HW4-Report

Murat Erbilici

200104004007

Time Complexities of Each Function:

checkIfValidUsername(…): The time complexity is O(n). Because function starts to check first index of the String. And invoking the function till last index.

containsUserNameSpirit(…): The time complexity is O(n*m). Let's say username's length is m, password1's length is n. There is a nested loop containing two while. The First one checks until password1 stack is empty, The Second one checks until username stack is empty.

isBalancedPassword(…): The time complexity is O(n). We push every element of password1 to the stack with for loop and check every element(worst-case) in stack with while loop. That's why the time complexity is O(n).

isPalindromePossible(…): The best case time complexity is O(n/2) which means O(n) because if every letters in start and end are same, it checks only n/2 time since start+1 and end-1 in every recursive call. On the other hand, the worst-case time complexity is quadratic O(n^2) because if any letter doesn't match until replacing all characters this means n time replacing for n/2 characters. That's why the worst-time complexity is O(n^2).

isExactDivision(…): The time complexity is O(n^2). Because each function can call itself two times to check all possibilities.

The Functions I added:

isletter(…), isopen(…), isclose(…), ismatch(…), checkpassword2(…) and isparanthesis(…): These have constant time complexity because there is no loop depending on any length.

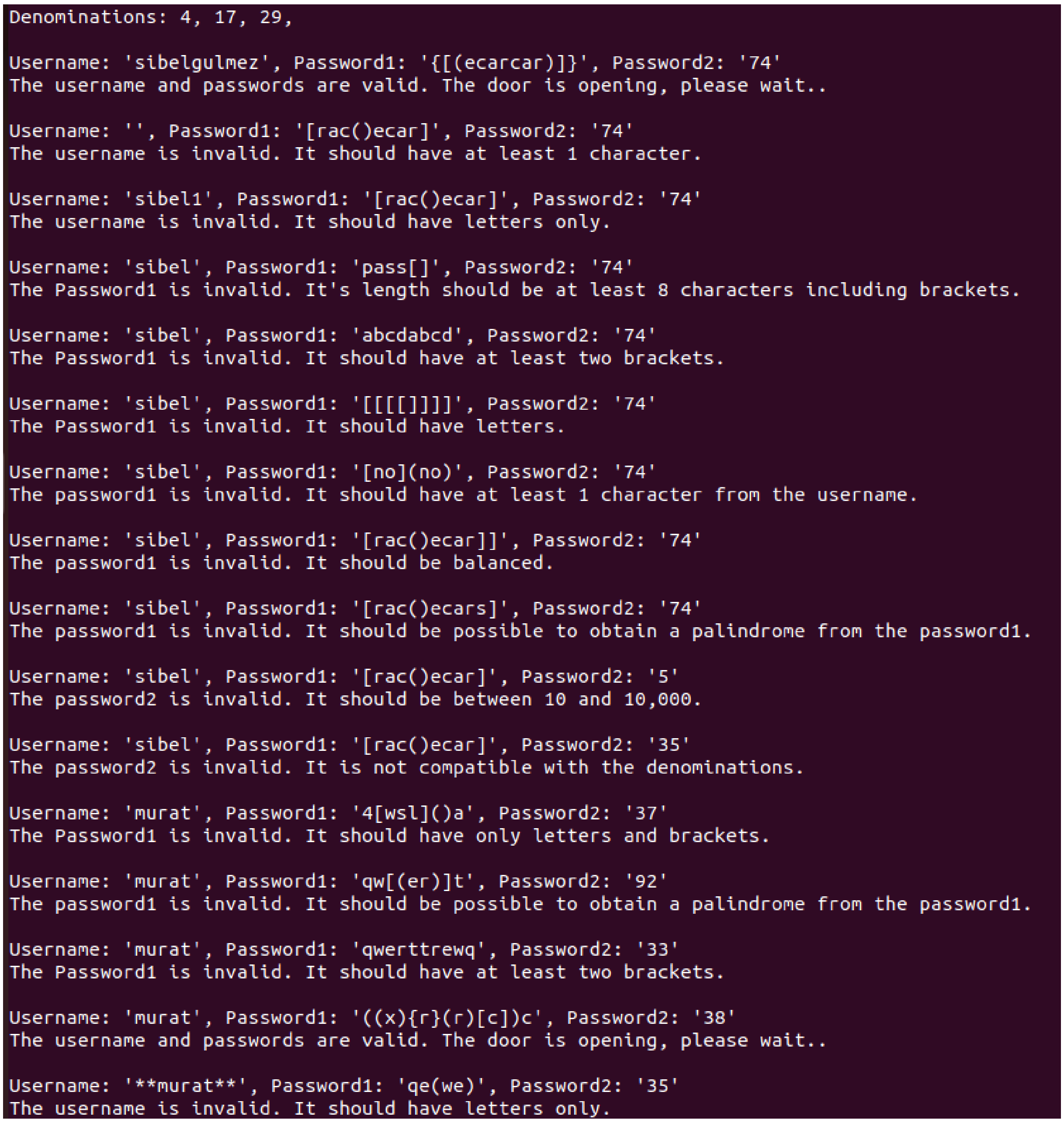
checkpassword1(…): The time complexity is O(n). Because the function checks every element

