Introduction to Computer Science
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CH08-320101
Date: 2017-09-12
Due: 2017-09-19

ICS Problem Sheet #1

Problem 1.1: *boyer moore bad character rule* (2+2+2+2 = 8 points)

Let $\Sigma = \{A, B, C, D\}$ be an alphabet and $t \in \Sigma^*$ be a text of length n and $p \in \Sigma^*$ be a pattern of length m. We are looking for the first occurance of p in t.

Consider the text $t = ABAABCABBABABBCABABA$ and the pattern $p = ABAB$.

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.

a) 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?

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

c) 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).

d) 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 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.

Problem 1.2: *factorials (haskell)* (1+1 = 2 points)

Write a recursive function that computes the factorial function mapping a positive natural number n to a positive natural number by calculating the product of all positive natural numbers less than or equal to n. The type signature of the `factorial` function you have to implement is shown below.

```
factorial :: Integer -> Integer
```

a) Write a plain recursive function using pattern matching. Since the type signature allows any integer values, what is the result produced for -4?

b) Write a plain recursive function using guards. Make sure that the function signals an error for non-positive integers. You can signal an error by calling Haskell's `error` function, which has the following type signature:

   ```
   error :: [Char] -> a
   ```

   The error function expects a list of characters as an argument. Since the basic representation of strings in Haskell is a list of characters, the error function actually takes a string as an argument. Hence, you can return the error string "factorial undefined" if `factorial` is called with a non-positive integer.