COMP 352- Data Structure & Algorithms
Assignment 3

By Édouard Gagné
ID#: 40061204
Section: AA

16 June 2018

1. a) The general tree structure has two classes : Tree.java and Node.java. The tree class only has a root attribute which is a node, constructors and setters and getters for the root. The node class has four attributes: a value and a key which are integers and left and right pointers to nodes. It contains different constructors as well as setters and getters for each attributes.

b)

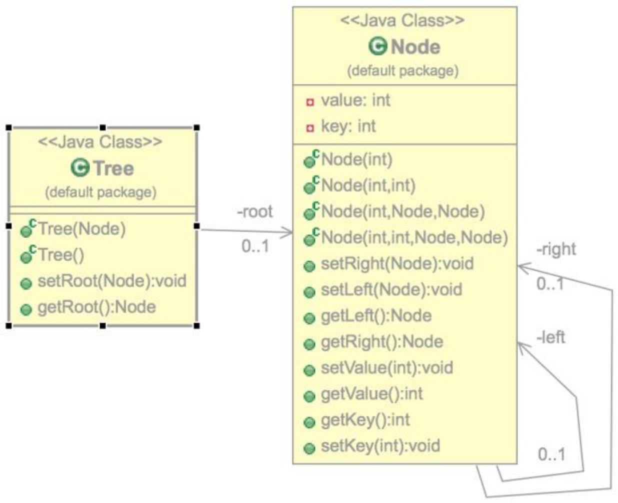

Figure 1: UML diagram for the general tree design

c) Each node of the tree has a left and right pointers since a Huffman coding tree and splay tree are implementations of binary search trees, therefore each node can only have two children. They each have a value which is the frequency in the Huffman tree and a number in the splay tree. They also have a key which represent the character in the Huffman tree but is not used in the splay tree. Different constructors allow the creation of node object and the setters and getters allow access to the attributes. Finally, the root attribute of the tree class marks the start of the tree and allow access to all the nodes in the tree.

d)

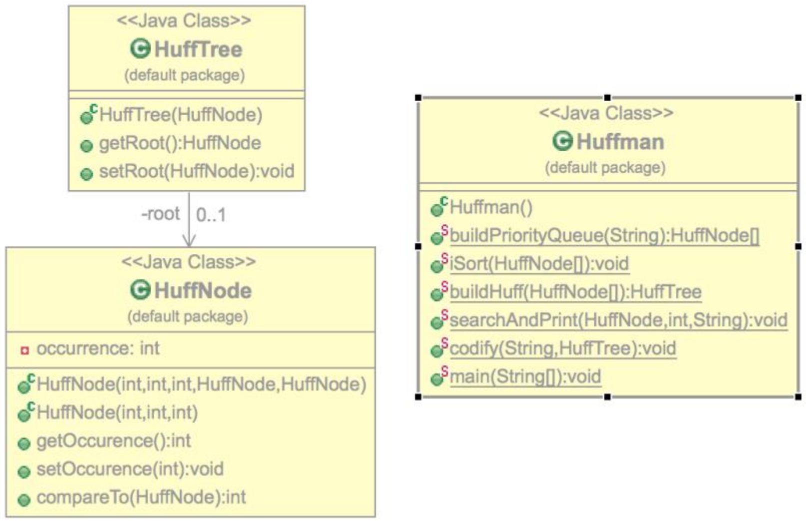

Figure 2: UML diagram for the Huffman tree design

e) The HuffNode class inherits the value, key, left and right attributes from the Node class as well as all the supporting methods. It has an additional attribute named "occurrence" that stores the order in which a character is encountered in a source text. It has some setters and getters for this attribute as well as additional constructors and a compareTo which compare the values of two nodes and the occurrence of two nodes if their values are equal. This will help on a lot of method in the Huffman class. The HuffTree inherits the root from the Tree which is now a HuffNode but does not require other attributes or method. The Huffman class has methods that allows the creation and manipulation of HuffNode and HuffTree objects as well as containing the main method. The buildPriorityQueue method builds an array containing sorted HuffNode objects with frequencies and occurrences extracted from the file parsed as a parameter. The nodes in the queue are sorted using the iSort method that sort an array using the insertion sort algorithm. The buildHuff class will build a Huffman Tree using a priority queue and return a built HuffTree object. The method codify will execute the searchAndPrint method on each element of a string, which is a method that iterate through a Huffman tree and print the code corresponding to the parsed character. The main method uses these methods to build a tree using a parsed file name and then ask the user to encode a string using the Huffman tree encoding.

f)

