Assignment 6: Text algorithms

UMaine COS 350

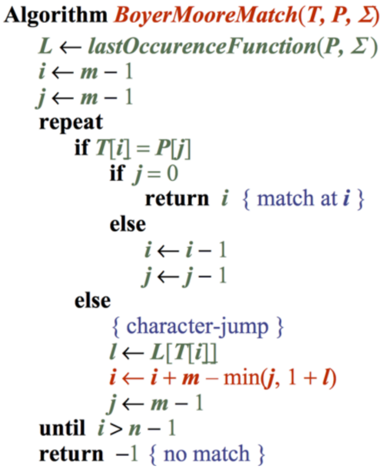
Assigned: 09 April 2020

Due: 01 May 2020

This assignment will ask you to implement two text-matching algorithms and compare them.

Boyer–Moore Algorithm

Implement the Boyer–Moore Algorithm (BMA) for text matching (see Figure 1). Put your code in a file called “BMA.py”. There should be a function, `match`, that



```
Algorithm BoyerMooreMatch($T$, $P$, $\Sigma$)
    $L \leftarrow$ lastOccurenceFunction($P$, $\Sigma$)
    $i \leftarrow m - 1$
    $j \leftarrow m - 1$
    repeat
        if $T[i] = P[j]$
            if $j = 0$
                return $i$ { match at $i$ }
            else
                $i \leftarrow i - 1$
                $j \leftarrow j - 1$
        else
            { character-jump }
            $l \leftarrow L[T[i]]$
            $i \leftarrow i + m - \min(j, 1 + l)$
            $j \leftarrow m - 1$
    until $i > n - 1$
    return $-1$ { no match }
```

Figure 1: The Boyer–Moore Algorithm

- takes three arguments:
  - the text in which to search (a string)
  - the pattern to search for (a string)
  - an alphabet from which characters are drawn for the other two arguments; you can let this default to the upper- and lower-case letters plus a space, for example, as:

    ```
    def match(T,P,alphabet='abcdefghijklmnopqrstuvwxyz' +
                           'ABCDEFGHIJKLMNOPQRSTUVWXYZ' + ' ')
    ```

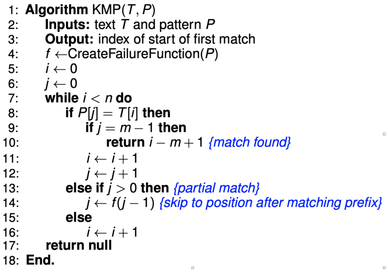
    so you don’t have to enter it each time (or put alphabet in a variable, of course).
- returns:
  - position match began or
  - -1 for no match

  You will need to implement the “last” function, too, of course.

Knuth–Morris–Pratt Algorithm

Implement the Knuth–Morris–Pratt (KMPA) algorithm for text matching (see Figure 2). This should be in a file “KMPA.py”, and it should contain a `match` function as described above.

You will need to implement the failure function, too, of course.



```
Algorithm KMP($T$, $P$)
    Inputs: text $T$ and pattern $P$
    Output: index of start of first match
    $f \leftarrow$CreateFailureFunction($P$)
    $i \leftarrow 0$
    $j \leftarrow 0$
    while $i < n$ do
        if $P[j] = T[i]$ then
            if $j = m - 1$ then
                return $i - m + 1$ {match found}
            $i \leftarrow i + 1$
            $j \leftarrow j + 1$
        else if $j > 0$ then {partial match}
            $j \leftarrow f(j - 1)$ {skip to position after matching prefix}
        else
            $i \leftarrow i + 1$
    return null
End.
```

Figure 2: The KMP Algorithm

Turn in:

Turn in a zipfile named `lastf.py`, where "last" is your last name and "f" is your first initial, all lower case. It should contain:, lower case only an containing *only*:

- Your (well-commented) Python code in two files, "BMA.py" and "KMPA.py". Please pay attention to the specs for the function `maxflow` above, since I'll likely grade this with a program.
- A brief write-up (**in PDF only**) that discusses your implementation, problems you encountered, and results.
- Results of running your code for each algorithm for at least two examples where the match succeeds and one where it does not.