Assignment 2 (Due November 26 2019)

1. Knapsack Problem (60) : Consider a collection of K items out of which n items cannot be subdivided (0-1 knapsack) and m items can be divided into fractions (fractional knapsack), K = n + m. For each item, we are given the weight in pounds and the total cost. Given a maximum weight limit V , design an algorithm that selects the set of elements that will give the maximum value. Note that the problem is a combination of 0-1 and fractional knapsack problem.

(a) Give the pseudocode of your algorithm (10)

```
get input from the items id,value($), weight(lbs) along with the knapsack capacity.
calculate the decimal ratios between weight/value and store it in frac[].
creating a maximum function that will get the maximum value of two items and sort in descending order

int knapSack(int W, int w[], int v[], int n, char type[], int frac[], Item items[]){
    int i, wt;
    int K[n + 1][W + 1]

    if(type[W] == 'W'){ //performing 0/1 knapsack algorithm

    for i = 0 to n{
        for wt = 0 to W{
            if (i == 0 or wt == 0){
                K[i][wt] = 0
            //accounts for the first row and first column of the matrix/aka nothing else to compare to
            }
            else if (w[i - 1] <= wt){
                 K[i][wt] = max(v[i - 1] + K[i - 1][wt - w[i - 1]], K[i -1][wt])
                //get max value of the value above and the value of w[i - 1] plus the value of the previous item
            }
            else{
                K[i][wt] = K[i - 1][wt]
            }
        }
    }
    }else if(type[W] == 'F'){ //performing fractional knapsack algorithm
            //use densities here
            sort(items, items+sizeOfItems, desc); //sorting densities in descending order
            for(i=0; i<n; i++) {
                if(totalWeight + items[i].w<= W) { //if the value is less then the weight capacity
                    totalValue += items[i].v ;
                    totalWeight += items[i].w;
                } else { //then the observed value is added into the knapsack
                    int wt = W-totalWeight;
                    totalValue += (wt * items[i].d);
                    totalWeight += wt;
                    break;
                }
            }
        return totalValue; //returning max value for selected W
    }
   return K[n][W]; //returning the value of the max value at a given weight
}
```

(b) Show how the algorithm works, step by step, for the given example in Table 1. The maximum weight, V, is 10 pounds. W (whole) indicates for the items that cannot be subdivided and F(fractional) stands for items whose fractional value can be taken (10)

There are essentially 2 choices that can occur at each step of the 0/1 knapsack dynamic programming algorithm at each cell.

1. We decide not to use the i-th ID
2. We decide to use the ith ID & reap v[i-1[[w-Wi] rewards; where w is the current column we are evaluating and Wi is the weight of the ith ID

I=ID, the row we are in
w=maxWeight, the column we are in
Vi=the value for the ith element
V[i][w] =
If(Wi <= w){ max(V[i-1][w], V[i-1][w-Wi]+Vi, }
V[i-1][w](otherwise)
If(i=0 | | w=0){ V[i][w]=0}

	Weight Constraint (max weight)										
	0	1	2	3	4	5	6	7	8	9	20
ID 1 VALUE:$50 WEIGHT:3	0	0	0	50	50	50	50	50	50	50	50
ID 2 VALUE:$25 WEIGHT:2	0	0 *adopt from top	25 * V[i-1][w-Wi]+Vi	50 * V[i-1][w],	50 * V[i-1][w],	75 * V[i-1][w-Wi]+Vi	75 * V[i-1][w-Wi]+Vi	75 * V[i-1][w-Wi]+Vi	75 * V[i-1][w-Wi]+Vi	75 * V[i-1][w-Wi]+Vi	75 * V[i-1][w-Wi]+Vi
ID 3 VALUE:$60 WEIGHT:4	0	0 *adopt from top	25 *adopt from top	50 *adopt from top	60 * V[i-1][w-Wi]+Vi	75 * V[i-1][w],	85 * V[i-1][w-Wi]+Vi	110 * V[i-1][w-Wi]+Vi	110 * V[i-1][w-Wi]+Vi	135 * V[i-1][w-Wi]+Vi	135 * V[i-1][w-Wi]+Vi
ID 4 VALUE:$15 WEIGHT:1	0	15 * V[i-1][w-Wi]+Vi	25 * V[i-1][w],	50 * V[i-1][w],	65 * V[i-1][w-Wi]+Vi	75 *max(75,75) =75	90 * V[i-1][w-Wi]+Vi	110 * V[i-1][w],	125 * V[i-1][w-Wi]+Vi	135 * V[i-1][w],	150 * V[i-1][w-Wi]+Vi
ID 5 VALUE:$40 WEIGHT:3	0	15 *adopt from top	25 *adopt from top	50 * V[i-1][w],	65 * V[i-1][w],	75 * V[i-1][w],	90 Max(90,90) =90	110 * V[i-1][w],	125 * V[i-1][w],	135 * V[i-1][w],	150 Max(150, 150)=150
ID 6 VALUE:$30 WEIGHT:2	0	15 *adopt from top	30 * V[i-1][w-Wi]+Vi	50 * V[i-1][w],	65 * V[i-1][w],	80 * V[i-1][w-Wi]+Vi	95 * V[i-1][w-Wi]+Vi	110 * V[i-1][w],	125 110 * V[i-1][w],	140 * V[i-1][w-Wi]+Vi	155 * V[i-1][w-Wi]+Vi

