Assignment 1

September 28, 2024

The assignment is due on Friday, **11 October 9:00pm**. Late submissions will be accepted until 14 October, 9:00pm, with a penalty of 10%. After that no submissions will be accepted.

Question 1: (10 points) Prove that if $g(n) \in O(f(n))$ and $h(n) \in O(f(n))$, then for any real constants $a > 0$ and $b > 0$, the function $ag(n) + bh(n) \in O(f(n))$. In other words, prove that $O(f(n))$ is closed under linear combinations.

Question 2: (10 points) Let r be a real number such that $0 < r < 1$. Prove by induction that

$$\sum_{i=0}^{n} r^i = \frac{1 - r^{n+1}}{1 - r}$$

for all $n \geq 0$, where n is an integer.

Question 3: (8 points) Suppose that `algorithm1` runs in $O(n \log n)$ time and that the number of times `algorithm2` performs/calls `algorithm1` is $O(n)$ (and it does nothing else). What can be said about the running time of `algorithm2` in terms of Big-O notation? Justify your answer using the definition of Big-O.

Question 4: (12 points) Help! I wrote some code as part of my job as a TA, but it takes too long to run! Can you explain why it is taking so long? Assume that eClass stores data in two objects:

1. `student_info`: a list of lists, where each sublist represents a student's ID (first element) and name (second element).
2. `grades`: a list of lists, where each sublist represents a student's ID (first element) and grades (second element).

I want to display the grades for the students using their names rather than their IDs. Below is some code I wrote to accomplish this task:

```python
class EClass:
    def __init__(self, student_info, grades) -> None:
        self.student_info = student_info
        self.grades = grades

    def search(self, id):
        for i in range(len(self.student_info)):  # 1 step to increment i
            student = self.student_info[i]  # 1 step
            if student[0] == id:  # 1 step
                return student[1]  # 1 step

    def generate_report(self):
        report = []  # 1 step
        for i in range(len(self.grades)):  # 1 step to increment i
            id = self.grades[i][0]  # 1 step
            grade = self.grades[i][1]  # 1 step
            name = self.search(id)  # ? steps
            report.append([name, grade])  # 1 step
        return report  # 1 step

if __name__ == "__main__":
    student_info = [
        [301, "Charlie"],
        [123, "Alice"],
        [274, "Bob"],
    ]
    grades = [
        [123, 90],
        [301, 70],
        [274, 80],
    ]
    eclass = EClass(student_info, grades)
    report = eclass.generate_report()
    print(report)
```

a. **(3 points)** For the **worst case**, write an expression for a function that counts the **exact** number of operations performed by `EClass.search(id)` in terms of the number of students (the length of `EClass.student_info`), which we will call n.

$$f(n) = \cdots$$

b. **(3 points)** For the **worst case**, write an expression for a function that counts the **exact** number of operations performed by `EClass.generate_report()` in terms of n. Explain how the lines of code contribute to the expression. (Hint: you can use $f(n)$ from part (a) in your expression. Also, assume `Eclass.search(id)` will use the exact same number of steps on each iteration, $f(n)$)

$$g(n) = \cdots$$

c. **(3 points)** Find and justify the worst-case running time of `EClass.generate_report()` in Big-O notation, in simplest terms.

d. **(3 points)** A friend of mine wrote similar code to accomplish the same task. The difference is that in `EClass.generate_report()`, they first sorted `EClass.student_info`, which takes $O(n \log n)$ time, and this allows `EClass.search(id)` to run in $O(\log n)$ time. Show that `EClass.generate_report()` takes fewer steps than my version for sufficiently large n by **stating** the running time of my friend's algorithm in big-Oh, allowing you to compare asymptotic growth rates.

```python
    def generate_report(self):
        report = []  # 1 step
        sort(self.student_info)  # n log n steps
        for i in range(len(self.grades)):  # 1 step to increment i
            id = self.grades[i][0]  # 1 step
            grade = self.grades[i][1]  # 1 step
            name = self.search(id)  # log n steps
            report.append([name, grade])  # 1 step
        return report  # 1 step
```

Coding Questions

1 Email Validator (30 points)

Overview: In today's digital landscape, email communication plays a crucial role in personal and professional exchanges. Validating email addresses ensures that user inputs are formatted correctly and can help prevent errors in registration systems. In this exercise, you will create a program that validates email addresses based on specific criteria.

The input for this program will consist of multiple email addresses in a single line, separated by spaces. The program should first output the validity of each email.

A sample input to the program:

```
mscott@gmail.com jhalpert@domain.ca dschrute@gmail.com $km@gmail.com
pbeesly@spam.ca
```

Expected output:

```
Valid
Valid
Valid
Invalid
Forbidden
```

To illustrate the parts of an email, consider the address `cmput274@ualberta.ca`. The part before the "@" symbol (`cmput274`) is the local part of the email. The part to the right of the "@" symbol (`ualberta.ca`) is the domain, which can be further divided into two components: the second-level domain (`ualberta`), and the top-level domain (TLD), represented by the extension (.ca).

You will write a function called `validate_email(email)` that takes a string input representing an email address and returns whether the email is "Valid", "Invalid", or "Forbidden".

The following criteria should be used to classify the email addresses:

- **Valid:** The local part contains only allowed characters (letters, digits, periods, and dashes). The email must contain exactly one "@" symbol. The domain part must contain a period "." and the characters after the period (TLD) must be from a list of allowed domain suffixes (`.com, .ca, .org, .net, .gov, .edu`).
- **Invalid:** The email address does not satisfy any of the above properties.
- **Forbidden:** The email address is valid but belongs to specific forbidden second-level domains (`@scam, @spam, @fakeemail, @trashmail, @pleasenotspam, @therealtaylorswift, @sendmoney`).

