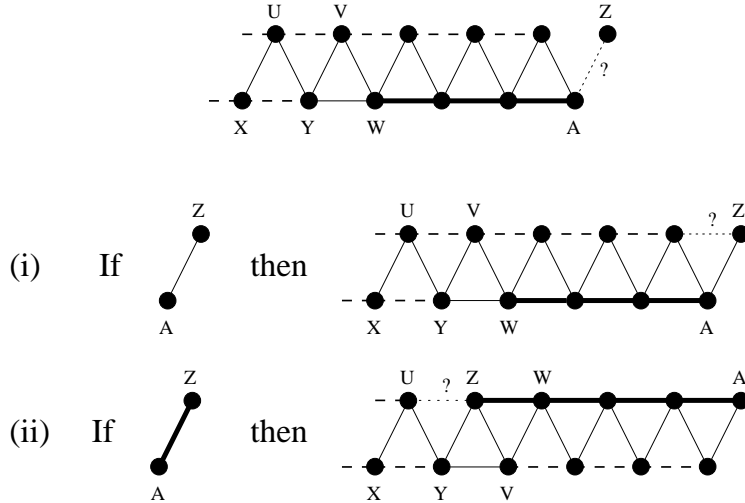


Figure 5: Insertion of Z using 2 comparisons only.

re-connected after removal of Z . Two colour comparisons suffices to maintain invariants.

- (ii) If U and V have identical colours then we remove both Z and V from the tri-ladder, and reinsert V using the method of the previous section. Maintenance of invariants requires one colour comparison in addition to those needed for the reinsertion of V .
- (iii) If B and Z have identical colours, we act similarly to (ii).

It should be clear that a single deletion can be done in time $O(1)$ and uses at most 4 colour comparisons ($4 = \max\{2, 1 + 3\}$) to maintain the tri-ladder structure.

3 Minimising the number of colour comparisons

In some applications a comparison of colours may be relatively expensive (see [2] for a more general study of this issue). We have therefore studied the possibility of handling an update using only two colour comparisons, which is optimal by the results of [1, 3]. We found a solution with this property

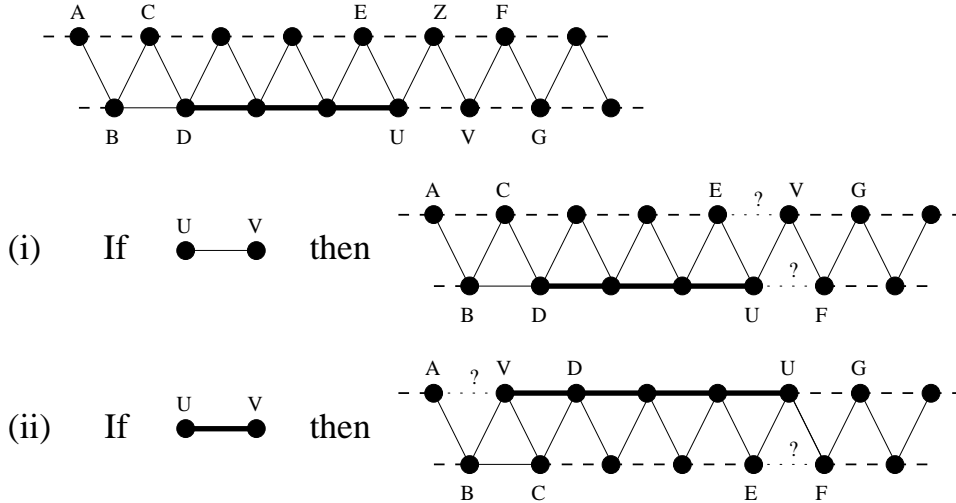


Figure 6: Deletion of Z using 2 comparisons only.

by using a more expensive data structure (that requires $O(\log n)$ time per update).

The excessive number of colour comparisons in our constant time solutions stems from the fact that we are unable to locate the end of a block from somewhere inside the block in constant time, and as a consequence we are also unable to split a block in constant time. However, at the cost of $O(\log n)$ time per operation we may use balanced binary trees to maintain blocks under all necessary operations including split and locate-end.

Assuming availability of the relevant block operations, we illustrate a typical insertion using only two colour comparisons in Figure 5, and we illustrate a typical deletion using only two colour comparisons in Figure 6.

For all we know there may exist a solution that obtain both constant time and 2 colour comparisons per update in the worst case.

References

- [1] Fischer, M. J. and Salzburg S. L., Solution to problem 81-5. *Journal of Algorithms* **3** (1982) 376–379.
- [2] Frandsen, G. S., Miltersen, P. B. and Skyum, S., The complexity of finding replicas using equality tests. In *Proc. Mathematical Foundations*

of Computer Science 1993, LNCS 711, 463–472.

- [3] Matula, D. W., An Optimal Algorithm for the Majority Problem. Manuscript, Southern Methodist University, Texas, 1990.

Recent Publications in the BRICS Report Series

- RS-95-45 Gudmund Skovbjerg Frandsen and Sven Skyum. *Dynamic Maintenance of Majority Information in Constant Time per Update*. August 1995. 9 pp.
- RS-95-44 Bruno Courcelle and Igor Walukiewicz. *Monadic Second-Order Logic, Graphs and Unfoldings of Transition Systems*. August 1995. 39 pp. To be presented at CSL '95.
- RS-95-43 Noam Nisan and Avi Wigderson. *Lower Bounds on Arithmetic Circuits via Partial Derivatives (Preliminary Version)*. August 1995. 17 pp. To appear in *36th Annual Conference on Foundations of Computer Science, FOCS '95, IEEE, 1995*.
- RS-95-42 Mayer Goldberg. *An Adequate Left-Associated Binary Numeral System in the λ -Calculus*. August 1995. 16 pp.
- RS-95-41 Olivier Danvy, Karoline Malmkjær, and Jens Palsberg. *Eta-Expansion Does The Trick*. August 1995. 23 pp.
- RS-95-40 Anna Ingólfssdóttir and Andrea Schalk. *A Fully Abstract Denotational Model for Observational Congruence*. August 1995. 29 pp.
- RS-95-39 Allan Cheng. *Petri Nets, Traces, and Local Model Checking*. July 1995. 32 pp. Full version of paper appearing in *Proceedings of AMAST '95, LNCS 936, 1995*.
- RS-95-38 Mayer Goldberg. *Gödelisation in the λ -Calculus*. July 1995. 7 pp.
- RS-95-37 Sten Agerholm and Mike Gordon. *Experiments with ZF Set Theory in HOL and Isabelle*. July 1995. 14 pp. To appear in *Proceedings of the 8th International Workshop on Higher Order Logic Theorem Proving and its Applications, LNCS, 1995*.
- RS-95-36 Sten Agerholm. *Non-primitive Recursive Function Definitions*. July 1995. 15 pp. To appear in *Proceedings of the 8th International Workshop on Higher Order Logic Theorem Proving and its Applications, LNCS, 1995*.
- RS-95-35 Mayer Goldberg. *Constructing Fixed-Point Combinators Using Application Survival*. June 1995. 14 pp.