ID 7 VALUE:$50 WEIGHT:3	Density = current highest value/divided by weight =36.6667	Descending sorted →	ID 7 VALUE:$50 WEIGHT:	Add items with the highest ratio to the knapsack until we cant add the next item as whole →		Total value = 166					
ID 8 VALUE:$30 WEIGHT:2	Density = current highest value/divided by weight =15		ID 9 VALUE:$50 WEIGHT:4								
ID 9 VALUE:$50 WEIGHT:4	Density = current highest value/divided by weight = 35		ID 8 VALUE:$30 WEIGHT:2								
ID 10 VALUE:$10 WEIGHT:1	current highest value/divided by weight =15		ID 10 VALUE:$10 WEIGHT:1								

(c) Prove the correctness of your algorithm (10)

```
Enter the capacity of 0/1 knapsack
10

Max value for  0/1 knapsack is 160

Enter the capacity of fractional Knapsack
10

Max value for fractional Knapsack is 160

Enter the capacity of combo Knapsack
10

Max value for  combo Knapsack is 160
Malenas-MBP:a3 malenareyes$
```

(d) Compute the complexity of your algorithm (10)

The algorithm with the worst instance of having all Ws is bounded by a O(WN) runtime where W is the capacity of the knapsack. In the worst case scenario with having all F types the worse run time is O(NlogN). The combination leaves O(WN) as the lower bound, where W represents the weight capacity for the knapsack.

(e) Let the weight of all the items be W and the cost of all the items be C. Let n=m=K/2. Let the limit of the weight be V = K/2W. Prove that in this case, taking any subset of half the items will provide the maximum value (10)

2. Longest Common Subsequence (60) : Write the code for computing the longest common subsequence. Using this code find the LCS between each pair of protein sequence in the file tulp3 relatives.

(a) Submit a clear and well commented code. We will test your code on some other sequences (10 points)

q2.cpp

(b) Find the LCS for each sequence to every other sequence (10 points)

q2.cpp

(c) Based on the length of the LCS to create groups of organisms, where organisms in the same group will have longer LCS. Describe clearly the algorithm/software that you used to compute the groups. Show the groups.(20)

The algorithm used is the UPGMA, as it is a practical algorithm and is a clock-requiring algorithm. The UPGMA will utilize a sequential clustering algorithm where the local topological relationships are identified in order of similarity, and the phylogenetic tree is built in a stepwise order.
initializing the sequences we can observe the two most similar sequences.
We can initially observe these two by assigning the distances between them evenly to the two branches.
This can be filled out in a table two at a time and comparing them in a matrix.
Next we would rewrite the distance matrix replacing those two sequences with their average.
if the two sequences are equal to each other or their is an instance with where there are multiple
ties of averages then we will break ties at random.
we will continue this pattern until all of the sequences have been connected

The following depicts the UPGMA sequence analysis:
We can initially observe a pairwise evolutionary distance by the following matrix

We can first compare sequences A and B and observe the differences between them and apply them in a 2 by 2 matrix. We then compare sequences a and c and complete the graph in the following manner until we fill out the rest of the table.

	A	B	C	D	E
B	2	-	-	-	-
C	4	4	-	-	-
D	6	6	6	-	-
E	6	6	6	4	-
F	8	8	8	8	8

After completion of the table we identify the sequences with the fewest differences between them. In this particular case we can identify that A and B have a seperated distance of 2. We are then to take the first grouping on the tree with the pair of those with the fewest distances(A-B). This branching point is replaced and averaged for a substitution of 1 (2/2=1). We can construct the subtree as depicted.

Next we will determine the average difference between A-B and B C, B D, B E, and B F We can calculate the new distance matrix with the following distance formulas:

distance(A,B), C = (distanceAC + distanceBC)/2 = 4
distance(A,B), D = (distanceAD + distanceBD)/2 = 4
distance(A,B), E = (distanceAE + distanceBE)/2 = 4
distance(A,B), F = (distanceAF + distanceBF)/2 = 4

After these calculations we can substitute the new distances as the average of distances. The following matrix for the second comparison can be depicted as follows.

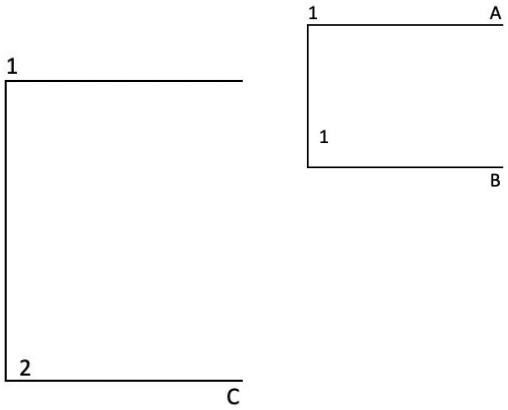
	A-B	C	D	E
C	4	-	-	-
D	6	6	-	-
E	6	6	4	-
F	8	8	8	8

After completion of this table we identify the sequences with the fewest differences between them. In this particular step we can identify that D and E have a seperated distance of 4. We are then to take the second grouping on the tree with the pair of those with the fewest distances(D-E). We can construct the subtree as depicted.

Next we will determine complete the table with D-E grouped together and compare the distances resulting in the following table.

