Gebze Technical University
Computer Engineering

CSE 222
2017 Spring

HOMEWORK 6 REPORT

VAKHID BETRAKHMADOV
141044086

Course Assistant:

1. Class Diagrams

Class Diagrams.pdf

2. Problem Solutions Approach

Q1.
BinaryHeap is a complete tree, and all addition and removal operations in a heap are done just like in a complete tree. So, I extended BinaryTree in my CompleteBinaryTree class and implemented add method in a proper way. Because of the node like underlining structure of the BinaryTree this add method works in linear time, because first we need to find the most left empty position on the very last level in order to add new element. This is done using queue data structure, just like in level order traverse in hw5. I have modified Node class in the BinaryTree, and added parent link, so that we could move nodes around in the BinaryHeap (swap parent and child) . Offer method in BinaryHeap uses CompleteBinaryTree add method to add new element to the tree, and then moves this element up the tree until right position is found. Poll method in the BinaryHeap fist finds the most right non-empty node, sets root to its value, and then removes this node. Then root moves down the tree swapping position with one of its children until right position is found. This method also takes linear time. Peek method in the BinaryHeap just returns data of the root, so it works in constant time. Method add, remove and element of the BinaryHeap are just adapter methods of the offer, poll and peek. So,all add or remove operations in our BinaryHeap take linear time, this is because to the Node like underling structure. We could have had log(n) addition and removal operation efficiency if used arrays as underling structure in our BinaryHeap.

Q2.
We can implement encode method in two ways. First one takes constant time and just looks up character code value in the map, which is created with the Huffman Tree is build. So, we create map only once, and then just check the character code value in it. Second method which works in linear time searches character code value recursively in the Huffman Tree. If we want to encode a string of characters of size n it will take O(n) time with the method using map, and O(n^2) with the method searching character codes in the Huffman tree.

Q3.
We can implement level-order traverse of a tree using queue data structure, by
offering root node to the queue and polling it back subsequently offering left and right nodes and so on. This is done just the same ways as in he BinarySearch tree in the hw5.

3. Test Cases

Q1.
I have changed the Passenger class, to include priority and type of the passenger, I have used only one queue (BinaryHeap) to keep passengers, where passengers are inserted to a proper position in the queue according to their priorities. In the simulations, I used higher priority for Frequent Flyers and lower one for Regular passengers. So frequent flyers get served before most of the regular passengers.

Q2.
I have encoded and then decoded few strings of characters and got the very same strings of characters, which shows that both methods work correctly.

Q3.
I have constructed a Family Tree and printed it to the screen, then I have traversed it in the level-order using level-order iterator. After cross-checking results, it’s clear that level-order iterator works as expected.