CSE1322L Assignment 5 - Fall 2024

Introduction:

In 1977, Dr. Robert S. Boyer and Dr. J. Strother Moore published a paper describing an algorithm to find a string of characters (called a pattern) in a large text called “Fast String Searching Algorithm”. You can read it here. The easiest way to check if a pattern can be found in a text is by starting at the beginning of the text and then matching each letter of the pattern against the text. If a letter matches, check the next letter. If there is a mismatch, slide the pattern one character to the right and then try to match the whole pattern again.

```
which-finally-halts.--at-that-point
at-that (mismatch. Move one to the right)

which-finally-halts.--at-that-point
 at-that (mismatch. Move one to the right)

which-finally-halts.--at-that-point
  at-that (mismatch. Move one to the right)

etc.
```

The approach above, while guaranteed to find if a match exists or not, is slow as every character of the text and the pattern must potentially be checked more than once. The reason the Boyer-Moore approach was considered fast was because the algorithm allows a computer to skip checking parts of the text depending on which part of pattern caused a mismatch on the text. The idea is to compare the pattern backwards relative to the text, and then slide the pattern forwards a number of character, depending on the character that caused the mismatch.

```
which-finally-halts.--at-that-point
at-that (f does not show up in pattern: Skip past f)

which-finally-halts.--at-that-point
       at-that (- shows up in pattern: Skip to match pattern’s last -)

which-finally-halts.--at-that-point
           at-that (t matched but a did not.
                    l does not show up in pattern:
                    Skip past l)

etc.
```

Their algorithm is comprised of a total of 4 observations. These observations dictate how much of the text we can skip checking based on which character in pattern failed to match. We will implement Observations 1, 2, and 3(a) in this assignment.

Note that their original paper describes building a skip table, so we can quickly lookup how much 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():
  - This method **recursively** determines, from right to left, how many characters its 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
    lengthOfMatch("moderator", "generator") returns 6
    lengthOfMatch("bbanana", "bananas") returns 0
    lengthOfMatch("elite", "elite") returns 5
    lengthOfMatch("decision", "taxation") returns 3
    ```

- calculateSkip():
  - This method **recursively** determines, from right to left, the first time the character in the input occurs in the string in the input. **You cannot use lastIndexOf().** Note the following:
    - Even though we are counting right to left, the "first" character is still at position 0 relative to the end of the string
    - If a character is not present in the input string, then we pretend the character exists at the start of the string. This is because this method is being used to calculate how many characters we can slide the pattern down the text; and if the character cannot be found in the pattern, we can skip the whole pattern.

        - 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**
- Calculate the length of the match between the start of text and pattern. Use lengthOfMatch() for this. Recall that lengthOfMatch() requires both of its inputs to have the same length, so you’ll have to acquire a substring of text to achieve this.

```
original_text = “which-finally-that.--at-that-point”
pattern = “at-that”

lengthOfMatch(“which-f”, “at-that”) returns 0
```

- **If the length of the match matches the length of the pattern, return 0**. This is because we found the pattern at the start of the text.

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

lengthOfMatch(“at-that”, “at-that”) returns 7
length of pattern == lengthOfMatch(“at-that”, “at-that”)

findString(“at-that-point”, “at-that”) returns 0
```

- If the step above did not trigger a return, that means that the text and the pattern have at least one character which mismatches. That character can be found in text at the position (pattern.length() - lengthOfMatch() – 1). **Store that character in a variable**.

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)
        findString("inally-halts.--at-that-point", "at-that")          // slides 4. Returns (4 + number_below)
                findString("ly-halts.--at-that-point", "at-that")      // slides 6. Returns (6 + numbers_below)
                        findString("ts.--at-that-point", "at-that")    // slides 2. Returns (2 + numbers_below)
                                findString(".--at-that-point", "at-that")// slides 3. Returns (3 + numbers_below)
                                        findString("at-that-point", "at-that")    // slides 0, returns 0
```

text = "which-finally-halts.--at-that-point"
pattern = "at-chat"
pattern found at position = -1 (pattern not found)

```
findString("which-finally-halts.--at-that-point", "at-chat")                          // slides 7. Returns -1
        findString("inally-halts.--at-that-point", "at-chat")                         // slides 4. Returns -1
                findString("ly-halts.--at-that-point", "at-chat")                     // slides 6. Returns -1
                        findString("ts.--at-that-point", "at-chat")                   // slides 2. Returns -1
                                findString(".--at-that-point", "at-chat")             // slides 3. Returns -1
                                        findString("at-that-point", "at-chat")        // slides 4, Returns -1
                                                findString("hat-point", "at-chat")    // slides 7, Returns -1
                                                        findString("nt", "at-chat")   // Cannot slide, Returns -1
```

END OF DESCRIPTION

- main() must prompt the user for two strings: the text where the pattern must be searched through, and the pattern they are looking for. If pattern doesn't show up in the text, print a message saying so. Otherwise print the position of where the pattern first appears in the text.

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,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est
laborum.
Enter pattern to find: Except

Pattern found at position 335
```

Example: [User input in red]

```
[Pattern Matcher]
Enter original text: Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est
laborum.
Enter pattern to find: Exception

Pattern could not be found in text!
```

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