Examples:

Input: `john_doe@gmail.com`
Return: "Invalid" (Reason: Invalid character "_")

Input: `jane@domain.co`
Return: "Invalid" (Reason: Invalid domain suffix ".co")

Input: `jdoe@fakeemail.com`
Return: "Forbidden" (Reason: Forbidden second-level domain)

Input: `cmput274@fakeemails.com`
Return: "Valid"

Marking Rubric: The code must solve the problem algorithmically. If there is any hardcoding (e.g., `if email == "cmput274@ualberta.ca": return "Valid"`) for the provided test cases, zero correctness marks will be given.

This question will be marked out of 10 for correctness:

- 30/30: Pass all 10 of the provided test inputs.
- 21/30: Pass any 9 of the provided test inputs.
- 12/30: Pass any 4 (or more) of the provided test inputs.
- 6/30: Pass any of the provided test inputs.

2 String manipulation and analysis (30 points)

Overview: Strings are essential components in modern programming, acting as the primary means of handling and manipulating textual data. They are widely used in various applications, including user input processing, data representation, and communication between different software systems. Strings allow developers to create dynamic content, such as generating web pages or forming database queries, making them vital for web development and data management. Their flexibility and ease of use make them indispensable in a wide range of programming tasks, highlighting their importance in contemporary software development.

In this question, your task is to demonstrate your ability to manipulate strings and perform data analysis on them. Your program will take one sentence as input and perform 3 tasks as follows:

1. **Sentence reversal:** Reverse the order of words in the input sentence, ensuring that each word retains its original form while the sequence is inverted.
2. **Duplicate removal:** Remove any duplicate words from the reversed list from task 1, ensuring that each word appears only once.
3. **Median calculation:** Analyze the resulting list from task 2 to find the median word length.

Tasks 2 and 3 will build on the output of their preceding tasks, which means that the results of task 1 will be used as the input for task 2, and so on.

Input:

```
Bears, beets, Battlestar Galactica.
```

Expected output:

```
Galactica Battlestar beets Bears
Galactica Battlestar beets Bears
7
```

Input:

```
Me think, why waste time say lot word, when few word do trick?
```

Expected output:

```
trick do word few when word lot say time waste why think Me
trick do word few when lot say time waste why think Me
3
```

2.1 Sentence Reversal

In this task, you will have to reverse the input sentence. To make the subsequent parts easier, you should remove all occurrences of a few characters from the input string. Write 2 functions, named `clean_input_string(string)` and `reverse_string(string)`. Their responsibilities are explained below.

2.1.1 `clean_input_string(string)`

This function will take the input string as a parameter and "clean it up" so that the subsequent tasks are easier to execute. In order to clean up the input string, you will have to remove all occurrences of the following characters from the string:

1. .
2. ,
3. !

4. ?
5. ’

Example:

> **Input:** The cat and the dog saw the cat jump over the fence while the cat chased the dog around the yard with the cat and the dog both running.
>
> **Output:** The cat and the dog saw the cat jump over the fence while the cat chased the dog around the yard with the cat and the dog both running

2.1.2 `reverse_string(string)`

This function will take the “clean” input string (i.e., the output of the `clean_input_string` function) and reverse it.

Example:

> **Input:** The cat and the dog saw the cat jump over the fence while the cat chased the dog around the yard with the cat and the dog both running
>
> **Output:** running both dog the and cat the with yard the around dog the chased cat the while fence the over jump cat the saw dog the and cat The

2.2 Duplicate removal

In this part, you will have to write a function named `remove_duplicates(string)` which will take the reversed string from part 1 as a parameter, and remove all duplicate words from that string. You should keep the first occurrence of the word and remove all other occurrences. The comparison is case sensitive, so that means ”The” and ”the” are considered two unique words. But ”the” and ”the” are duplicate words.

Example:

> **Input:** running both dog the and cat the with yard the around dog the chased cat the while fence the over jump cat the saw dog the and cat The
>
> **Output:** running both dog the and cat with yard around chased while fence over jump saw The

2.3 Median calculation

The median is the middle value of a set of numbers when they are arranged in order from smallest to largest. If the set has an odd number of values, the median is the middle number. If the set has an even number of values, the median is the average of the two middle numbers.

You will have to write a function named `calculate_median_length(string)` which will take the string without any duplicates (from part 2) as a parameter, and calculate the median word length from that string. The output of this function has to be an integer. Therefore if the actual median is 3.5, then the output of this function should be 3.

Example:

> **Input:** running both dog the and cat with yard around chased while fence over jump saw The
>
> **Output:** 4

Marking Rubric: This question will be marked out of 30 for correctness (pass test cases):

- 30/30: Pass all 5 of the test cases
- 24/30: Pass 4 of the test cases

- 18/30: Pass 3 of the test cases
- 12/30: Pass 2 of the test cases
- 6/30: Pass any of the test cases

3 Writing and testing your solution

For the programming questions, you should edit the provided files and add your solution. You can check your solution by running `driver.py`. You can see the output of your program in the `Output/` folder, and any errors in the `Error/` folder.

Please do not change the driver file, only edit the solution file.

You can run the driver in a terminal as follows (assuming your working directory is the assignment folder):

```
$ cd assignment1
$ cd Question1
$ python3 driver.py
All tests passed!
$ cd ../Question2
$ python3 driver.py
All tests passed!
$
```

Please add your name, student number, ccid, operating system and python version at the top of each solution file by replacing the provided comments.

4 Submission Instructions

Please submit a single file to eclass, called `ccid.zip`. Please create a folder with the name as your ccid, put your solutions in that folder, zip it and upload it. It is important for the files to have the same name as described below.

```
ccid/
    solutionQuestion1.py
    solutionQuestion2.py
    writtenQuestions.pdf
```