CSC3100 Assignment 2

Important Notes:

1. The assignment is an individual project, to be finished on one's own effort.
2. Submissions before the deadline 6pm Nov. 19, 2021 are treated as normal submissions. Submissions within one week after the deadline are treated as late submissions.
3. Plagiarism is strictly forbidden, regardless of the role in the process. Notably, ten consecutive lines of identical codes are treated as plagiarism.

Marking Criterion:

1. The full score of the assignment is 100 marks.
2. Normal submission: 100 marks are given if a submission passes both compression and decompression tests. 80 marks are given if it passes compression test only.
3. Resubmission: For normal submissions that fail in compression test, a resubmission is allowed within one week after the marking. 60 marks are given if the resubmission passes compression test. 75 marks are given if the resubmission passes both compression and decompression tests.
4. Late submission: 60 marks are given if the submission passes compression test. 75 marks are given if it passes both compression and decompression tests. No resubmission is allowed.
5. Zero mark is given if: there is no (normal or late) submission; a normal submission fails compression test and the resubmission fails compression test; a normal submission fails compression test and there is no resubmission; a late submission fails compression test.
6. In case of identical copies, both submissions will be marked as zero. Similar rules apply to other cases of plagiarism found.

Running Environment:

1. The submissions will be evaluated in Java environment under Linux platform.
2. The submission is acceptable, if it runs in any of recent versions of Java SDK environment. These versions include from Java SE 8 to the most recent Java SE 17.
3. The submission is only allowed to import three packages of (java.lang.*; java.util.*; java.io.*) included in Java SDK. No other packages are allowed.
4. In each test, the program is required to finish within 5 second of time, with no more than 128MB memory.
5. Each student is free to test his/her program in the evaluation platform before the submission.

Submission Guidelines:

1. Detailed submission guideline will be given in a separate manual around Nov. 15.
2. Inconsistency with or violation from the guideline leads to marks deduction.

Functional Requirement:

This assignment implements the Huffman tree (https://en.wikipedia.org/wiki/Huffman_coding), which is a powerful algorithm widely in data compression. Two programs are required.

1. *HuffmanCompression.java* compresses a given text file (encoded with ASCII code) using Huffman tree method with the following command:

java HuffmanCompression input.txt dictionary.txt compressed.txt

An example of input file follows:

input.txt (note: there is a linefeed symbol at the end of the following sentence)
SUSIE SAYS IT IS EASY

After constructing a Huffman tree, one file of code dictionary will be output:

dictionary.txt
10:01110 32:00 65:010 69:1111 73:110 83:11 84:0110 85:01111 89:1110

The dictionary shows that the space character will encoded as "01110" (note that 10 is the ASCII code of the linefeed character. 32 is the ASCII code of the space character…)

The other output will be the compressed text:

compressed.txt
1101111111101111100110101110110011001100011011001111010111111001110

The file gives the compressed text of the input.txt based on the dictionary shown above.

2. *HuffmanDecompression.java* decompresses a compressed text based on the Huffman code dictionary with the following command:

java HuffmanDecompression compressed.txt dictionary.txt output.txt

The compressed.txt and dictionary.txt are input files obtained from the compression process. The output file ("output.txt") should have the same contents as "input.txt" used in the compression process.

Program Template:

Each submission is expected to strictly follow the template to implement the required function.

1. Modify getHuffmanCode() and getCompressedCode() in ***HuffmanCompression.java*** to implement the required compression function.
2. Modify main() in ***HuffmanDecompression.java*** to implement the required decompression function.

```java
import java.io.*;
public class HuffmanCompression {
    public static String getCompressedCode(String inputText, String[] huffmanCodes) {
        String compressedCode = null;
        // Your code obtains the compressed code for inputText based on huffmanCodes
        // For example, if inputText="SUSIE SAYS IT IS EASY\n", and '\n' denotes linefeed.
        // , and huffmanCodes for inputText are:
        // Space="00", A="010", T="0110", Linefeed="01110", U="01111", S="11",I="110",Y="1110",E="1111".
        // Then, your program will return the String shown in the following line:
        // "1101111111101111001101011101100110011000110110011110101111001110"
        //  S U    S I  E   SpS A  Y   S   I  T     I  S   E   A  S Y   Lf
        return compressedCode;
    }
    public static String[] getHuffmanCode(String inputText) {
        String[] huffmanCodes = new String[128];
        // Your code would obtain huffmanCodes for inputText
        // huffmanCodes[i]: huffman code of character with ASCII code i
        // if a character does not appear in inputText, then its huffmanCodes = null
        // For example, if inputText="SUSIE SAYS IT IS EASY\n", and '\n' denotes linefeed.
        // A possible huffmanCodes would be:
        // Space="00",A="010",T="0110",Linefeed="01110",U="01111",S="11",I="110",Y="1110",E="1111".
        return huffmanCodes;
    }
    public static void main(String[] args) throws Exception {
        // obtain input text from a text file encoded with ASCII code
        String inputText = new String(java.nio.file.Files.readAllBytes(java.nio.file.Paths.get(args[0])), "US-ASCII");
        // get Huffman codes for each character and write them to a dictionary file
        String[] huffmanCodes = HuffmanCompression.getHuffmanCode(inputText);
        FileWriter fwriter1 = new FileWriter(args[1], false);
        BufferedWriter bwriter1 = new BufferedWriter(fwriter1);
        for (int i = 0; i < huffmanCodes.length; i++)
            if (huffmanCodes[i] != null) {
                bwriter1.write(Integer.toString(i) + ":" + huffmanCodes[i]);
                bwriter1.newLine();
            }
        fwriter1.flush();
        fwriter1.close();
        // get compressed code for input text based on huffman codes of each character
        String compressedCode = HuffmanCompression.getCompressedCode(inputText, huffmanCodes);
        FileWriter fwriter2 = new FileWriter(args[2], false);
        BufferedWriter bwriter2 = new BufferedWriter(fwriter2);
        if (compressedCode != null)
            bwriter2.write(compressedCode);
        fwriter2.flush();
        fwriter2.close();
    }
}
```

```java
public class HuffmanDecompression {

    public static void main(String[] args) throws Exception {
        // args[0] is the file of compressed text
        // args[1] is the dictionary file
        // args[2] is the file of decompressed text to output
        //
        // For example: if your compressed text is:
        // 1101111111101111001101011101100110011000110110011110101111001110
        //
        // , and your dictionary file is:
        // 10:01110
        // 32:00
        // 65:010
        // 69:1111
        // 73:110
        // 83:11
        // 84:0110
        // 85:01111
        // 89:1110
        //
        // Then your expected output would be:
        // SUSIE SAYS IT IS EASY\n
        // where '\n' denotes linefeed character.
        // Note: 10 is the ASCII code of '\n', 32 is ' ', 65 is 'A', ...

        return;
    }
}
```

Appendix

1. HuffmanCompression.java
2. HuffmanDecompression.java
3. input.txt
4. dictionary.txt
5. compressed.txt
6. output.txt