TrinaryHeap

Problem being addressed

We often have some tasks or items ranked by priority such that tasks with priority need to be completed first. One example is that Dijkstra's algorithm for shortest paths using a priority queue to select which node of the graph to visit next. One efficient method of implementing a priority queue is to store priorities and tasks in an array with a tree-based layout called a heap. Most heaps are binary, that is every node has two children. However, we can reduce the height of the tree and therefore the number of balancing operations required by using a trinary heap where every node has three children.

Data Structure

The nodes array is an array that stores IntNode objects, each with an integer priority and integer item. The position in this array represents position in the trinary heap. Node 0 is the root, nodes 1, 2 and 3 are the children of node 0, and so on. In general, the children of Node i are stored at 3i+1, 3i+2, and 3i+3. This means the parent of a node i is stored at (i-1)/3.

The TrinaryHeap fulfilles the min-heap property. This means that lower numbers are preferred and the children of any node have larger or equal priority to their parent. In particular, the smallest priority node is always at the root so it can be quickly found. Maintaining the min-heap property is necessary to provide efficient operations.

The TrinaryHeap supports several operations including constructing the TrinaryHeap from ArrayLists or arrays, adding items to the heap with a given priority, removing the minimum priority item, determining whether the heap contains a given item, and finding the location of an item with a designated priority.

add(int priority, int item) first adds the item to the end of the nodes array. If the nodes array is full we first replace it with a larger array containing the same values. We then compare its priority values to its parent and swap their position if item has a smaller priority value. The item moves up the heap until it reaches a position where its parent has an equal or smaller priority in a process called tricking up.

removeMin() first removes the minimum priority item and then replaces it with the item from the end of the nodes array. We then compare priority values of the replacement and its children and swap downward until it reaches a position such that all of its children have a larger or equal priority in a process called trickling down.

contains(int item) searches the heap for the designated item and returns true if it is found and false if it is not found. This process uses a recursive containsHelper method to search each subtree starting at the root.

indexOf(int priority) searches the heap for an item with the designated priority and returns its index in the nodes array if the item is found. Otherwise it returns -1 if there is no item with that priority in the TrinaryHeap. This process uses a recursive indexOfHelper method to search each subtree. The parent is searched. Of the children then the first child (3i+1) is searched first, then the second, then the third. This is called a preorder traversal. If there are multiple items with the same priority then the one found first in preorder will be returned.

Example

Figure 1 shows an example of a series of insertions using a TrinaryHeap.

Characteristics of TrinaryHeap

- TrinaryHeaps are sorted with respect to parents and children. There is no specific ordering of siblings.
- TrinaryHeaps do not ensure that added items are unique in the array.
- A TrinaryHeap stores data contiguously from 0 to size in the nodes array.
- TrinaryHeaps have a set capacity and increase their size when the capacity is reached.
- The item with smallest priority value is at the root at position 0.

Required functionality

1. You can create an empty TrinaryHeap, or from an existing priority array and item array, or from two such ArrayLists.
2. You build the TrinaryHeap by applying the add operation in order to each item from the arrays or ArrayLists.
3. You can search the TrinaryHeap to see if it contains a particular item or to get the index of the first instance of a particular priority of a particular item in the TrinaryHeap
4. You can remove the item with minimum priority
5. You can report on the number of items stored in the TrinaryHeap
6. You can convert the TrinaryHeap to some other useful forms, including extracting sub trees of stored items as a new TrinaryHeap.

Hints

- If you insert nodes in increasing (smallest to largest) sorted priority order, the heap will stay in sorted priority order. This may help when constructing a TrinaryHeap from a list of priorities and items.
- removeMin removes the node with the smallest priority and returns its item. This means that repeatedly calling removeMin will return the items in sorted priority order.

TrinaryHeap with capacity 4	i=0	i=1	i=2	i=3	
Add (3,7)	(3,7)				Priority = 3, item = 7, added at root
Add (0,4)	(5,7)	(0,4)			Added to end (child 1 of root) and trickles up
	(0,4)	(5,7)			Compared to parent and swapped
Add (3,9)	(0,4)	(5,7)	(3,9)		Added to end (child 2 of root) and trickles up
	(0,4)	(5,7)	(3,9)		Compared to parent and not swapped
Add (4,6)	(0,4)	(5,7)	(3,9)	(4,6)	Added to end (child 3 of root) and trickles up
	(0,4)	(5,7)	(3,9)	(4,6)	Compared to parent and not swapped
removeMin		(5,7)	(3,9)	(4,6)	Returns item = 4
	(4,6)	(5,7)	(3,9)		Moves last item to root and trickles down
	(4,6)	(5,7)	(3,9)		Compares to first child and not swapped
	(3,9)	(5,7)	(4,6)		Compares to second child and swapped
Contains(9)	(3,9)	(5,7)	(4,6)		Returns true because (3,9) has item = 9
Contains(4)	(3,9)	(5,7)	(4,6)		Returns false because no item = 4
indexOf(4)	(3,9)	(5,7)	(4,6)		Returns 2, the index of (4,6)
indexOf(9)	(3,9)	(5,7)	(4,6)		Returns -1 because no item has priority 9
removeMin	(4,6)	(5,7)			Returns item = 9, replaces root with last, trickles down
removeMin	(5,7)				Returns item = 6, replaces root with last, trickles down
removeMin					Return item = 7, nothing to replace root

Figure 1 Example operation sequence on a TrinaryHeap of capacity 4.