

# CSE222 HOMEWORK 4

# REPORT

Süleyman Koramaz

1901042615

# FUNCTIONS

## username.check if valid username

Control first character of the string each call. It is a recursive function; it calls function with same string first character removed until string length is 1. If length is 0, it returns false and print error. It controls every character in the string, they must be letter.

TIME COMPLEXITY:

O(n) because it depends on the length of the string.

## username.contains username spirit

This function calls fillStack to fill the stack. After that, in a loop compare every character of the password with the popped from stack.password1 string must contain at least 1 letter from username.

TIME COMPLEXITY:

O(n), because it travels all elements on the stack and number of the elements in the stack depends on the string.

fillStack() method has O(n) time complexity too so it does not effect all function. It is like O(n + n) = O(n).

## username.fill stack

This function fills the Stack that is variable of the class.

TIME COMPLEXITY:

O(n) because it depends on length of the string.

## password1.is balanced password

Until traveled all string (or balanced time), just control the brackets.

Check bracket is open or close with using indexOf() method on declared open and close strings.

If bracket is open, push it to the stack, if it is close, check top of the stack is it’s complemented.

. If it is, pop it from Stack and continue. İf it is not return false, because it is not balanced. When a bracket opened, same type bracket must close it.

TIME COMPLEXTY:

O(n), because it travels the string with loop, so it depends on length of the string. There is a just one function that may affect time complexity. It is indexOf(). Normally this method has O(n) time complexity, but strings have constant length because we declared them, and they can’t change. So, this function does not affect main function.

## password1.is palindrome

It is recursive function that control string is palindrome of it can relocate to palindrome. Logic of the algorithm is removing letter pairs two by two until the end of the recursion. At the end, if there is a 0 or 1 letter remain it is palindrome or it can relocatable. If it is not, string is not palindrome, or it cannot be relocatable.

First, function check length is 0 or 1 if it is return true. (If string’s length is 1 it is palindrome, if it is not at this point is end of the recursion).

Secondly, remove brackets from string. After that, control the first character of the string if it has pair in remaining. If it has, remove both from string and call function again with remain of the string.

If it has no pair, control second character with same way. And do same process. If second character has no pair either, string cannot be palindrome. Because all letters must have pair and removed with that, exceptionally 1 letter can be left. Not 2!

TIME COMPLEXITY:

O(n^3), because there is a for loop that travels all string and in this for loop deleteChatAt() function which has O(n) time complexity. And out function works recursively and in the worst-case it is O(n) complexity. At the end, out function has O(n\*n\*n) complexity, so O(n^3).

## password1.remove parantheses

Remove brackets from the string for check palindrome situation.

TIME COMPLEXITY:

O(n) because it travels all string with loop. It depends on the length of the string.

## password1.control length

Control length of the password2, it must minimum 8.

TIME COMPLEXITY:

O(1) because there is just 1 statement and it takes constant time.

## password1.control brackets

Control number of the brackets. It must be at least 2.

TIME COMPLEXITY:

O(n) because it travels all string with loop.

## password2.is exact division

Recursive function that controls, password2 is obtainable by adding any coefficient of the denominators.

If password2 == 0, return true because it is our purpose. We are going recursively by decreasing password2.

If denominators array size become 0, return false. Because there is no option remain to get password2 by adding.

If password2 become negative, return false because it is not obtainable by adding.

Recursive logic is, return an OR statement, one side of the OR statement call function with subtract first element of the array from password2. The other side of the OR statement call function with same password2 but remove first element of the array. With this OR statement in the return, function will try all possibilities recursively.

TIME COMPLEXITY:

O(n^2) because recursive call number is n + (n-1) + (n-2) + … 1 which is equal to = n(n+1) / 2. So the simplified time complexity is O(n\*n+1) = O(n\*n) = O(n^2).

## password2.remove first element

Removing first element from array to use in the isExactDivision() function

TIME COMPLEXITY:

O(n), because it copies all elements of the array to new array one by one.

## password2.check siz

Control the size of the password2. It must: 10 < password2 < 10000

TIME COMPLEXITY:

O(1), because there is a only one time statement and it takes constant time.

# OUTPUT RESULTS

Text

Description automatically generated