A pointer-free data structure for merging heaps and min-max heaps

Giorgio Gambosi¹, Enrico Nardelli², Maurizio Talamo¹,³

(1) Istituto di Analisi dei Sistemi ed Informatica, C.N.R. - Roma, Italy
(2) Dipartimento di Matematica, 2nd University of Rome "Tor Vergata", Roma, Italy
(3) Dipartimento di Informatica ed Applicazioni, University of Salerno, Salerno, Italy

Abstract

In this paper a data structure for the representation of mergeable heaps and min-max heaps without using pointers is introduced. The supported operations are: Insert, Deletemax, Deletemin, Findmax, Findmin, Merge, Newheap, Deleteheap. The structure is analysed in terms of amortized time complexity, resulting in a O(1) amortized time for each operation but for Insert, for which a O(lgN) bound holds.

1. Introduction

The use of pointers in data structures seems to contribute quite significantly to the design of efficient algorithms for data access and management. Implicit data structures [12] have been introduced in order to evaluate the impact of the absence of pointers on time efficiency. Traditionally, implicit data structures have been mostly studied for what concerns the dictionary problem both in 1-dimensional [4, 5, 8, 9, 13] and in multidimensional space [1]. In such papers, the maintenance of a single dictionary has been analysed, not considering the case in which several instances of the same structure (i.e. several dictionaries) have to be represented and maintained at the same time and within the same array-structured memory.

In this paper, the implicit representation of a different data structure, the mergeable heap, is studied. For a general discussion of mergeable heaps see [2, 11, 15]. Heaps are traditionally considered as the first example of implicit structures, since they can be used to easily implement a priority queue by means of an array [17]: the approach introduced can be extended to the implementation of double ended priority queues (dequeues) by means of min-max heaps [3].

The extension of the implicit representation of priority queues to support the merge operation has been considered in [14], where an algorithm is presented for merging two heaps represented in different arrays.

We are interested to introduce a data structure which makes it possible to represent at the same time different instances of mergeable priority queues and dequeues within the same array. Thus, the major goal of this paper is to develop a pointer free data structure for mergeable heaps and min-max heaps under which the basic operations of Insert, Deletemax, Deletemin, Findmax, Findmin, Merge, Newheap, and Deleteheap can be performed efficiently. The approach introduced to obtain such a result is related to the techniques introduced in [6] for the dynamization of decomposable searching problems.

Time complexity will be considered within the paper in an amortized sense [16], i.e. time complexity will be analyzed by averaging a worst case sequence of operations over time.
We shall first consider a data structure for mergeable heaps and then extend it to manage also mergeable min-max heaps, thus settling, in the amortized analysis framework, the open question posed in [3].

The paper is organized as follows: in section 2 the proposed data structure is presented; operations on the structure are described in section 3 and analyzed in section 4. In section 5 extensions and further remarks are given.

2. Description of the data structure

Let \( \mathcal{P}(S) \) be a dynamic search problem defined on a (partially) ordered universe \( S \); let \( \mathcal{D}(\mathcal{P}) \) be a data structure used to represent sets of elements of \( S \) in order to solve \( \mathcal{P} \). Since we focus our attention on pointerless structures, our model for \( \mathcal{D}(\mathcal{P}) \) is a 1-dimensional array on which some suitable partial order relation, depending on \( \mathcal{P} \), can be defined among array locations: the access to the structure will then be performed taking care of such a partial order relation.

Moreover, we are interested in dynamic data structures, in which the number of data items represented by the structure may change with the time. In particular, we are interested to extend the approaches introduced in the framework of implicit data structures design to the simultaneous representation of multiple instances of dynamic data structures, subject, among others, to the operations of melding, creation and deletion of such instances. That is, we are interested to deal at the same time with both a set of instances of a data structure and a time varying set of elements represented in instances of such structures. In the following, we shall denote with \( M \) the current number of instances of data structures and with \( N \) the current overall number of elements represented in the instances.

The need of efficiently dealing with creation and deletion of multiple instances of the same structure makes it necessary to relax the usual definition of implicit data structure [12] by allowing a non contiguous allocation of the data elements in the array.

Anyhow, given \( M \) instances of the structure, we still represent the \( N \) items (and the relations between them) which are present at any time in the structure by using \( N + c \cdot M \) space, where \( c \) is a constant, and using a \( O(1) \) additional space for updating the overall structure. Therefore we do not allow the presence of pointers, that is we do not allow to explicitly represent structural information.

In this paper, we are interested to pointer free representations of a set of mergeable heaps, under which it is possible to efficiently manage a sequence of \texttt{Newheap}, \texttt{Deleteheap}, \texttt{Insert}, \texttt{Findmax}, \texttt{Deletemax}, \texttt{Merge} operations. In particular given \( M \) mergeable heaps and \( N \) elements it is possible to represent them using an \( N+M \) space (\( c=1 \)): this can be obtained if we keep as additional information only the number of elements associated to each heap. Firstly, we consider a simpler approach where \( c=4 \) and subsequently we briefly discuss how it is possible, using some additional technicalities, to obtain \( c=1 \).

Note that the approach for merging heaps introduced in [14] cannot be extended to cope with multiple instances of the same structure. In fact, that approach is based on arrays which are upper-unbounded and therefore are not acceptable in our 1-dimensional model of the memory. If, on the other side, we fix \( N \) and set "a priori" the size of each array to \( N \), then the structure is not able to adapt itself to changes of \( N \) without spending \( O(N) \) time and \( O(N) \) space for restructuring. Moreover the analysis given in [14] for space complexity does not consider the additional \( O(N) \) space needed for merging heaps as space used for element representation.

Let us now start with the definition of the structure: we shall assume \( N=2^k-1 \) for some integer \( k \).