Digdarshan Kunwar
Computer Science

Problem 1.1:
Boyer Moore Bad Character Rule (2+2+2+2 = 8 points)
Let Σ = {A, B, C, D} be an alphabet and t ∈ Σ ∗ be a text of length n and p ∈ Σ ∗ be a pattern of length m. We are looking for the first occurrence of p in t.
Consider the text t =ABAABDABBABABBCABABA and the pattern p = BABA. You are asked to carry out string search. Write down how the search proceeds using the notation used on the slides. Show each alignment (via indentation) and indicate comparisons performed with a capital letter and comparisons not performed with lowercase letters.

- Execute the naive string search algorithm. Show all alignments and indicate comparisons performed by writing uppercase characters and comparisons skipped by writing lowercase characters. How many alignments are used? How many character comparisons are done?

| | A | B | A | A | B | D | A | B | B | A | B | A | B | B | C | A | B | A | B | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | |
| 1 | B | a | b | a | | | | | | | | | | | | | | | | |
| 2 | | B | A | B | a | | | | | | | | | | | | | | | |
| 3 | | | B | a | b | a | | | | | | | | | | | | | | |
| 4 | | | | B | a | b | a | | | | | | | | | | | | | |
| 5 | | | | | B | A | b | a | | | | | | | | | | | | |
| 6 | | | | | | B | a | b | a | | | | | | | | | | | |
| 7 | | | | | | | B | a | b | a | | | | | | | | | | |
| 8 | | | | | | | | B | A | b | a | | | | | | | | | |
| 9 | | | | | | | | | B | A | B | A | | | | | | | | |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | | | | | | | | | | |

| 🟩 | First Occurrence |
|---|---|

No of Alignments used for first occurrence of p in t: 9
No of character comparisons for first occurrence of p in t: 16

- Execute the Boyer-Moore string search algorithm with the bad character rule only. How many alignments are used? How many character comparisons are done?

| | A | B | A | A | B | D | A | B | B | A | B | A | B | B | C | A | B | A | B | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | |
| 1 | b | a | B | A | | | | | | | | | | | | | | | | |
| 2 | | b | a | b | A | | | | | | | | | | | | | | | |
| 3 | | | b | a | b | A | | | | | | | | | | | | | | |
| 4 | | | | | | | b | A | B | A | | | | | | | | | | |
| 5 | | | | | | | | b | a | b | A | | | | | | | | | |
| 6 | | | | | | | | | B | A | B | A | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |
|---|---|
| 🟩 | First Occurrence |

No of Alignments used for first occurrence of p in t: 6
No of character comparisons for first occurrence of p in t: 12

- Determine the two-dimensional lookup table for the given pattern p, where each row represents a character of the alphabet and each column an index position for the pattern p (first index position 0).

|  | B | A | B | A |
|---|---|---|---|---|
| A | 0 | - | 0 | - |
| B | - | 0 | - | 0 |
| C | 0 | 1 | 2 | 3 |
| D | 0 | 1 | 2 | 3 |

- The description of the bad character rule in the slides assumes that there is a two-dimensional lookup table indexed by the character not matching the current character of the pattern and the current position within the pattern. An alternative is to use a one-dimensional lookup table, which stores for every character the last occurance in the pattern. If the character does not exist in p, the lookup table contains -1. Since the lookup table only stores information about the last occurance of a character in p, it will not always produce optimal shifts. Write down the one-dimensional lookup table for p and execute the Boyer-Moore string search algorithm using only the bad character rule with this one-dimensional lookup table.

Index from Starting from Right
Length of String: 4

|  | B | A | B | A |
|---|---|---|---|---|
| Actual Index | 0 | 1 | 2 | 3 |

**One-Dimensional Table:**

| Letter | Values |
|---|---|
| ? | -1 |
| B | 2 |
| A | 3 |

"?" Other Characters / Alphabets that are not in p

Now the alignment skip table is given by:

*Skip = Length – Actual Index -1*

|  | A | B | A | A | B | D | A | B | B | A | B | A | B | B | C | A | B | A | B | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | b | a | B | A |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  | b | a | b | A |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  | b | a | b | A |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  | b | A | B | A |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  | b | a | b | A |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  | B | A | B | A |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Problem 1.2: leap year in the Gregorian calendar (haskell) (1+1 = 2 points)

In the Gregorian calendar, a leap year occurs if (i) the year is a multiple of four and (ii) the year is not divisible by 100 or (iii) the year is divisible by 400. Note that (ii) and (iii) overlap but (iii) takes precedence. Write a Haskell function isLeapYear to determine whether a year is a leap year or not