in the string with one for loop.

onlyletters(…): The time complexity is O(n). Because the function checks every element

in the string with one for loop.

Output Results:



```
Denominations: 4, 17, 29,

Username: 'sibelgulmez', Password1: '{[(ecarcar)]}', Password2: '74'
The username and passwords are valid. The door is opening, please wait..

Username: '', Password1: '[rac()ecar]', Password2: '74'
The username is invalid. It should have at least 1 character.

Username: 'sibel1', Password1: '[rac()ecar]', Password2: '74'
The username is invalid. It should have letters only.

Username: 'sibel', Password1: 'pass[]', Password2: '74'
The Password1 is invalid. It's length should be at least 8 characters including brackets.

Username: 'sibel', Password1: 'abcdabcd', Password2: '74'
The Password1 is invalid. It should have at least two brackets.

Username: 'sibel', Password1: '[[[[]]]]', Password2: '74'
The Password1 is invalid. It should have letters.

Username: 'sibel', Password1: '[no](no)', Password2: '74'
The password1 is invalid. It should have at least 1 character from the username.

Username: 'sibel', Password1: '[rac()ecar]]', Password2: '74'
The password1 is invalid. It should be balanced.

Username: 'sibel', Password1: '[rac()ecars]', Password2: '74'
The password1 is invalid. It should be possible to obtain a palindrome from the password1.

Username: 'sibel', Password1: '[rac()ecar]', Password2: '5'
The password2 is invalid. It should be between 10 and 10,000.

Username: 'sibel', Password1: '[rac()ecar]', Password2: '35'
The password2 is invalid. It is not compatible with the denominations.

Username: 'murat', Password1: '4[wsl]()a', Password2: '37'
The Password1 is invalid. It should have only letters and brackets.

Username: 'murat', Password1: 'qw[(er)]t', Password2: '92'
The password1 is invalid. It should be possible to obtain a palindrome from the password1.

Username: 'murat', Password1: 'qwerttrewq', Password2: '33'
The Password1 is invalid. It should have at least two brackets.

Username: 'murat', Password1: '((x){r}(r)[c])c', Password2: '38'
The username and passwords are valid. The door is opening, please wait..

Username: '**murat**', Password1: 'qe(we)', Password2: '35'
The username is invalid. It should have letters only.
```

I created three classes for three inputs(Username, Password1, Password2). In these classes, I implemented the related functions that check the validity of inputs. These three classes have only functions, doesn't have any fields because inputs are given in as parameters of functions. There is one TestClass which I implemented to test functions according to inputs that are created manually.

Implementing functions 1,4 and 5(from pdf) as recursive makes sense because firstly breaking down into smaller subproblems and solving the entire problem for these cases make my job easier. I used stack for functions 2 and 3. Although using stack in function 3 is sensible because we need to deal with only top of the stack, using stack in function 2 is not efficient. Reason of that, to check every element in stacks requires pushing all element into the stack in every loop because stacks allow us to access only top. In function 3, removing letters and in function 4 removing brackets make my job easier while checking the validity of inputs.