Algorithms

Huffman Compression & Decompression

Marwan ElSafty	4690
Asmaa Lotfy	4555
Karim Elhamy	4578

1. Problem Statement

Huffman's algorithm takes a very simple idea and finds an elegant way to implement it. At its heart is the observation that the more a thing is mentioned, the shorter its name should be. This idea manifests itself in daily life too. For example, we use nicknames for people we call often, we have abbreviations for long words we often have to use and even the word ok was once an abbreviation for "all correct".

Translated to the world of computers, it means that if a chunk of data repeats itself more than another chunk of data in the same group, it is best to encode the more occurring chunk with smaller code word. Practically what it means is that the compressed file has two parts:

- Dictionary of code words and their corresponding chunks of data.
- The data itself, encoded using codewords

i.e. often called *payload* in compression jargon.

During decompression, the dictionary is read and the codewords are sequentially translated into actual data.

For Example:

BANANA

The ASCII codes for the alphabets in binary are:

B : 01000010 A : 01000001 N : 01001110

Thus the ASCII encoding of BANANA is:

01000010 01000001 01001110 01000001 01001110 01000001

Let’s try to compress this with the aforementioned scheme of calling more frequent chunks of data with smaller nicknames. We will start with a naive interpretation.

A : Occurs thrice. Let’s call it a 0 N : Occurs twice. Let’s call it a 1 B : Occurs once. Let’s call it a 10

The compressed data thus turns out to be:

1001010

2. Pseudocode

2.1. Compression

```
// C is the set of n
Procedure Huffman(C):
    characters and related information
    n = C.size
    Q = priority_queue()
    for i = 1 to n
        n = node(C[i])
        Q.push(n)
     end for
     while Q.size() is not equal to 1
         Z = new node()
         Z.left = x = Q.pop
         Z.right = y = Q.pop
         Z.frequency = x.frequency + y.frequency
         Q.push(Z)
      end while
Return Q
```

2.2. **Decompression**

```
// root represents the root of Huffman Tree
Procedure HuffmanDecompression(root, S):

// S refers to bit-stream to be decompressed
n := S.length
for i := 1 to n
    current = root
    while current.left != NULL and current.right != NULL
            if S[i] is equal to '0'
               current := current.left
            else
               current := current.right
            endif
     i := i+1
    endwhile
    print current.symbol
endfor
```

4. Complexities

Characters frequency: O(n)

Huffman Tree Construction: O(nlgn)

Compression: O(nm)

Decompression: O(k)

Where:

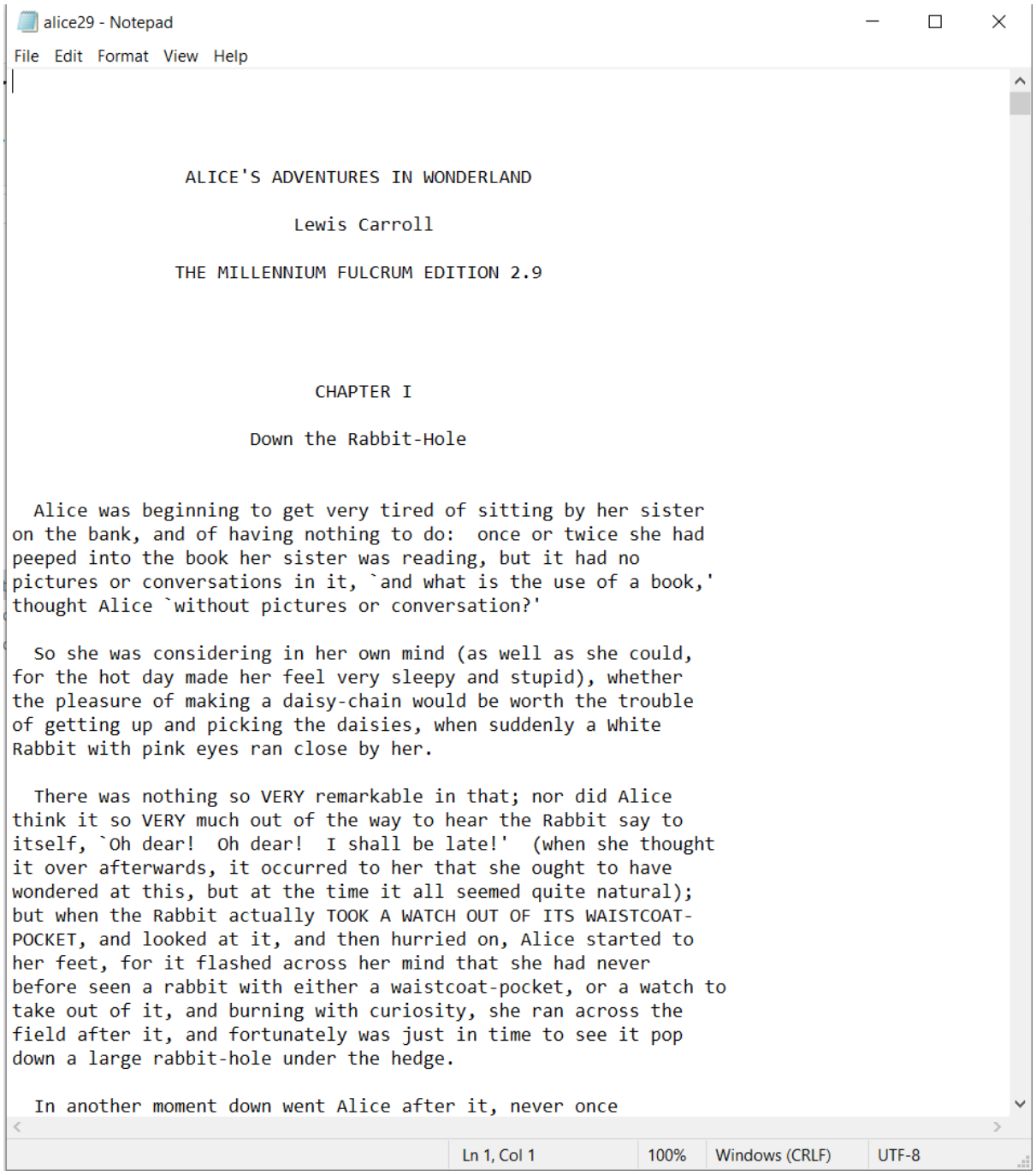
- n is the number of characters in the original text file
- m is the length of the longest code generated
- k is the size of the file in bytes

4. Data Structures

- HashMap were used 3 times:
    1. When computing the frequency each character is stored with its corresponding frequency
    2. When generating the Huffman code each character is stored with its corresponding code
    3. When decompressing the file, the original character-code map is reversed to ease the decoding operation
- Tree is used to construct the Huffman tree where each character found in the original text file resides in a leaf.
- Priority queue (minimum heap) is the used in the operation of Huffman tree construction
- Arrays and ArrayLists of bytes used in encoding and decoding

4. Usage & Sample Runs

4.1. Test Case



4.2. **Compression**

4.3. **Decompression**