GTU Department of Computer Engineering

CSE 222/505 - Spring 2023

Homework #4 Report

ÖMER SARIÇAM

200104004009

1)Time Complexity Analysis

Time Complexities of the Methods in security_system Package:

- checkIfValidUsername = O(n) where n is the length of the input string ‘username’. Because the method is recursively checking each character of the input string until it reaches the end of the string or finds a non-letter character.

- isPassword1Valid = O(n) where n is the length of the input string ‘password1’. I use matches method of String class to find that if there is a letter in string and if there is a non-letter and non-letter element. Matches method traverse the string to find the result. In another “else” statement, I traverse the string again to find if there a less than two brackets.

- containsUserNameSpirit = O(n*m) where n is the length of the input ‘username’ and m is the length of the input ‘password1’. Because I have nested loop.

- isBalancedPassword = O(n) where n is the length of the input ‘password1’ . Because first, I traverse the string for push the string elements to the brackets stack. Then I pop elements in that stack to check if there is a situation that breaks the balance.

- inverse = O(1) The inverse method takes a single character as input and returns another character based on a set of predefined rules. It consists of a series of if statements that check the input character against a set of specific characters and return a corresponding character if there is a match. If none of the if conditions are satisfied, the method returns the character 'e'. Since the inverse method performs a constant number of operations regardless of the input character, its time complexity is constant or O(1).

- isPalindromePossible = O(n*n) = O(n^2) where where n is the length of the input ‘password1’. Because the method traverse the string recursively. In each recursion, the indexOf method is called, and time complexity of indexOf method is O(n). Additionally, the substring method is called within the isPalindromePossible1 method can also take O(n) time in the worst case. Therefore, in the worst case, the isPalindromePossible1 method could potentially call the indexOf and substring methods for each character in the input string, resulting in a time complexity of O(n^2). However, in practice, the time complexity will likely be lower than O(n^2) in most cases.

- isExactDivision = O(S*n) = where where n is the length of the input ‘denominations’ array and S is the number of times the recursive method is called.. In the worst case, the recursive function call will be made S times (when the sum of denominations equals the target password2), and each recursive call involves iterating over the entire denominations array of length n.

2)RUNNING COMMAND AND RESULTS

Outside of the security_system package enter these command:

javac */*.java or javac security_system/*.java //to compile

java security_sistem.TestClass1 // to run the test class

```
pentagon@Pentagon:~/Desktop/UNI/2. sene/CSE 222-DATA/homeworks/hw4/hw4.1$ javac */*.java
Note: security_system/Password1.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
pentagon@Pentagon:~/Desktop/UNI/2. sene/CSE 222-DATA/homeworks/hw4/hw4.1$ java security_system.TestClass1



Test 1...
Inputs: username: 'sibelgulmez'  -  password1: '[rac()ecar]'  -  password2: '74'
The username and passwords are valid. The door is opening, please wait..


Test 2...
Inputs: username: ''  -  password1: '[rac()ecar]'  -  password2: '74'
The username is invalid. It should have at least 1 character.


Test 3...
Inputs: username: 'sibel1'  -  password1: '[rac()ecar]'  -  password2: '74'
The username is invalid. It should have letters only.


Test 4...
Inputs: username: 'sibel'  -  password1: 'pass[]'  -  password2: '74'
The password1 is invalid. It should have at least 8 characters.



Test 5...
Inputs: username: 'sibel'  -  password1: 'abcdabcd'  -  password2: '74'
The password1 is invalid. It should have at least 2 brackets.



Test 6...
Inputs: username: 'sibel'  -  password1: '[[[[]]]]'  -  password2: '74'
The username is invalid. It should have letters too.


Test 7...
Inputs: username: 'sibel'  -  password1: '[no](no)'  -  password2: '74'
The password1 is invalid. It should have at least 1 character from the username.


Test 8...
Inputs: username: 'sibel'  -  password1: '[rac()ecar]]'  -  password2: '74'
The password1 is invalid. It should be balanced.


Test 9...
Inputs: username: 'sibel'  -  password1: '[rac()ecars]'  -  password2: '74'
The password1 is invalid. It should be possible to obtain a palindrome from the password1.
```

```
Test 10...
Inputs: username: 'sibel'  -  password1: '[rac()ecar]'  -  password2: '5'
The password2 is invalid. It should be between 10 and 10,000.


Test 11...
Inputs: username: 'sibel'  -  password1: '[rac()ecar]'  -  password2: '35'
The password2 is invalid. It is not compatible with the denominations.


Test 12...
Inputs: username: 'Ömer'  -  password1: '[ra5c()ecar]'  -  password2: '35'
The username is invalid. It must contains only letter and brackets


pentagon@Pentagon:~/Desktop/UNI/2. sene/CSE 222-DATA/homeworks/hw4/hw4.1$
```