- .Write a isLeapYear function using a Boolean expression involving the Boolean operators && (and), || (or), and the Boolean function not.

```
{-| The program finds if given year is leap year -}

  isLeapYear :: Integer-> Bool
  isLeapYear year
   |year `mod` 4 == 0 && year `mod` 100 /= 0 || year `mod` 400 == 0 =True
   |otherwise = False
```

OR

```
{-| The program finds if given year is  leap year -}

isLeapYear1 :: Integer-> Bool
isLeapYear1 year =
  if year `mod` 4 == 0 && year `mod` 100 /= 0
    then True
    else if year `mod` 400 == 0
      then True
      else False
```

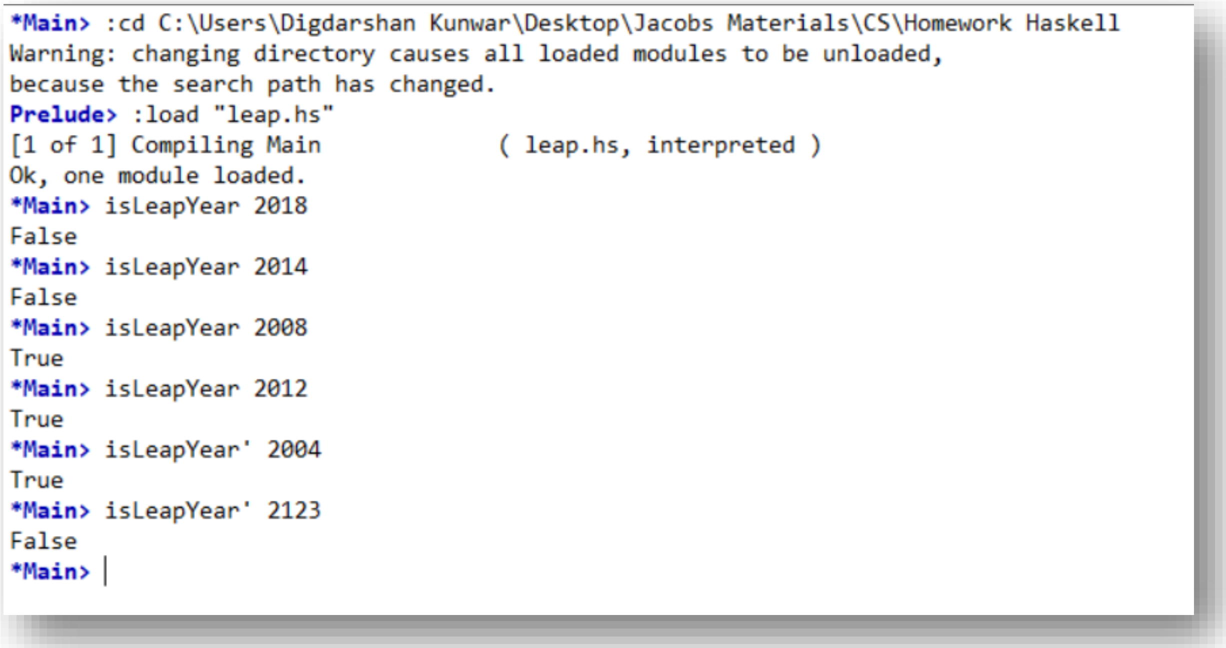
- Write a isLeapYear’ function using guards and without any usage of the Boolean operators && (and), || (or), and the Boolean function not.

```
{-| The program finds if given year is  leap year -}

isLeapYear' :: Integer-> Bool
isLeapYear' year
 |year `mod` 400 == 0 = True
 |year `mod` 4 == 0 = if (year `mod` 100 /= 0) then True else False
 |otherwise = False
```

The Haskell function div returns how many times the first number can be divided by the second one and the function mod returns the remainder of an integer division.

Explain how you have tested your isLeapYear and isLeapYear’ functions.
I have tested my isLeapYear and isLeapYear’ functions by doing a dryrun using a tracetable. And also by white box testing.
Also I test it by running it and checking if the values are true.



```
*Main> :cd C:\Users\Digdarshan Kunwar\Desktop\Jacobs Materials\CS\Homework Haskell
Warning: changing directory causes all loaded modules to be unloaded,
because the search path has changed.
Prelude> :load "leap.hs"
[1 of 1] Compiling Main             ( leap.hs, interpreted )
Ok, one module loaded.
*Main> isLeapYear 2018
False
*Main> isLeapYear 2014
False
*Main> isLeapYear 2008
True
*Main> isLeapYear 2012
True
*Main> isLeapYear' 2004
True
*Main> isLeapYear' 2123
False
*Main> |
```