	A-B	C	D-E
C	4	-	-
D-E	6	6	-
F	8	8	8

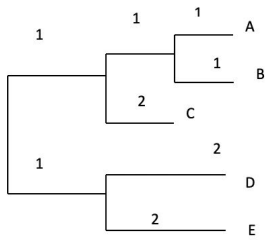
After calculating the average distances we are able to add the phylogenetic tree to produce the branches together as follows.



Repeating these steps for filling out the table with the average distances result in the following table comparing the new groupings of A-B, C.

	A-B, C	D-E
D-E	6	-
F	8	8

Observing the lowest distance as 6 we are able to add the branches to the phylogenetic tree producing the following.



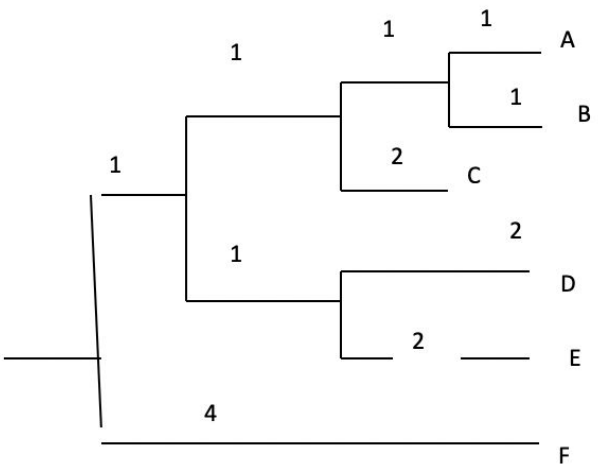
The final table noticing the differences can be seen below depicting an unrooted tree. The mid-point rooting can be applied along all of the branches.

	A-B-C, D-E
F	8

With the mid-pount rooting applied we can root the entire tree positioned with the following formula.

distance(ABCDE),F/2=4

This results in a phylogenetic tree using the UPGMA algorithm and is depicted as follows.



(d) Go to this website ” https://www.ebi.ac.uk/Tools/msa/clustalo/”. Under the space for enter your input sequence cut and paste the entire files. Then press submit. Once the processing is done, click on phylogenetic tree. The tree also groups the organisms. Compare this grouping with the one you obtained (10).

The sequences were ran using the UPGA algorithm and were relatively similar however it appears that some distance information may have been lost as it is a common issue with the algorithm itself. The decision making process depends on the evolutionary processes and incorporated mutations over a given time. It is anticipated that one may end up with an incorrect topology. In order to account for this is it is best to include edge cases that involve more evolutionary edge cases.

3. SpellChecker (40) : Consider an editor that compares a typed in word say of length n and changes it to the nearest word in the dictionary, which can be of size m. For example, if you mistyped ”floyre”, it can change the word to ”flower” or ”flow” or ”floe” or ”floor”. To judge the best word, the spell checker computes the minimum number of changes between the typed word and the words in its dictionary. The changes can be as follows; (i) inserting a character, (ii) deleting a character or (iii) changing a character. All the change operations have equal weightage.

(a) Describe an algorithm to find the minimum number of changes between two given words. This is a well known problem in dynamic programming, so you can take help of other resources. Just explain the process in your own words (20)

The algorithm would be recursive. We start at then end or right side of the word/string and compare the characters. There are two possibilities for each pair of characters. The first would be that the character pair are the same. If this happens you would simply ignore the the character pair and get the count for the remaining strings. Then you recur the lengths m-1 (m = length of first word/string) and n-1 (n = length of second word/string). The other possibility is the last characters are not the same. If this is the case, you look at the 3 operations on the first string, all operations on the last character on the first string, compute the minimum cost for all 3 operations, and take the lowest of the 3 values. For the insert operation you recur for length m and n-1. For the delete operation you recur for length m-1 and n. For the change operation you recur for length m-1 and n-1.

differences.cpp

(b) Using this method show step by step how to determine to which word ”sdimnas” should be changed. The candidates are (i)dynast, (ii) summer, (iii)dismal, (iv)dimmer, (v)sadden. (3*5=15)

(some recursions are not written, but each difference has 3 paths)

(i) str1 : sdimnas, str2: dynast

Compare: s and t. They’re different so compute differences.

-Path 1: Insert t: 1 , dsim**nas**t d**y****nas**t minimum : 4

Compare: s and s. They’re the same so move left.

Compare: a and a. They’re the same so move left.

Compare: n and n. They’re the same so move left.

Compare: m and y. They’re different so compute differences

-Path 1.a: Delete m: 2, dsinast dynast

Compare: i and y. They’re different so compute differences.

-Path 1.a.a: Delete i: 3, d<u>s</u>nast d<u>y</u>nast

Compare: s and y. They're different so compute differences.

-Path 1.a.a.a: Change s to y: 4 **dy**nast **dy**nast

Compare: y and y. They're the same so move left.

Compare: d and d. They're the same and at beginning of both strings so add differences (4).

-Path 1.b: Insert y: 2, dsi<u>m</u>**y**nast <u>d</u>**y**nast

Compare: y and y. They're the same so move left.

Compare: m and d. They're different so compute differences.

