WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Add links

Double-ended priority queue

From Wikipedia, the free encyclopedia

Not to be confused with Double-ended queue.

In computer science, a **double-ended priority queue (DEPQ)**[1] or **double-ended heap**[2] is a data structure similar to a priority queue or heap, but allows for efficient removal of both the maximum and minimum, according to some ordering on the *keys* (items) stored in the structure. Every element in a DEPQ has a priority or value. In a DEPQ, it is possible to remove the elements in both ascending as well as descending order.[3]

Contents

Operations [edit]

A double-ended priority queue features the follow operations:

isEmpty()
Checks if DEPQ is empty and returns true if empty.

size()
Returns the total number of elements present in the DEPQ.

getMin()
Returns the element having least priority.

getMax()
Returns the element having highest priority.

put(*x*)
Inserts the element *x* in the DEPQ.

removeMin()
Removes an element with minimum priority and returns this element.

removeMax()
Removes an element with maximum priority and returns this element.

If an operation is to be performed on two elements having the same priority, then the element inserted first is chosen. Also, the priority of any element can be changed once it has been inserted in the DEPQ.[4]

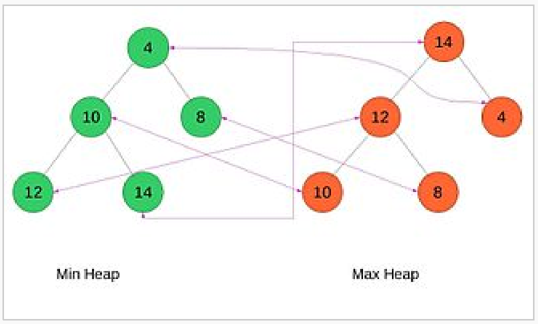
Implementation [edit]

Double-ended priority queues can be built from balanced binary search trees (where the minimum and maximum elements are the leftmost and rightmost leaves, respectively), or using specialized data structures like min-max heap and pairing heap.

Generic methods of arriving at double-ended priority queues from normal priority queues are:[5]

Dual structure method [edit]

In this method two different priority queues for min and max are maintained. The same elements in both the PQs are shown with the help of correspondence pointers.
Here, the minimum and maximum elements are values contained in the root nodes of min heap and max heap respectively.

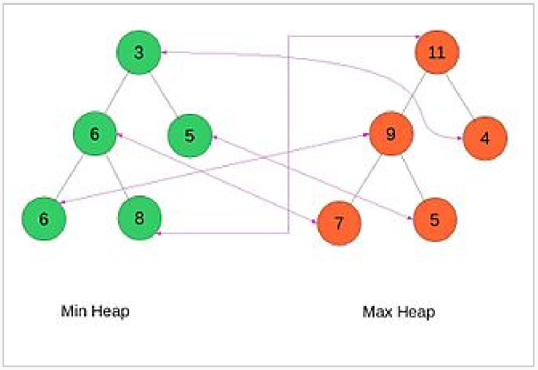


A dual structure with 14,12,4,10,8 as the members of DEPQ.[1]

- **Removing the min element**: Perform removemin() on the min heap and remove(*node value*) on the max heap, where *node value* is the value in the corresponding node in the max heap.
- **Removing the max element**: Perform removemax() on the max heap and remove(*node value*) on the min heap, where *node value* is the value in the corresponding node in the min heap.

