Nicolas Ramirez
CMPT 225
David Mitchell
Assignment 3

Assignment 3 Report

For this project, we have made a Priority Queue Class using an AVL tree and a Binary heap. As implemented in my project, I have created the PQ Class which contains an AVL Tree Object and a Binary Heap object. The AVL tree class objects have the same members and methods that were created by the professor for the first assignment, however I have added a few helper functions to the class. The binary heap object has two parallel vectors. The first has the priorities of each TaskID and the second has the pointers to the corresponding nodes in the AVL Tree that hold the Task ID and the corresponding vector element index.

They are the following:

AVL Tree:

updatePriority(): This function takes in a pointer to the node to be modified with the task Id and the updated array index. This function is called after the heap vectors have been percolated and the new vector indices have changed.

Find(): This helper function finds the heap index corresponding to a given task id found in the AVL tree.

DeleteMin(): this function is not fully functional, however, what was intended was this function takes in a pointer to the node to be deleted, stores the value of the Task ID of the node to be deleted, and then deletes the node, and updates the tree connections depending on if the node deleted has only one child, no children or two children. After the connections have been restored, the AVL tree would call the balance() function as defined by the professor in the first assignment.

Insert(): This function takes place after the given node has been inserted into the heap vector. This function takes in the taskID of the new node and the array index of this node present in the heap vectors and begins traversing the tree starting at the root. Then, This function calls the Findpointer() function and returns a pointer to the newly inserted node to be inserted into the vector of pointers in the heap.

Findpointer():this is a helper function that finds the pointer to the node with a given task ID. It returns this pointer.

Binary Heap:

Insert(): this function takes in a pointer to the node newly inserted and the corresponding priority and pushes each back into the corresponding vector. Pointers go into the BinaryHeap.Pointers vector and the priorities go into the BinaryHeap.Priorities vector. Afterwards the heap is percolated upwards to recover the min heap property.

PercolateUp(int heapindex) and percolateDown(int heapindex): these functions follow the logic provided in the slides. The difference is that these take in the heap index at which we would like to begin percolating at and return the index at which the percolation finishes. This index is where newly inserted nodes are left. This index is usually passed to the AVL tree Insert function.

Nicolas Ramirez
CMPT 225
David Mitchell
Assignment 3
PercolateUp() and percolateDown(): these are the same as the previous, however the percolate through the entire vector and do not return.

Deletemin(): this function does work. It simply accesses the 0th element in each vector and deletes them. Afterwards we percolate fully through the vector restoring min heap properties.

Findmin(): This function finds the pointer of the element with the minimum priority (which is the 0th element in each vector) and returns a reference to the AVL node pointed to by the pointer at that vector index.

isEmpty(): this function checks if the size of both vectors is 0 and returns true if so.

size(): this function checks If the size of both vectors are equal and, if so, returns the size of the pointers vector.

makeempty(): When called, this function clears both vectors completely. If they are already empty an error message is outputted in the terminal.

updatePriority(): This function takes in the index of the node to be modified and the priority it should be modified to. From here it stores a copy of the priority at this index and changes the priority at the index with the new priority. It then compares the old and new priorities and calls percolate up or down depending whether if the old priority is greater or less than the new priority. Lastly this function returns the index of the updated node which is usually passed to the AVL update priority function to update the node in the AVL tree.

PQ Class:

This class calls the methods of the AVL tree and Binary heap objects as described above when necessary.

Insert(): this function takes in a task ID and its corresponding priority and inserts the priority to the last element in the priorities vector of the binary heap object of this PQ object. Afterwards, this calls the AVL insert method which takes the task ID and the index of the newly inserted priority and creates a new node in the AVL tree with these two values. Lastly the AVL insert function returns a pointer to the newly instated node which is pushed into the pointers vector of the binary heap object.

Deletemin(): This function stores the taskID of the node to be deleted in a temporary variable. Afterwards the AVL Tree deleteMin() method is called. Then, the deleteMin() method of the Binary heap is called. In this way the minimum node is deleted from both structures and lastly a reference to the temporary variable holding the Task Id of the deleted node.

findMin(): this function calls the Binary Heap findMin() mthod and returns a reference to the node in the AVL tree with the minimum Priority.

isEmpty(): this functions calls the Binary Heap isEmpty() method and returns a boolean depending on if the both vectors have size 0 or not.

size(): this function calls the Heap size() method and returns the size of the heap.

Nicolas Ramirez
CMPT 225
David Mitchell
Assignment 3
makeempty(): this function calls the makeEmpty() methods of both the Binary Heap object and the AVL tree object. Each method eliminates all elements of their respective objects.

updatePriority(): This function calls the find() method of the AVL tree and stores the index in the vector of the node to be modified. Afterwards, the Heap updatePriority() method with this old index and we then store the new index returned by the heap method. Finally, we call the updatePriority() method from the AVL tree to update the index of the modified node.

DisplayPQ(): this method traverses the vector arrays of the Binary heap object outputting each task ID pointed to by each pointer in the pointers vector and the priority in the priorities vector at the corresponding index. Additionally it calls the displayLinks() method of the AVL tree which outputs a diagram of the nodes present in the tree and the memory addresses corresponding to each node.