-Path 1.b.a: Delete m: 3, ds<u>i</u>ynast <u>d</u>ynast

Compare: i and d. They're different so compute differences

-Path 1.b.a.a: Delete i: 4, d<u>s</u>ynast <u>d</u>ynast

Compare: s and d. They're different so compute differences

-Path 1.b.a.a.a: Delete s: 5, **d**ynast **d**ynast

Compare: d and d. They're the same and both at the beginning of string so add differences. (5)

-Path 1.c : Change m to y: 2 ds<u>i</u>ynast <u>d</u>ynast

Compare: i and d. They're different so compute differences:

-Path 1.c.a: Delete i: 3 d<u>s</u>ynast <u>d</u>ynast

Compare: s and d. They're different so compute differences:

-Path 1.c.a.a: Delete s: 4 **d**ynast **d**ynast

Compare: d and d. They're the same and botha at beginning of string so add differences. (4)

-Path 2: Change s to t : 1, sdimn<u>a</u>**t** dyna<u>s</u>**t** minimum : 5

Compare: a and s. They're different so compute differences:

-Path 2.a: Insert s : 2, sdi<u>m</u>**nas**t d<u>y</u>**nas**t

Compare: a and a. They're the same so move left.

Compare: n and n. They're the same so move left.

Compare: m and y. They're different so compute differences:

-Path 2.a.a: Delete m : 3, sd<u>i</u>nast d<u>y</u>nast

Compare: i and y. They're different so compute differences:

-Path 2.a.a.a: Change i to y: 4, <u>s</u>**dy**nast **_dy**nast

Compare d and d. They're the same so move left.

There's an s in the 1st string but the 2nd string is finished so delete s

-Path 2.a.a.a: Delete s: 5, dynast dynast

Words are the same and at beginning so add differences. (5)

-Path 3: Delete s: 1, sdimn<u>a</u> dynas<u>t</u> minimum: 6

Compare a and t. Different compute differences:

-Path 3.a: Insert t: 2, sdimn<u>a</u>t dynas<u>t</u>

Compare a and s. Different compute differences:

-Path 3.a.a: Insert s: 3, sdi<u>m</u>**nas**t d<u>y</u>**nas**t

Compare a and s. Same move left.

Compare n and n. Same move left.

Compare m and y. Different compute differences:

-Path 3.a.a.a: Change m to y: 4, sd<u>i</u>ynast <u>d</u>ynast

Compare i and d. Different compute differences:

-Path 3.a.a.a.a: Delete i: 5, s**d**ynast **d**ynast

Compare d and d. Same move left

S is left in 1st string so delete.

-Path 3.a.a.a.a.a : Delete s: 6, dynast dynast

Compare and same and beginning of words so add differences. (6)

Finally take the lowest of the 3 initial paths (4,5,6) which is 4.

```
zcb0015@cse05:~/cs4110/A2/4110assignment2/part3$ ./a.out
Number of differences for each word candidate:
dynast : 4
summer : 5
dismal : 4
dimmer : 4
sadden : 5

The suggested word is:
dynast : 4
dismal : 4
dimmer : 4
zcb0015@cse05:~/cs4110/A2/4110assignment2/part3$
```

(c) Suggest two other criteria to select the most appropriate word, excluding the minimum number of changes. Gives reasons for why you selected the criteria (5)

One criteria to select the most appropriate word would be to add weight to how a character changes for when words have the same number of changes. This might be changing a character has a weight of 1, while adding or deleting characters has a weight of 2 or 1.5. Another criteria might be to save words that were previously selected to be transformed along with the frequency at which they were selected. Then, when multiple words have the same number of changes, it would check which word has the highest frequency of being selected as the suggested word.

4. Huffman Coding (40) : The Latin alphabet is used in many European languages such as English, Spanish, French, German, etc. Show that the Huffman coding depends on the language used as well as the content. To do so apply Huffman coding on texts from different languages and different subjects, as follows;

Compress, using Huffman coding, the first 2000 words from Chapter 1 of the following novels in their original language; (Moby Dick; Don Quixote, Three Musketeers). (10)

Translate the exact 2000 words into English, Spanish and French (whichever of the two is not the original language), using an online translator. Compress these using Huffman coding as well (10)

Create a histogram comparing of the length of the code per letter for (i) the three novels in the same languages and (ii) the same novel in different languages. Discuss your results (10+10)

Part 4

(A) Huffman coding in original languages

```
zcb0015@cse05:~/cs4110/A2/4110assignment2/part4$ ./a.out
Original Languages Codes:
Three Musketeers in French:
t: 000
v: 00100
z: 0010100
j: 0010101
h: 001011
f: 001100
y: 00110100
x: 00110101
b: 0011011
p: 00111
l: 0100
o: 0101
g: 011000
q: 011001
c: 01101
r: 0111
m: 10000
d: 10001
i: 1001
e: 101
u: 1100
s: 1101
n: 1110
a: 1111
Moby Dick in English:
f: 00000
g: 00001
y: 00010
c: 00011
t: 001
m: 01000
u: 01001
r: 0101
h: 0110
n: 0111
e: 100
d: 10100
b: 101010
p: 101011
i: 1011
s: 1100
o: 1101
a: 1110
w: 111100
z: 1111010000
q: 1111010001
j: 1111010010
x: 1111010011
k: 11110101
v: 1111011
l: 11111
```

```
Don Quixote in Spanish:
o: 000
b: 00100
h: 001010
f: 0010110
v: 0010111
u: 0011
d: 0100
l: 0101
m: 01100
y: 011010
q: 011011
r: 0111
a: 100
e: 101
n: 1100
s: 1101
t: 11100
c: 11101
p: 111100
z: 11110100
x: 111101010
j: 111101011
g: 1111011
i: 11111
```