Total correspondence [edit]

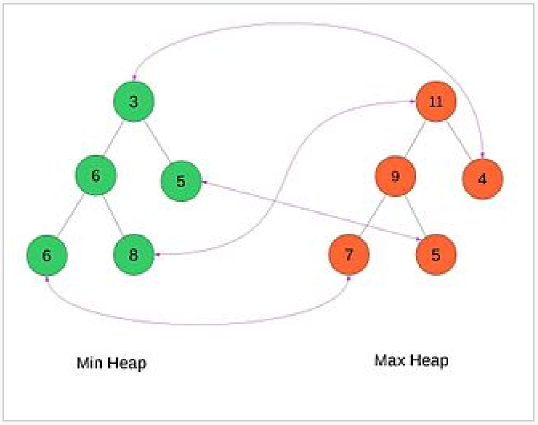
Half the elements are in the min PQ and the other half in the max PQ. Each element in the min PQ has a one to one correspondence with an element in max PQ. If the number of elements in the DEPQ is odd, one of the elements is retained in a buffer.[1] Priority of every element in the min PQ will be less than or equal to the corresponding element in the max PQ.



A total correspondence heap for the elements 3, 4, 5, 5, 6, 6, 7, 8, 9, 10, 11 with element 11 as buffer.[1]

Leaf correspondence [edit]

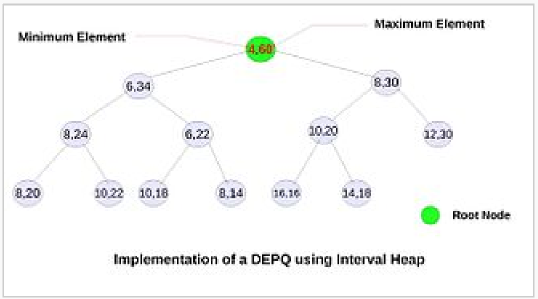
In this method only the leaf elements of the min and max PQ form corresponding one to one pairs. It is not necessary for non-leaf elements to be in a one to one correspondence pair.[1]



A leaf correspondence heap for the same elements as above.[1]

Interval heaps [edit]

Apart from the above mentioned correspondence methods, DEPQ's can be obtained efficiently using interval heaps.[6] An interval heap is like an embedded min-max heap in which each node contains two elements. It is a complete binary tree in which:[6]



Implementing a DEPQ using interval heap.

- The left element is less than or equal to the right element.
- Both the elements define a closed interval.
- Interval represented by any node except the root is a sub-interval of the parent node.

- Elements on the left hand side define a min heap.
- Elements on the right hand side define a max heap.

Depending on the number of elements, two cases are possible[6] -

1. **Even number of elements:** In this case, each node contains two elements say *p* and *q*, with *p* ≤ *q*. Every node is then represented by the interval [*p*, *q*].
2. **Odd number of elements:** In this case, each node except the last contains two elements represented by the interval [*p*, *q*] whereas the last node will contain a single element and is represented by the interval [*p*, *p*].

Inserting an element [edit]

Depending on the number of elements already present in the interval heap, following cases are possible:

- **Odd number of elements:** If the number of elements in the interval heap is odd, the new element is firstly inserted in the last node. Then, it is successively compared with the previous node elements and tested to satisfy the criteria essential for an interval heap as stated above. In case if the element does not satisfy any of the criteria, it is moved from the last node to the root until all the conditions are satisfied.[6]
- **Even number of elements:** If the number of elements is even, then for the insertion of a new element an additional node is created. If the element falls to the left of the parent interval, it is considered to be in the min heap and if the element falls to the right of the parent interval, it is considered in the max heap. Further, it is compared successively and moved from the last node to the root until all the conditions for interval heap are satisfied. If the element lies within the interval of the parent node itself, the process is stopped then and there itself and moving of elements does not take place.[6]

The time required for inserting an element depends on the number of movements required to meet all the conditions and is O(log *n*).

Deleting an element [edit]

- **Min element:** In an interval heap, the minimum element is the element on the left hand side of the root node. This element is removed and returned. To fill in the vacancy created on the left hand side of the root node, an element from the last node is removed and reinserted into the root node. This element is then compared successively with all the left hand elements of the descending nodes and the process stops when all the conditions for an interval heap are satisfied.In case if the left hand side element in the node becomes greater than the right side element at any stage, the two elements are swapped[6] and then further comparisons are done. Finally, the root node will again contain the minimum element on the left hand side.
- **Max element:** In an interval heap, the maximum element is the element on the right hand side of the root node. This element is removed and returned. To fill in the vacancy created on the right hand side of the root node, an element from the last node is removed and reinserted into the root node. Further comparisons are carried out on a similar basis as discussed above. Finally, the root node will again contain the max element on the right hand side.

