CSE 222 HOMEWORK #7

Huffman Trees

Orhan Aksoy

09104302

1. Problem

A huffman tree encoding/decoding application is to be implemented.

The application is supposed to

- Use a text file to generate character frequency table to use in encoding/decoding
- Use a default 'english' frequency table to use in encoding/decoding
- Encode/decode text files using the huffman algorithm.

2. Analysis

When building a tree from a text file, the operation is trivial: Read each character from the file (it does not matter whatever character is read, because any character can be stored in the huffman tree), calculate their relative frequencies, create the huffman tree and use it in decoding/encoding.

However, in this application, the algorithm is supposed to be evaluated using two frequency tables: One table derived using the input text file, and the other being a standard 'english' huffman tree. The standard huffman trees have only the small letter alphabet (from 'a' to 'z') plus the space character. So, in order to compare the trees, a file with only the allowed characters is to be used for (a) building a frequency table for turkish tree generation and (b) encode/decode using both the standard english tree and the generated one.

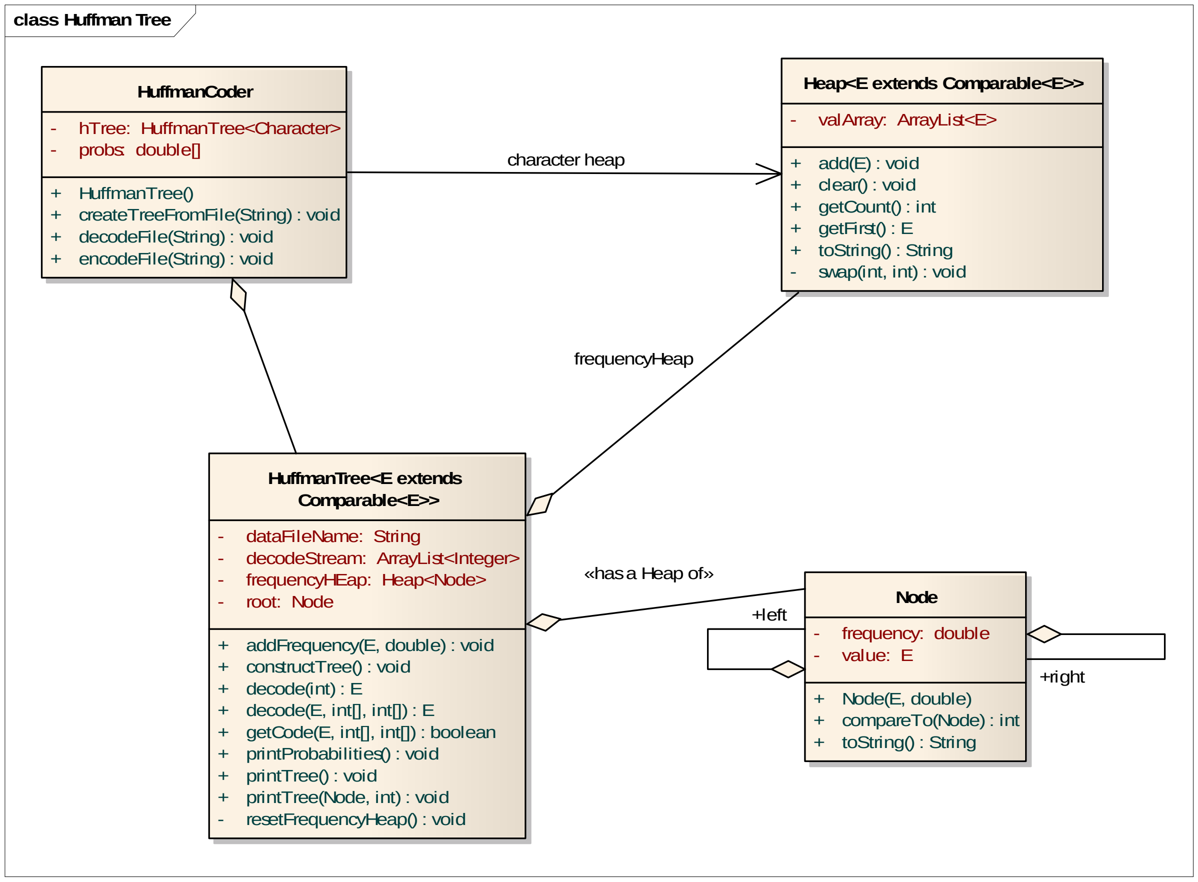
In order to create a Huffman tree using a text file , first, the characters are read one by one and put into a Heap structure. Afterwards, the items are read from the heap one by one from the root. This way, the same items always follow each other, allowing the application calculate the character counts easily. This is the mechanism used to evaluate the frequency table.

Then, the character-frequency pairs can be parsed from the least frequent ones to the more to generate the Huffman tree.

To encode and decode the file, this generated tree or a default one can be used.

3. Design

The class diagram of the solution is shown below:



The HuffmanCoder class instantiates the HuffmanTree class with the type ‘Character’. The basic functions of this class is to

a. During tree creation, read characters from the input file for frequency calculations and feeding them into the HuffmanTree class.
b. During encoding, read encoded bit strings from HuffmanTree class and merge them
c. During decoding, feed bits one by one into the HuffmanTree and receive decoded characters.