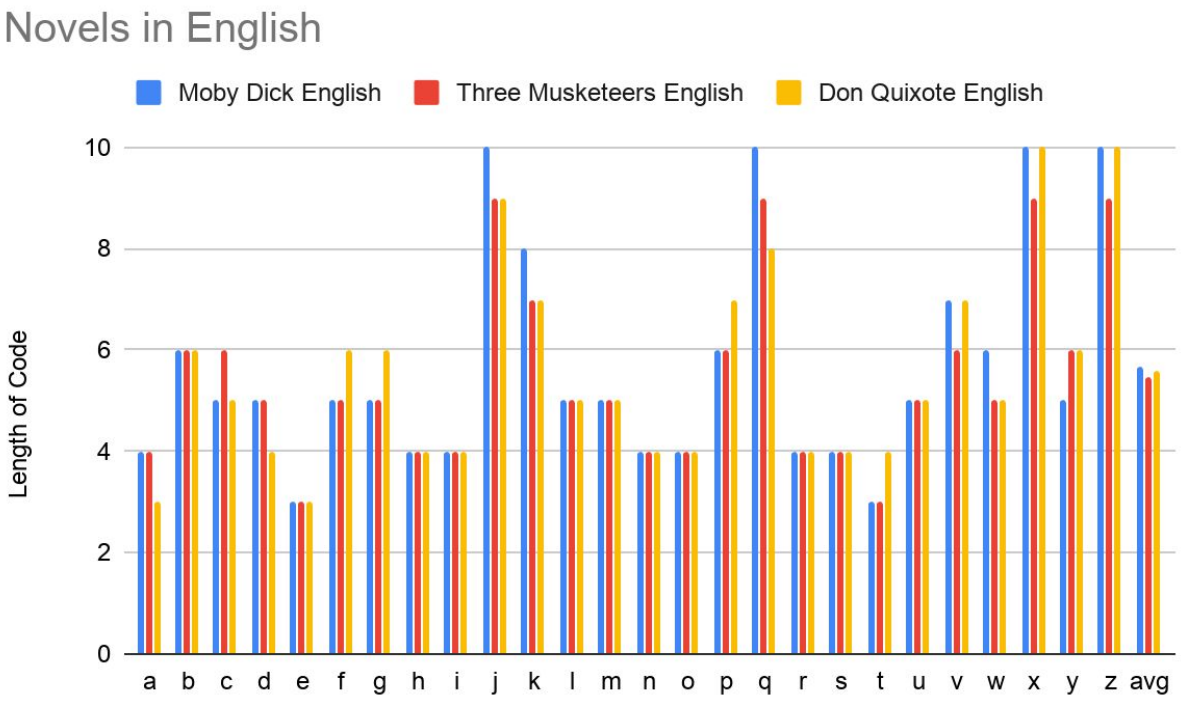
(B) Huffman coding on translated novels

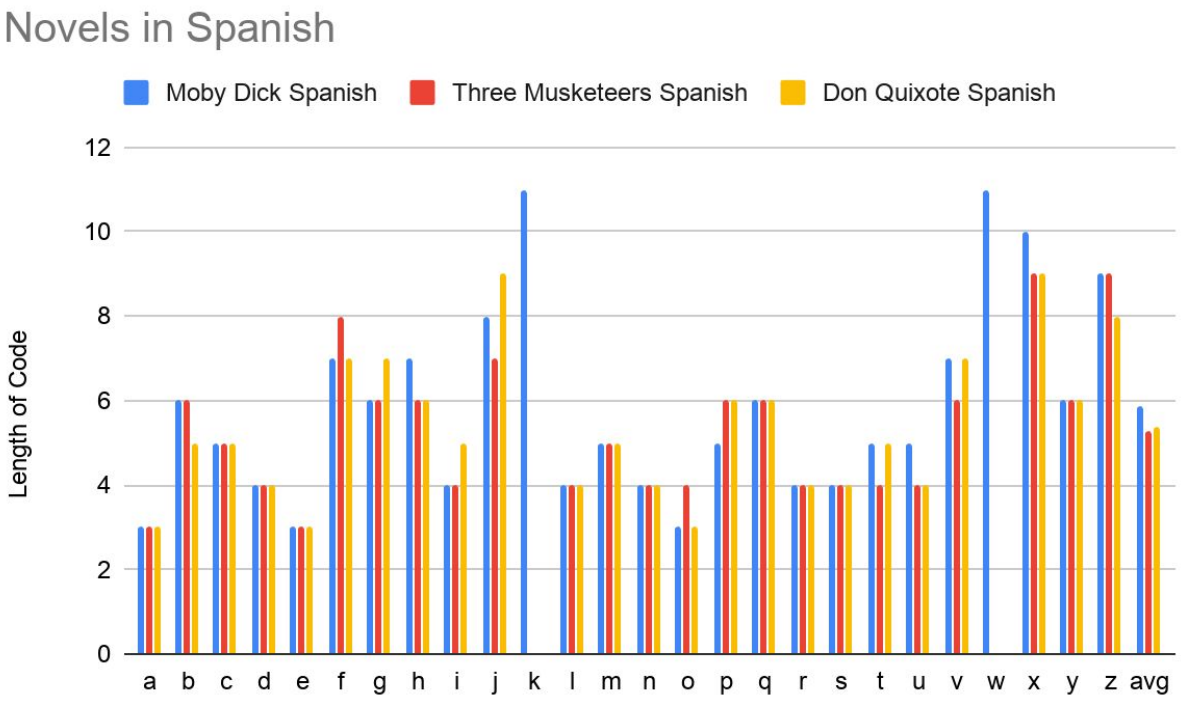
```
Translated into Different Languages:
Three Musketeers in English:
f: 00000
v: 000010
z: 000011000
j: 000011001
x: 000011010
q: 000011011
k: 0000111
g: 00010
m: 00011
t: 001
w: 01000
b: 010010
p: 010011
r: 0101
e: 011
u: 10000
l: 10001
s: 1001
i: 1010
h: 1011
n: 1100
o: 1101
y: 111000
c: 111001
d: 11101
a: 1111
Three Musketeers in Spanish:
v: 000000
h: 000001
q: 000010
y: 000011
t: 0001
i: 0010
u: 0011
d: 0100
l: 0101
a: 011
m: 10000
g: 100010
b: 100011
s: 1001
e: 101
r: 1100
n: 1101
j: 1110000
x: 111000100
z: 111000101
f: 11100011
p: 111001
c: 11101
o: 1111
```

