A5SearchEngine Report

# Function index(*string,pattern,caseSensitive*)

In my search engine project the index function takes in the **string** and **pattern** variables and the *caseSensitive* Boolean as parameters. From this point, the code checks whether the *caseSensitive* Boolean value is false, if this is the case then both the string and pattern variables are changed to lower case using the .*toLowerCase* method, this makes sure that when proceeding to the next step all the characters are of the same case so that the comparison is case insensitive. The function would still be case insensitive if it changed them both to upper case as long as both variables are in the same case. Following any changes due to the *caseSensitive* parameter the index of the start of the pattern in string is returned using the *.indexOf* method, this method returns -1 if the pattern cannot be found.

This function matches the specification and performs as required. When testing the correct output values are achieved when using the example input values.

# Function idxP1(contents,pattern)

The second function takes the parameters **contents**, which is an array containing strings and **pattern** which is a string variable. The function starts by changing **pattern** into lowercase using the *.toLowerCase* method. It then uses a unique type of for loop to search through the **contents** array item by item. Within this for loop a variable, **item** is created this is set to the current item in the array. This allows it to be changed to lowercase this makes both **item** and **pattern** the same case which allows them to be compared without any case sensitivity.

The following if statement returns the index of the start character of where the pattern is found in the array item. It returns the index of which array item if it is found in it else it continues to the next iteration of the for loop to continue searching through the array. If the entire array is searched and the pattern is not found the function returns -1.

# Function match(string,pattern)

The next function takes the string parameters **string** and **pattern**. The variable **found** is defined as an end condition for the following for statement that steps through the **pattern** character by character. The following if statement then checks that the character currently selected in the for statement is a lowercase letter and not any special characters. Within the if statement the found variable is reset to false and a nested for statement that checks through the **string** character by character, checking for a match if a match is found anywhere in **string**, **found** is set to true and subsequently the outer for statement moves on to check the next value in the pattern. However, if found remains false after checking all of the string then the string and pattern are not exact matches and therefore the function returns false. If the function checks all the way through the **pattern** and doesn’t set end any search with a false report then the function can conclude that **pattern** and **string** match and therefore the function returns true. As an improvement I would like to find a shorter way of writing this function.

# Function matchContents(contents,pattern)

The next function takes in the **contents** array and **pattern** string as used previously and it defines an empty array, **returnArray** at the start of the function. It then uses a for loop to step through the **contents** array and uses an if statements to call function 4 *match*if the match function returns true the current **i** value is added to the end of the return array creating a list of all the matching items. Upon the completion of the for loop the **returnArray** is returned from the function.

# Function url1(pages,pattern)

The next function takes the **pages** array and **pattern** string as its parameters, it initializes the **endValue** variable to -1. It then searches through all the array items in **pages** for each array item it creates a variable **item** which is the array item minus the first character this is done using the *.substring* method. A for loop is then used to search the newly created **item** variable using a variety of conditions to check for a symbol that marks the separation between the url and description part of the page, once found the **endValue** is set. Another if statement is then used to check the **item** for a match by using the *index*function with a substring parameter of **item** starting at **endValue** + 1. If the fuction call returns something other than -1 then the function returns the URL part of the **item**. If no match is found over all of the pages then an empty string is returned.

# Function urls(pages,pattern)

The next function is very similar to *url1* and takes the same parameters the only change is the integration of **returnArray** rather than returning a url as soon as a match is found. As an improvement, I would like to modify *url1* so that *urls* can call the *url1* function to avoid code duplication and to shorten my code.

# Function score(string,pattern)

The function *score* takes **string** and **pattern** parameters as used previously, it then defines **count** as a variable and sets it to an integer value of 0. The function then changes both parameters to lowercase to enable case insensitive comparison. A for loop then steps through **string** and checks whether the substring of **string** starting at the index value with a length of the length of **pattern** matches **pattern** using an if statement. If it does **count** is incremented by 1. Upon the completion of the for loop **count** is returned from the function

# Function scores(web,pattern)

The function *scores* takes the parameters **web**, a record array with **url** and **content** fields and **pattern** a string array. It creates a blank **returnArray**, it then steps through all of the items in web using a for loop using the index from this it sets the same index of **returnArray** to the value of the call of the *score* function using the parameters of the **content** field of the current **web** index and the **pattern**. Once the for loop has gone through all the records in **web** the function returns **returnArray**.

# Function urlScores(web,pattern)

*urlScores* takes the same parameters as the previous function it then creates **returnArray** and **item** initialising **item** to 0. The function searches through all the array items in **web** and then calls *scores* on each of these items to first check the score is not 0. If so **returnArray** using **item** as its index is set to a record with a **url** field of the url from the currently indexed **web** record. The **score** field is set to the value of the called *score* function with the parameters being **pattern** and the **content** field from the currently indexed **web** array record. **Item** is then incremented for the next item to be added to **returnArray**

# Function rankedScores(web,pattern)

The function *rankedScores* takes **web** an array of records and sorts this array into descending order through the webpages score. I start by using a do while loop to allow the function to continue sorting items until the sort is complete upon each iteration of this do loop a bubble sort is completed and another item is moved into the right place this is done by an if loop comparing pairs of items and sorting them so that the urls are sorted by there score this is done by calling the *score* function. Temporary variables are used to switch the 2 items if required. Upon completion of the sort the **scoreArray** is returned. Unfortunately I was unable to get my function to sort correctly, this is something I need to improve upon.