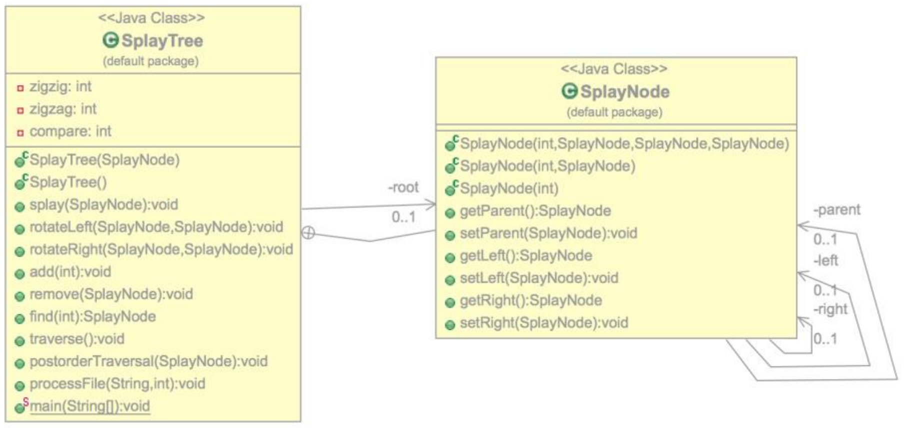

Figure 3: UML diagram for the Splay Tree design

g) The SplayTree class has an inner SplayNode class that extends the Node class as well as having a new attribute named "parent" that point to its parent in the tree structure. This will allow the access of a node's parent to be easier than recursively going to the tree. The SplayTree class has tree new attributes: the number of zigzag, the number of zigzags and the number of compare used to build a Splay Tree. These attributes are all incremented and decremented in the following methods. The splay method takes a splay node as an argument and splay it to the root of the tree or one of its children. It does so by executing multiple series of rotations using the rotateLeft and rotateRight classes depending on how it is positioned in respect to its parent and grandparent until the node is either at the root or is one of its children. The rotateLeft classes rotates a node to the left and the rotateRight classe rotates it to the right. The splay method is used in the add method which adds a node to the tree. If the tree has no node, the tree is initialized with the node as its root, otherwise the method finds where the node is supposed to go thanks to the binary tree property, insert it and splay it. The find method finds a node in the tree using again the binary search tree property and splay it. The remove method takes a node usually found using the find method and removes it depending on three cases. If it has two children, the node swaps its value with the smallest value of its right sub tree and delete this node, then splay its parent. If it has one children, the node is deleted, and its child is connected to its parent. If it has no parent, it is simply deleted. The processFile method will iterate a file containing a serie of command and execute either add, remove or find methods. If the step argument is the same as the current iterative step in the method, a post-order traversal is executed using the traverse method. This method will also print the number of zig-zigs, zig-zags and compares executed by the process. The traverse method execute the postorderTraversal method on the root, which will print the post order traversal of the tree. The main method will process the file parsed as an argument.

h) The Splay Tree and AVL Tree are both implementations of a Binary Search Tree that try to maximize performances. In the AVL Tree, the tree is always height-balanced which allows a guaranteed fast lookup time of complexity O(logn). The Splay Tree cannot guarantee this as it is not always perfectly height balanced but compensate having a reduced total cost for series of operations. This can be helpful to solve question 3 since there possibly could be a big number of operations in the Operations.txt file needing to be executed.

4. a) Huffman

The string is : “she was too short to see over the fence.”. Huffman Encoding: 11100100 01111000 10011001 11100001 01101100 11000111 00100001 10110001 01100101 10110001 11001111 11110001 10010100 11111110 00001011 10001111 00111010 01111110 10101000 11111100 1011. Regular fixed-length ASCII of the string: 01110011 01101000 01100101 00100000 01110111 01100001 01110011 00100000 01110100 01101111 01101111 00100000 01110011 01101000 01101111 01110010 01110100 00100000 01110100 01101111 00100000 01110011 01100101 01100101 00100000 01101111 01110110 01100101 01110010 00100000 01110100 01101000 01100101 00100000 01100110 01100101 01101110 01100011 01100101 00101110. At first glance, the Huffman encoding is smaller than the ASCII encoding. In fact, the Huffman encoding has 164 bits while the ASCII encoding has 320 bits, so the Huffman encoding has 48.75% fewer bits than the ASCII encoding. This is explained by the fact that the string has similar frequencies to the string used for the Huffman encoding, as the top characters are: ‘ ‘: 8, ‘e’: 6, ‘o’: 5, ‘t’: 4, ‘s’: 4, ‘h’: 3 for the string and ‘ ‘: 193, ‘e’: 80, ‘t’: 66, ‘a’:61, ‘h’: 61, ‘o’:53 for the Huffman encoding. ‘ ‘, ‘e’, ‘o’, ‘t’ and ‘h’ are all in the top 6 of frequency for both cases, therefore the Huffman encoding is more efficient for the string than the ASCII encoding.

b) Advanced Trees

38782 compares, 4856 Zig-Zigs and 4473 Zig-Zags. As said in question 1 h), the Splay Tree approach is better for a large number of commands, so this approach is the best as compared to AVL or a regular Binary Search Tree approach since the file contained a lot of commands. Interestingly enough the rotateLeft and rotateRight method can be re-used to implement an AVL tree, since similar rotations are executed. Most of the method can also be shaved or tweaked to implement a functional Binary Search Tree, since most method are based on Binary Search Tree approaches.