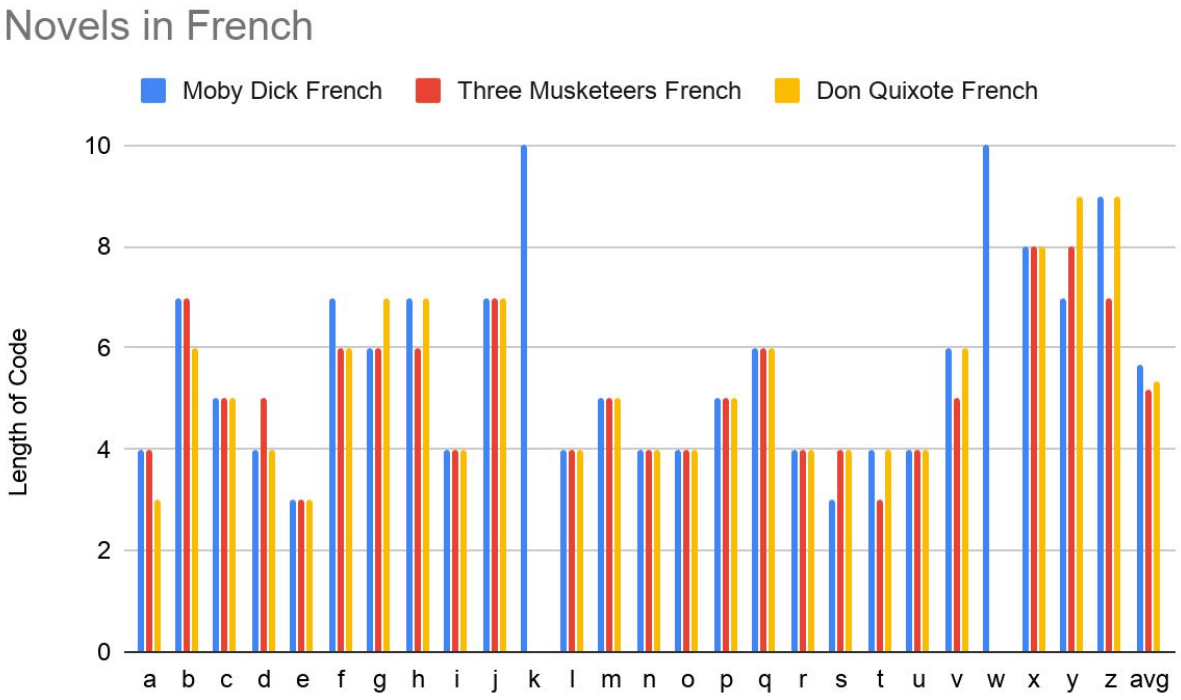
```
Moby Dick in Spanish:
o: 000
d: 0010
y: 001100
q: 001101
p: 00111
l: 0100
i: 0101
a: 011
g: 100000
b: 100001
m: 10001
r: 1001
e: 101
n: 1100
f: 1101000
h: 1101001
k: 11010100000
w: 11010100001
x: 1101010001
z: 110101001
j: 11010101
v: 1101011
c: 11011
s: 1110
u: 11110
t: 11111
Moby Dick in French:
s: 000
d: 0010
l: 0011
g: 010000
y: 0100010
j: 0100011
c: 01001
o: 0101
u: 0110
q: 011100
w: 0111010000
k: 0111010001
z: 011101001
x: 01110101
h: 0111011
p: 01111
r: 1000
i: 1001
t: 1010
m: 10110
f: 1011100
b: 1011101
v: 101111
n: 1100
a: 1101
e: 111
```

```
Don Quixote in English:
a: 000
c: 00100
u: 00101
w: 00110
q: 00111000
j: 001110010
z: 0011100110
x: 0011100111
v: 0011101
b: 001111
r: 0100
d: 0101
e: 011
s: 1000
m: 10010
y: 100110
k: 1001110
p: 1001111
i: 1010
h: 1011
o: 1100
n: 1101
l: 11100
g: 111010
f: 111011
t: 1111
Don Quixote in French:
d: 0000
b: 000100
j: 0001010
x: 00010110
z: 000101110
y: 000101111
p: 00011
a: 001
o: 0100
u: 0101
f: 011000
q: 011001
m: 01101
r: 0111
l: 1000
c: 10010
h: 1001100
g: 1001101
v: 100111
t: 1010
n: 1011
e: 110
s: 1110
i: 1111
zcb0015@cse05:~/cs4110/A2/4110assignment2/part4$
```

