ELSEVIER

Information Processing Letters 69 (1999) 197–199

Information Processing Letters

Symmetric Min–Max heap: A simpler data structure for double-ended priority queue

A. Arvind, C. Pandu Rangan *

Department of Computer Science and Engineering, Indian Institute of Technology, Madras, 600 036, India

Received 27 January 1998; received in revised form 6 November 1998
Communicated by D. Gries

Keywords: Priority queue; Heap; Data structures

1. Introduction

The double-ended priority queue supports the following operations on a set S of elements.

- Insert(S, x): Insert a new element into S.

  $S := S \cup \{x\}$.

- Delete-min(S): Delete the minimum element of S.

  $S := S - \{\min(S)\}$.

- Delete-max(S): Delete the maximum element of S.

  $S := S - \{\max(S)\}$.

Several data structures have been proposed to implement double-ended priority queue operations in $O(\log n)$ time, e.g., Min–Max heap [1], Deap [2], Diamond deque [3] and back-to-back heap [4]. In Min–Max heaps, the even layers form a Min-heap and the odd layers form a Max-heap. Deap has separate Min-heaps and Max-heaps that are built on the left and right subtrees, respectively. In Diamond deque, the Min-heap and Max-heap are not separable.

The separation of the Min- and Max-heaps in [1] and [2] tend to make computations complicated. The Diamond deque has quite complex predecessor and successor relations, and the algorithms for the priority queue operations are quite involved. Another data structure, proposed by J.W.J. Williams, is discussed in [4]. This data structure stores $2n$ elements of a set in two arrays A and B, each containing n elements. Array A is ordered as a

$$\text{Min-heap}\left(A\left[\lfloor i/2 \rfloor\right] \leqslant A[i],\ 1 \leqslant i \leqslant n\right),$$

B as a

$$\text{Max-heap}\left(B[i] \leqslant B\left[\lceil i/2 \rceil\right],\ 1 \leqslant i \leqslant n\right).$$

Moreover there is an additional constraint that $A[i] \leqslant B[i]$, for all $1 \leqslant i \leqslant n$. This introduces certain complications in the algorithms for the priority deque operations. In this paper, we provide a simple data structure for implementing double-ended priority queues that is extremely easy to understand and implement. Our data structure uses only one array, and there is no explicit separation of the Min-heap and Max-heap.

2. Symmetric Min–Max heaps

We call our data structure the *Symmetric Min–Max* (SMM) *heap*.

* Corresponding author. Email: rangan@iitm.ernet.in.

PII: S0020-0190(99)00014-9

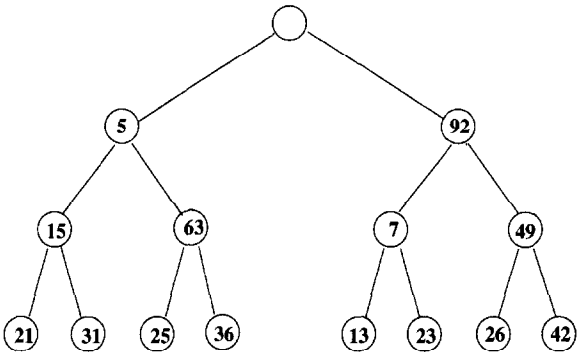

Fig. 1. SMM heap.

The properties satisfied by SMM are:

(1) SMM heap is a heap-like structure, implemented in array of size n.
(2) The root does not contain an element; it is a dummy node.
(3) Every node of the SMM heap satisfies the following property $P1$. For a node X, let T_X denote the subtree rooted at X. Node X satisfies the property $P1$ if (a) the maximum among all the values of subtree T_X (excluding the element at the node X) is found at the right child of X and (b) the minimum element in T_X is found at the left child of X.

An example of an SMM heap is shown in Fig. 1.

A node Y is said to be a *sibling* of node X if X and Y have the same parent. Let N be a node and X be its parent. Let Y be the sibling of X. Let P be the parent of X and Y. X is said to be the *left (right) sibling* of Y, depending on whether X is the left (right) child of P. Node X is called Lnode(N) if X is the left child of its parent. Otherwise it is called Rnode(N). Similarly Y is called Lnode(N) if Y is the left child of P and Rnode(N) if Y is the right child of P. For example, in Fig. 1, let $N1$ be the node containing the value 13, $N2$ the node containing the value 7, and $N3$ the node containing the value 49. Then Lnode($N1$) is node $N2$ and Rnode($N1$) is $N3$. If N is any node of the heap, then the value stored in the node is denoted by key(N). In the algorithms for the priority queue operations given below, A denotes the array in which the SMM heap is implicitly stored. The indices of the array are numbered $1, \ldots, n$. We denote the highest index of array A by the variable *last*.

A node X of the data structure defined above is said to satisfy property $Q1$ if Lnode(X) is not defined or if key(X) $\geqslant$ key(Lnode(X)). Similarly, a node X is said to satisfy property $Q2$ if Rnode(X) is not defined or if key(X) $\leqslant$ key(Rnode(X)).

Lemma 1. *Let H be a heap with one value stored in each node of H, except the root. H is an SMM heap iff*
(1) *each node of H satisfies both $Q1$ and $Q2$ and*
(2) *the value in the* right sibling *is greater than the value in the* left sibling.

Proof. Follows easily by induction. □

3. Insertion

The Insert operation is similar to the usual heap insertion. The new element is bubbled up to its appropriate position from the position in which it is inserted. We give a brief description of the insert procedure. Let x be the value to be inserted. Let X denote the node in which x was stored.

1. Increment *last* by 1 and insert x in the position pointed to by *last*.
2. If x is smaller than its *left sibling* (if it has one) then swap the two.
3. stop := false
4. **while** ¬stop
   if $x <$ key(Lnode(X)) **then** swap(X, Lnode(X))
   else if $x >$ key(Rnode(X))
   then swap(X, Rnode(X))
   else stop := true

Correctness and complexity

When the insertion procedure terminates, all the nodes satisfy properties $Q1$ and $Q2$. This follows immediately from the termination condition for the loop at step 4. Using Lemma 1 it follows that when the insertion procedure terminates it results in an SMM heap. It is clear that the complexity of insert is O($\log n$).

4. Delete-min

The Delete-min operation is quite similar to the Delete-min operation of the conventional heap. The

minimum element is located and the element in the last position is moved to this position and bubbled down. We leave this procedure to the reader to implement.

Acknowledgement

We would like to thank an anonymous referee for pointing out reference [4] to us. We would also like to thank Professor David Gries for his numerous suggestions that greatly improved the presentation of this paper.

References

[1] M.D. Atkinson, J.-R. Sack, T. Strothotte, Min–Max heaps and generalized priority queues, Comm. ACM 29 (1986) 996–1000.
[2] S. Carlsson, The Deap—a double-ended heap to implement double-ended priority queues, Inform. Process. Lett. 26 (1987) 33–36.
[3] S.C. Chang, M.W. Du, Diamond deque: A simple data structure for priority deques, Inform. Process. Lett. 46 (1993) 231–237.
[4] D.E. Knuth, The Art of Computer Programming, Vol. 3, Addison-Wesley, Reading, MA, 1973, Problem 31, p. 159.