The HuffmanTree class is the actual class representing the Huffman tree. It packs the frequency-value pairs in nodes that are stored in a Heap. It also has a method called ‘createTree’ that parses the frequency heap and builds the tree by making the left-right connections of the Nodes that are stored in the heap.

The Heap class represents the Heap data structure. It stores the items in an array and reaches/inserts/deletes using the standard algorithms.

The details of the class member functions can be found on javadoc files.

4. Implementation

4.1. Tree Creation

The tree creation process is started by the method ‘Main’. Once it sees a filename with an argument that is passed by ‘-u’, it calls the ‘createTreeFromFile’ method of the HuffmanCOder class. This class, in turn, reads characters one by one from the file (regardless of whatever character they are), and puts them into a heap.

Once all the characters are put into the heap, they are popped back from the heap one by one. Since they were stored in a heap, they are received in increasing order, with the same items following each other. This mechanism facilitates the calculation of the character counts in the file. At the end, a table is created with all the characters in the input file with their relative frequencies.

These frequencies are fed into the HuffmanTree class one by one. During this operation, the HuffmanTree class simply pushes the frequency-character pairs into a heap. These pairs are packed in Nodes that have a left and right Node references to be used in building a tree in the future.

Once the frequencies are all fed into the HuffmanTree class, the HuffmanCoder class calls the ‘constructTree’ method of the HuffmanTree class. This method implements the standard HuffmanTree building algorithm, where

a. It finds the least frequent two characters, binds them with a common parent and removes them from the heap.
b. It assigns the frequency of the parent as the sum of two
c. It puts the parent back into the heap
d. It repeats steps 1 to 3 until only one item is left in the heap.
e. This last item is the root of the Huffman tree.

4.2. Encoding

When the application initializes, it checks whether a new file is to be used to build the Huffman tree. If not, it uses a standard probability table (standard english) to build the tree. Once the steps in 4.1 are done, the encoding/decoding process is possible.

In order to encode a file

a. Characters are read one by one by the HuffmanCoder class
b. Once a character is read, it calls the 'getCall' method of the Huffman Tree class
c. This 'getCode' method is a wrapper around a recursive getCode method in the HuffmanTree class. This method returns two values:
   a. An integer value representing the coded bitstream
   b. Length of the bitstream

   Note that the integer value in (a) is not sufficient to represent the code, since, in a Huffman tree, the codes (0010) and (10) are different, but their integer representations are both 2.
d. The returned bitstream is appended to the current bitstream. Once they accumulate above 8 bits, a new byte is stored in an ArrayList. This continues until all the characters are encoded.
e. Finally, before storing this bitstream into the coded file, the total number of characters is written at the beginning of the file. Then the byte array explained in (d) are appended in this file.
   This total character count is necessary, because the last character may not be able to fill the final byte. In order to let the decoder know that the bits following the final character should be ignored.

4.3. Decoding

Decoding a coded file is accomplished by traversing the Huffman Tree using the input bit stream from an input file.

In order to decode a file

a. Bytes are read from the encoded file.

b. Once a byte is read, its bits are fed into the HuffmanTree using the method ‘decode’. This method is a wrapper around a recursive decode method, and accepts only ones and zeroes with their relative order in the stored file. The recursive method, and in turn, the wrapper, returns null as long as the fed bits do not arrive a character. When a character match is found, that character is returned.

5. Testing

In order to test the application, the following files are necessary:

The sources:

- Heap.java, HuffmanCoder.java, HuffmanTree.java, Main.java

A Turkish novel named ‘Simdiki Cocuklar Harika’ whose Turkish characters are replaced with their counterparts in English characters

- Harika.txt

The same novel, but converted into smallcase, cleaned up from characters other than letters and space.

- Harika_comp.txt

Before testing, the sources should be compiled using

```
# javac Main.java
```

5.1. Using the application

The application expects command line parameters to execute:

-e : Means ‘encode’
-d : Means ‘decode’
-f <filename> : Means, ‘use the file <filename” to encode/decode’
-u <filename> : Means, use the file <filename> to build a new Huffman tree for encoding/decoding

5.2. Creating and using a HuffmanTree to encode a file

To use the application to both generate the Huffman tree and use it to encode a file:

```
# java Main –e –u Harika.txt –f Harika.txt
```

The file Harika.txt is 246Kbytes and the coded Harika.txt.huff file is 141 Kbytes.

To decode the coded file:

```
# java Main –d –u Harika.txt –f Harika.txt.huff
```

The generated file after decoding is named 'Harika.txt.huff.decoded' and is exactly same as Harika.txt.

5.3. Using standard English character distribution in a HuffmanTree to encode a file and comparing it with the Turkish distribution .

To use the application with the standard English Huffman tree and encode a file:

```
#java Main –e –f Harika_comp.txt
```

The file Harika_comp.txt is 228 Kbytes and the coded Harika_comp.txt.huff file is 126 Kbytes.

To use the same file to get the character probabilities, and encode the file:

```
#java Main –e –u Harika_comp.txt –f Harika_comp.txt
```

The coded Harika_comp.txt.huff file is 117 Kbytes.

Using the English Huffman Tree, the compression ratio is 126/228 = 55.3%
Using the Turkish Huffman Tree, the compression ratio is 117/228 = 51.3%