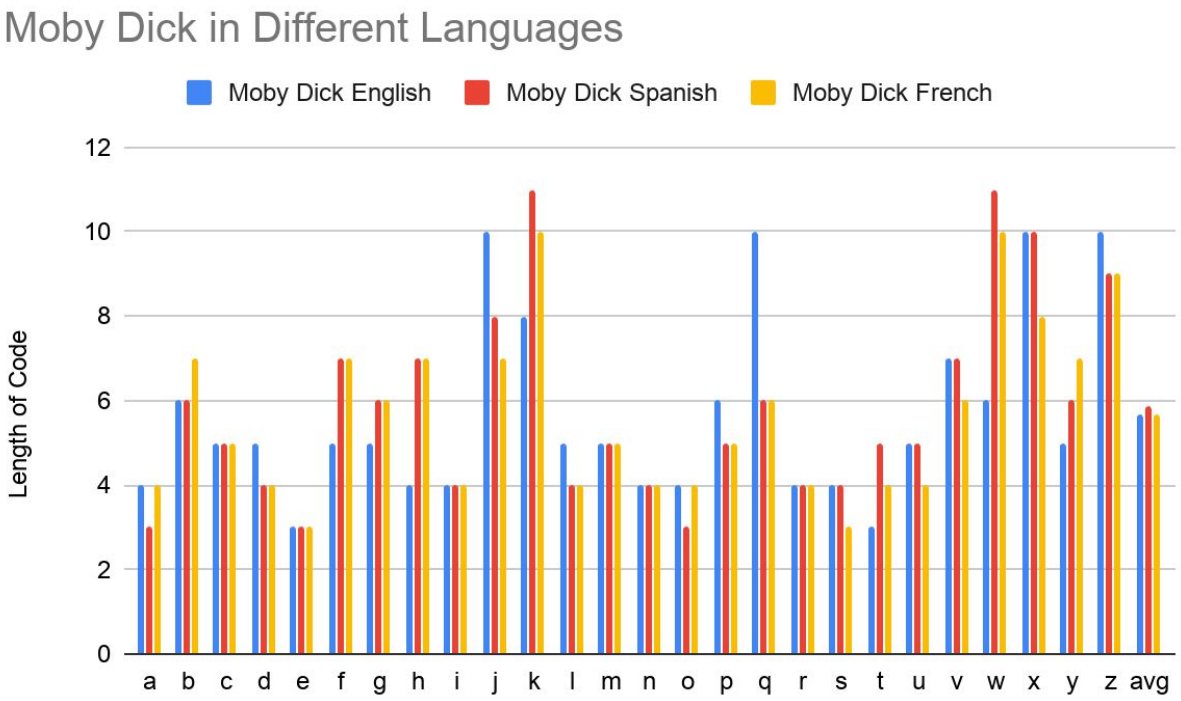
(C)
Length of Code per Letter in Same Language
Part i:

