CSE1322L Assignment 5 - Fall 202

Their algorithm is comprised of a total of 4 observations. These observations dictate how much

implement Observations 1, 2, and 3(a) in this assignment.

of the text can be skipped, based on the mismatch between the pattern and the text. We will not build this table but, instead, calculate the skip every time we need to slide the pattern down the text.

And we'll do so recursively.

Requirements

The features described below must be in your program.

The driver must have 4 static methods:

- main()
- lengthOfMatch(), which takes in 2 strings and returns an integer
- calculateSkip(), which takes in a character and a string, and returns an integer
- findString(), which takes in 2 strings and returns an integer

lengthOfMatch():

inputs have in common. It stops counting once both strings have differing characters or if both inputs run out of characters. **The method assumes both inputs have the same length**.

`lengthOfMatch("", "") returns 0`

Don’t worry about the cases which return 0. We’ll compensate for them.

```
calculateSkip(‘a’, “”) returns 0
calculateSkip(‘a’, “banana”) returns 0
calculateSkip(‘n’, “generator”) returns 6
calculateSkip(‘z’, elite) returns 5
calculateSkip(‘f’, “decision”) returns 8
```

findString():

- This method tries to **recursively** find pattern in text, returning either the first position of where pattern can be found in the original text, or -1 if pattern isn’t present in text.
- Below you can find a description on how to achieve this, which is based on the original Fast String Search Algorithm. The description has examples to help you understand what’s going on.
- If you would like to figure it out by yourself with the description above, you may skip the **DESCRIPTION** section below and work on main().

DESCRIPTION

- **If the pattern is larger than the text, return -1**
-

original_text = “ly-that.--at-that-point”
pattern = “at-that”

lengthOfMatch(“ly-that”, “at-that”) returns 5

position_of_character_to_save: 7 – 5 – 1 = **1**
character_to_save: ‘y’

- Next, we must split the pattern into two: the part before the match (which includes the mismatched character) and the part that matches. **Save both of these into separate strings**. You will need a combination of the substring() method that all strings have, as well as the result of lengthOfMatch().

```
original_text = “ly-that.--at-that-point”
pattern = “at-that”

lengthOfMatch(“ly-that”, “at-that”) returns 5

subpattern_before_match = “at”
subpattern_that_matched = “-that”
```

- Next, we want to calculate how much of the text we can skip, based on where the mismatched character appears in the pattern. **Call calculateSkip() using the mismatched character and the subpattern_that_matched which you saved above. Save this result.**

```
original_text = “ly-that.--at-that-point”
pattern = “at-that”
character_saved = ‘y’
subpattern_before_match = “at”
subpattern_that_matched = “-that”

skip = calculateSkip(character_saved, subpattern_that_matched)
```

- If the skip above turns out to be less than the length of subpattern_that_matched, that means that the mismatched_character can be found in the subpattern_that_matched. As such, we only want to slide the pattern a single character down the text: **replace skip with (1 + lengthOfMatch)**
- Otherwise, it means that the mismatched_character isn’t in subpattern_that_matched. This means we want to slide the pattern down to the first occurrence of mismatched_character in subpattern_before_match: **replace skip with calculateSkip(character_saved, subpattern_before_match).**

- Now is time to slide the pattern down the text, recursively. Call findString() passing the original text without the first (skip) characters and the pattern. **Save this result**.

  ```
  result = findString(text.substring(skip), pattern)
  ```

- **If the result above is -1, return -1**. This is because one of the recursive calls has hit our very first base case, and we must transmit this result upwards to the previous recursive call.
- Otherwise, **return (skip + result)**

Here's what your call stack will roughly look like.

text = "which-finally-halts.--at-that-point"
pattern = "at-that"
pattern found at position = 7 + 4 + 6 + 2 + 3 + 0 = 22

findString("which-finally-halts.--at-that-point", "at-that") // slides 7. Returns (7 + number_below)

Considerations

Remember that you will get partial credit for partial work. Try to deliver as much of the assignment as you can.

You may add any helper methods you believe are necessary, but you will not get points for them.

This assignment is about recursion. No loops are necessary to complete it.

Recall that, for a method to be used recursively, it needs to have at least two cases: one base case (solving the simplest instance of the problem) and one recursive case (which simplifies the problem and calls the function again, with the simplified parameters). The recursive case must eventually converge towards the base case.

You will likely make heavy use of the String's substring() method. Be sure to read up its documentation online so you are aware of how it works.

Example: [User input in red]

```
[Pattern Matcher]
Enter original text: Lorem ipsum dolor sit amet, consectetur adipiscing elit,
```

Submitting your answer:

Please follow the posted submission guidelines here:
https://ccse.kennesaw.edu/fye/submissionguidelines.php

Ensure you submit before the deadline listed on the lab schedule for CSE1322L here:
https://ccse.kennesaw.edu/fye/courseschedules.php