Thus, with interval heaps, both the minimum and maximum elements can be removed efficiently traversing from root to leaf. Thus, a DEPQ can be obtained[6] from an interval heap where the elements of the interval heap are the priorities of elements in the DEPQ.

Time Complexity [edit]

Interval Heaps [edit]

When DEPQ's are implemented using Interval heaps consisting of *n* elements, the time complexities for the various functions are formulated in the table below[1]

Operation	Time Complexity
init( )	O(n)
isEmpty( )	O(1)
getmin( )	O(1)
getmax( )	O(1)
size( )	O(1)
insert(x)	O(log *n*)
removeMin( )	O(log *n*)
removeMax( )	O(log *n*)

Pairing heaps [edit]

When DEPQ's are implemented using heaps or pairing heaps consisting of *n* elements, the time complexities for the various functions are formulated in the table below.[1] For pairing heaps, it is an amortized complexity.

Operation	Time Complexity
isEmpty( )	O(1)
getmin( )	O(1)
getmax( )	O(1)
insert(x)	O(log *n*)
removeMax( )	O(log *n*)
removeMin( )	O(log *n*)

Applications [edit]

External sorting [edit]

One example application of the double-ended priority queue is external sorting. In an external sort, there are more elements than can be held in the computer's memory. The elements to be sorted are initially on a disk and the sorted sequence is to be left on the disk. The external quick sort is implemented using the DEPQ as follows:

1. Read in as many elements as will fit into an internal DEPQ. The elements in the DEPQ will eventually be the middle group (pivot) of elements.
2. Read in the remaining elements. If the next element is ≤ the smallest element in the DEPQ, output this next element as part of the left group. If the next element is ≥ the largest element in the DEPQ, output this next element as part of the right group. Otherwise, remove either the max or min element from the DEPQ (the choice may be made randomly or alternately); if the max element is removed, output it as part of the right group; otherwise, output the removed element as part of the left group; insert the newly input element into the DEPQ.
3. Output the elements in the DEPQ, in sorted order, as the middle group.
4. Sort the left and right groups recursively.

See also [edit]

- Queue (abstract data type)
- Priority queue
- Double-ended queue

References [edit]

1. ^ *a b c d e f g h* Data Structures, Algorithms, & Applications in Java: Double-Ended Priority Queues, Sartaj Sahni, 1999.
2. ^ Brass, Peter (2008). *Advanced Data Structures*. Cambridge University Press. p. 211. ISBN 9780521880374.
3. ^ "Depq - Double-Ended Priority Queue".
4. ^ "depq".
5. ^ Fundamentals of Data Structures in C++ - Ellis Horowitz, Sartaj Sahni and Dinesh Mehta
6. ^ *a b c d e f g* http://www.mhhe.com/engcs/compsci/sahni/enrich/c9/interval.pdf

v · t · e	Data structures
Types	Collection · Container
Abstract	Associative array · **Double-ended priority queue** · Double-ended queue · List · Map · Multimap · Priority queue · Queue · Set (multiset) · Disjoint Sets · Stack
Arrays	Bit array · Circular buffer · Dynamic array · Hash table · Hashed array tree · Sparse array
Linked	Association list · Linked list · Skip list · Unrolled linked list · XOR linked list
Trees	B-tree · Binary search tree (AA · AVL · red-black · self-balancing · splay) · Heap (binary · binomial · Fibonacci) · R-tree (R* · R+ · Hilbert) · Trie (Hash tree)
Graphs	Binary decision diagram · Directed acyclic graph · Directed acyclic word graph
	List of data structures

Categories: Abstract data types | Priority queues