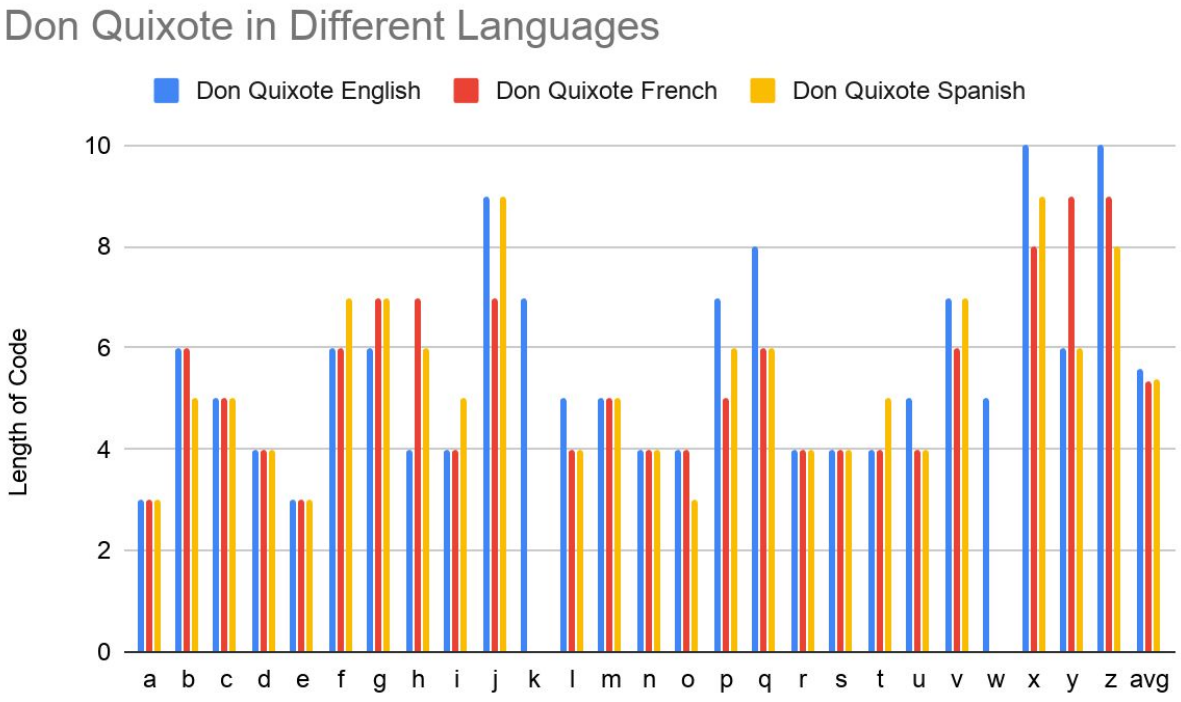


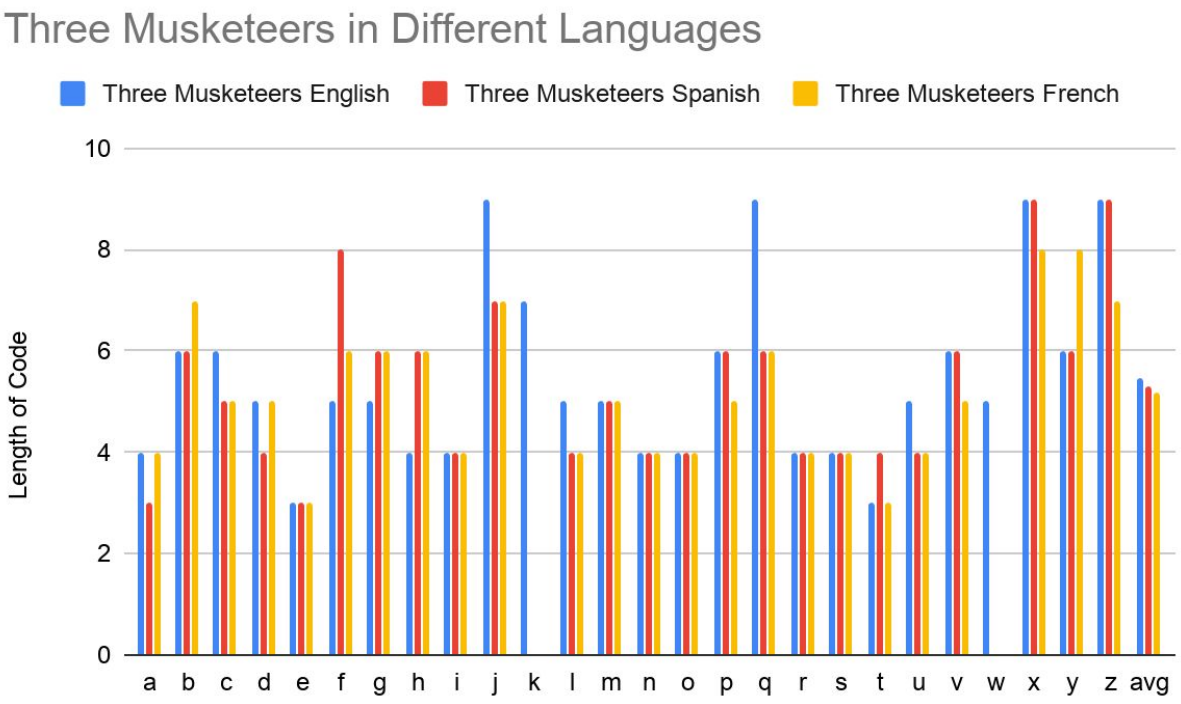



Novels in French
Moby Dick French
Three Musketeers French
Don Quixote French
Length of Code
0
2
4
6
8
10
a b c d e f g h i j k l m n o p q r s t u v w x y z avg

Part ii:


Moby Dick in Different Languages
Moby Dick English
Moby Dick Spanish
Moby Dick French
Length of Code
0
2
4
6
8
10
12
a b c d e f g h i j k l m n o p q r s t u v w x y z avg





From part i, we can see from the histograms that for each language Moby Dick has the highest average code length for characters after using Huffman coding and Don Quixote has the

lowest average for each language. We can also see that the letters ‘k’ and ‘w’ were only found in English and only for Moby Dick in Spanish and French. We can also see that if they were used that they were used the some of the fewest of all the letters since their code lengths were some of the longest. We can also see that vowels, especially ‘a’ and ‘e’ were some of the most used letters in all languages and novels since their code lengths were some of the shortest. Lastly, we can see that the novels had relatively the same length for each character if they are in the same language, showing that a language tends to use certain letters more than others. From part ii, we can see that English has the highest average code length for all novels except Moby Dick which is its original language. We also see that Spanish has the lowest average code length in all novels except Don Quixote which is its original language. We can also see that there are more differences in length for each character in comparing different languages for a novel in part ii than in comparing different novels in the same language in part i.