Homework 8

Question 1 (10 pt.)

Extend the implementation of the Huffman tree presented in class with support for decoding compressed text. Start by creating a new directory named `q1` and copying the baseline implementation of the Huffman tree available in files `Vector.java`, `Heap.java`, `Node.java`, and `HuffmanTree.java`.

a) (6 pt.) Add a new function called `Decode()` in class `HuffmanTree`. This function takes a Huffman-encoded string as an argument, and provides the decoded string as a return value. The string is decoded using the Huffman tree available in the current instance of `HuffmanTree` on top of which function `Decode()` is being invoked. This tree must have been previously built with a call to `BuildTree()` on that instance, as shown in class.

   The `Decode()` function must traverse the Huffman-encoded string, and perform the following actions for each input character:

   - Extract the current character of the input string. This should be a character set to `‘0’` or `‘1’`. If any other character is observed, raise an exception.
   - Keep track of a “current node” in the tree. Initially, the current node is the root. If the current character is `‘0’`, set the current node to its left child. If `‘1’`, set it to its right child.
   - If the current node reaches a leaf, obtain the ASCII character associated with that leaf, and add it as the next character of the decoded string. Then set the current node back to the root.

b) (4 pt.) Create a file named `Test.java` with a main program that takes some input text as a command-line argument. The program should run the following actions:

   - Check that exactly 1 argument was passed from the command line, and raise an exception otherwise. When you run your program, you can pass one single argument that contains multiple words by using double quotes (e.g., `“Huffman tree test”`).
   - Print the input text.
   - Create a Huffman tree, use it to encode the input text, and print the encoded text.

- Calculate the compression ratio as $size_{compressed}$ * 100 / $size_{original}$ and print it. Sizes are given in bits. Notice that each character in the input text counts as 8 bits, while each character in the encoded string counts as 1 bit.

- Decode the string by invoking the `Decode()` function implemented earlier. Print it and verify that it matches the original text.

This is a sample execution for your program:

```
$ java Test “Huffman tree test”

Original text: Huffman tree test
Encoded text: 010001111001001101111010110010111101010000101111000110111
Compression ratio: 41%
Decoded text: Huffman tree test
```

Create a ZIP file named `q1.zip` containing directory `q1` and submit it on Canvas. Your code should compile and run without errors.