Midterm Report

Mst Jahan

The purpose of this project is to perform a Caesar Cypher analysis on a given text file. In order to perform this task I first started off my code with a split function to separate my words in the text file. I used the RDD transformation .map and .flat map. Both these transformations apply a function to each element in the RDD. They both will help to return an RDD result and .flatMap will also help to extract words. To separate characters in a text file I apply the same transformations map , .flatMap and .filter(). The filter() transformation takes in a function and returns an RDD that only has elements that pass the filter function. This way it splits all the characters using flatMap() which will apply the function to each element and then return the RDD consisting of only elements that passed the condition of whether or not it was a letter. The next step is to also collect the total number of characters. Meaning the total numbers of character I have in the text file that are similar. For this I used the .map transformation and reduce by key. The reduce by key helps to combine the elements of the RDD together. Together these transformation helps to apply a function which will separate the characters and combine the key components in the text file. Once I have everything separated and combined with the similar elements to how many times they occur I can print it out using a collect function to do this I name my value k and set it to charCount.collect to give me back what happens after all the functions have been applied to it. I can now use the character count to find my most common and most occurring character in a text file. In order to do this we need to set my most occurrence to -1 and most common to a string. I used a foreach transformation which will help apply the function used in the character count to each element of the RDD. Since I had set my function for foreach to K which is my character counts. It’s going to take all the elements collected and have be applied to what is happening within the foreach function .Within the foreach statement I created an if statement which allowed me to set most occurrence to be less then element x.\_2 and I also set most common to equal to the element in x.\_1 which is the second index. This is going to give me back the most common letter we have in the text file. Once I finished that I was able to print the function and it gave me back the letter R which occurs 6366 times in the text. I can now use the letter R and find the difference of that from the letter E since the Letter E is the most common character we can use the difference of our most common character to set how many times we need to shift which is going to be 13 for our function . Now we can do our encrypt/decrypt to fix our text file. The formula for encryption is (x+n)mod26 since there are 26 letter in the alphabet. In order to perform this step I had created a java Caesar Cipher shift first and then converted it over to scala. We need to first traverse the text using a for loop and then use an if statement to encrypt the uppercase characters using the formula and setting n=65. Then have the result append the characters. We would set an if else statement to do the lower case characters but set n=97. We also need to set another if stamen within this method to give back the text once its been appended. Then we can create a driver code to call our text file once its been encrypted. We first set a value to our text file and then use a map transformation to apply the encrypt function to each element. Then we just need print it but we are going to use a foreach to use the function collect in order to print text out in an order. Now that we have our text file decrypted/encrypted we can use it to determine if it has any valid English words. I first created a dictionary file with one-hundred rand English words. Then I used that file to decide if there were similar words in my encrypt file to my dictionary file that were English. We can develop a function that takes in random parts of the text and give us a list of words from the encrypt file and have it go through a for loop which checks each word to see if it is similar to the dictionary words. If it is, it will tell us how many of the words were checked and how many of them are valid English words after comparing them in the for loop.