Artificial Intelligence

Exercise 4.9, *Computational Intelligence*, page 166

Priority queues are important to implement search algorithms. The use of sorted lists is very inefficient. This exercise is to implement a more efficient version of priority queues. The functionality you will need to provide is

- insertion of an element into a priority queue
- removal of the smallest element from the priority queue.

Removal should fail if the queue is empty.

Insertion and deletion must both be done in \( \log n \) time where \( n \) is the number of elements in the priority queue.

*Hint*: Represent a priority queue as a binary tree with the following properties:

- The smallest element of the tree rooted at any node is at that node.
- At every node, the left subtree has either the same number of elements or has exactly one more element than the right subtree. To maintain this invariant upon insertion, insert a new element into the right subtree and swap the left and right subtrees. Do a similar operation on deletion.

Test this program by using it for heap sort, a sorting algorithm that involves putting all of the elements to be sorted onto a priority queue, and removing them in order.

Further Details

Let us agree to represent a priority queue as either the empty list \([\phantom{\text{item}}]\), or a term \( t(Root, LeftPQ, RightPQ) \). For simplicity, let us assume that items in the queue are natural numbers, ordered relative to the usual relation \(<\) (a built-in infix predicate in Prolog). Your task is to define suitable Prolog predicates

\[
\% \text{insert}(+, \text{Item}, +, \text{PQueue}, ?, \text{NewPQ}) \\
\% \text{remove}(+, \text{PQueue}, ?, \text{Item}, ?, \text{NewPQ})
\]

where + indicates that the argument should be instantiated to a non-variable input, while ? indicates that it need not. To get started, here are the easy (base) clauses.

\[
\text{insert}(\text{Item}, [\phantom{\text{item}}], t(\text{Item}, [\phantom{\text{item}}], [\phantom{\text{item}}])). \\
\text{remove}(t(\text{Root}, [\phantom{\text{item}}], [\phantom{\text{item}}]), \text{Root}, [\phantom{\text{item}}]).
\]

To test your clauses, it will be convenient to define

\[
\% \text{listToPq}(+, \text{List}, ?\text{PQueue}) \\
\% \text{pqToList}(+, \text{PQueue}, ?\text{List})
\]

from which \text{heap-sort} can be derived as

\[
\text{heapSort}(\text{List}, \text{SortedList}) :- \text{listToPq}(\text{List}, \text{PQueue}), \\
\text{pqToList}(\text{PQueue}, \text{SortedList}).
\]