Leonard Blam 900086

Homework DS2 #4

1.

1.	(A,31),(B,30),(C,13),(D,10),(E,9),(F,7)	// remove smallest and combine
2.	(A,31),(B,30),(C,13),(D,10), (EF, 16)	// E & F
3.	(A,31),(B,30),(CD, 23), (EF, 16)	// C & D
4.	(A,31),(B,30),(CDEF, 39)	// CD & EF
5.	(AB, 61),(CDEF, 39)	// A & B
6.	(ABCDEF, 100)	// AB & CDEF

The compression rate is 8, since there is 6 letters and each letter is encoded using 8 bits in a constant length encoding

2.

So, we're trying to prove that any optimal prefix code (a code that minimizes the average number of bits per symbol and where no code is the prefix of any other code) must be represented by a full tree (a tree where all nodes have either 0 or 2 children). Here's how we can do that:

First, let's show that any optimal prefix code must be a prefix code. Suppose there's an optimal prefix code that's not a prefix code. Then, there must be two codes, C1 and C2, such that C1 is a prefix of C2. This means that any string encoded using C2 must also be a valid string encoded using C1. But that goes against the definition of a prefix code, where no code is the prefix of any other code. So, any optimal prefix code must be a prefix code.

Now, let's show that any prefix code can be represented by a full binary tree. Suppose we have a prefix code with n symbols, where the ith symbol has a code of length li. We can construct a full binary tree as follows:

Create a leaf node for each symbol and add them to the tree.

For each node in the tree, starting from the leaves and working upwards, assign a 0 or 1 to the edge between the node and its parent based on the corresponding bit in the code for the symbol represented by the node.

It's easy to see that this construction results in a full binary tree that represents the given prefix code.

Finally, let's show that any optimal prefix code must be represented by a full tree. Suppose there's an optimal prefix code that's not represented by a full tree. Then, there must be an internal node in the tree that has only one child. This means that the code for the symbol represented by the child must be a prefix of the code for the symbol represented by the parent. But that goes against the

definition of a prefix code, where no code is the prefix of any other code. So, any optimal prefix code must be represented by a full tree.

3a.

If n is a power of 2, the Huffman tree will be a full binary tree, meaning that all the internal nodes will have exactly two children. The number of leaf nodes will be equal to n, and the tree will have a height of log2(n)

3b.

If n is not a power of 2, the Huffman tree will still be a full binary tree, but it will not be perfectly balanced. This means that the height of the tree will be less than log2(n) and the tree will not have the maximum number of leaf nodes.

4.

To prove that in the corresponding Huffman tree, all leaves are on the same level or in two consecutive levels, we will consider two cases:

Case 1: n is even.

In this case, the Huffman tree will have n/2 internal nodes and n leaf nodes. The leaf nodes will be at the bottom level of the tree, and the internal nodes will be at the levels above them. Since the frequencies are decreasing, the internal nodes will be arranged in such a way that the internal node with the highest frequency is at the root of the tree, and the internal node with the second highest frequency is one of its children. The internal nodes with the next highest frequencies will be the children of this internal node, and so on.

Since the frequencies are decreasing, it follows that the internal node with the highest frequency will be the parent of the internal node with the second highest frequency, and so on. This means that the leaf nodes will be at the same level or in two consecutive levels.

Case 2: n is odd.

In this case, the Huffman tree will have (n-1)/2 internal nodes and n leaf nodes. The leaf nodes will be at the bottom level of the tree, and the internal nodes will be at the levels above them. The internal nodes will be arranged in the same way as in Case 1, with the internal node with the highest frequency at the root of the tree and the internal nodes with the next highest frequencies as its children.

The internal node with the lowest frequency will be a child of one of the other internal nodes, and the leaf node corresponding to this internal node will be at the same level as the other leaf nodes. This means that the leaf nodes will be at the same level or in two consecutive levels.

Therefore, in both cases, all leaves are on the same level or in two consecutive levels.

5a.

It is not possible to construct a valid encoding tree because a prefix is a type of ode in which no code is the prefix of any other code. Therefore, the encoding tree would be invalid.

b.

The tree would look like a full binary tree with the root node representing the symbol with the highest freq. and the leaf nodes representing the symbols with the lower frequencies.

```
Ex on a n = 5 tree                    f1 (0)
                                     /      \
                                f2 (10)      f3(11)
                                /     \
                          f4(100)   f5 (110)
```

c.

```
ex on n=4          f1 (0)
                  /      \
              f2 (10)    f3 (11)
                            \
                           f4 (111)
```

Trie Question:

1.

```
T        F        W        B        P
H        A        O        R        R
I        M        R        E        O
S        O        D        A        B
E        U                 K        L
V        S                          E
E                                   M
N
```

2.

Yes, we can add a count field to each node. The count field will store the number of time corresponding string appears.

Ex. Car, car, cat, cat, dog

```
      C       D
    A         O
  R   T     G
```

Here, cat and car will have a count value of 2, while dog only 1

3.

Lets assume we have a dictionary with all legal words (called dic). We can insert each letter into a trie tree as a child of the previous letter and also starting as a new tree. As we progress we should get the n number of trees where n is the same amount of letters in the string.

For each progress we will check if the word in the tree will result in a word, if we then have a valid word and we want to insert the next letter and this results to a invalid word, we will stop from inserting letters to the particular tree and continue to the next tree.

As we are finishing up, each tree should only have valid words. If not, we easily delete the tree.

```
Sort(Tree* T, char  cur, char  prev):

If prev == null:            // only prev can be null if it is beginning of string
    T <- new Node(cur)
    Return T                // we just need to insert first as the first node in the tree
for all strings in getString(prev) as temp // lets assume this function exist and returns all the string containing prev as the last letter
    newString = temp + cur
    If newString in dic:
        Insert(T, newString)
T <- new Node(cur)              // creates a new first Node with the new value

```

```
main(Tree* T, string s, prev == null):

If T is empty:
    T <- new Tree

cur <- s[0]
If cur == null:                 // return when the string is completely executed
    return;
```

```
        s.pop_front()    // pops the first letter and keeps the rest
        Sort(T, cur, prev)
        prev <- cur
        main(T, s, prev)            // go recursively until the string is over
        for treeString in getString(T):
            If treeString not in dic:
                delete treeString in T
        return T
```