Lab03/Lab04: String Matching

In these two labs you will create two Scala classes `lv.rbs.ds.lab03.KMPmatcher` and `lv.rbs.ds.lab03.BMmatcher` that implement Knuth-Morris-Pratt and Boyer-Moore algorithms respectively and output the JSON data structures that can be used to demonstrate step-by-step behavior of this algorithm.

Both algorithms are widely known and practically important, and there are many implementations available in the Internet (including in Scala), but in this lab assignment we add one more twist – our goal is to show the "debug-like" behavior of both string matchers. See these animations:

http://whocouldthat.be/visualizing-string-matching/
https://people.ok.ubc.ca/ylucet/DS/KnuthMorrisPratt.html
https://people.ok.ubc.ca/ylucet/DS/BoyerMoore.html
https://dwnusbaum.github.io/boyer-moore-demo/

To make your task more manageable, our goal is NOT a fully-functional Web application that would perform these tasks, but only a simple backend to support such Web applications. Below we define the two classes (with their respective public APIs that you should implement.

KMP Matcher (Lab03)

```
class KMPmatcher:
  new KMPmatcher(pattern: String)
  def getPrefixFun(): List[Int]
  def findAllIn(text: CharSequence):
        Iterator[Int]
  def toJson(text: CharSequence): String
```

- `KMPmatcher(pattern:String)` is a constructor to initialize a class instance. Since there are some initialization costs related to the preprocessing of the searchable `pattern`, it might be beneficial to reuse the same matcher with the same pattern to search multiple texts. We therefore avoid passing the target `text` right away.
- `getPrefixFun()` returns the prefix function $\pi(j)$ (for $j = 0, \ldots, m$), where m is the length of the searchable pattern. Prefix function is the key data structure used by the KMP algorithm. It is returned as a list of integers. So the length of this list is $m + 1$.
- `findAllIn(text: CharSequence)` returns an iterator of offsets in the `text` where the searchable `pattern` starts. Please note that in the LAB03 you have to return **all** offsets; it is not sufficient to find just the first instance and then give up.
- `toJson(text: CharSequence)` in this case we return a JSON data structure as in the provided JSON samples.

Boyer-Moore Matcher (Lab04)

```
class BMmatcher:
  new BMmatcher(pattern: String)
  def getGoodSuffixFun(): List[Int]
  def getBadCharFun(): List[(Char,Int)]
  def findAllIn(text: CharSequence):
        Iterator[Int]
  def toJson(text: CharSequence): String
```

This is similar to the previous class. The only difference is that JSON would now be different (since the Boyer-Moore matching has every step scanning backwards: `start` $\geq$ `end`. Also the two data structures (Good Suffix function and Bad Character function) are peculiar to this type of matching.

JSON sample for KMP (Lab03)

```
{
  "algorithm": "KMP",
  "pattern": "ABCDABD",
  "text": "ABC ABCDAB ABCDABCDABDE",
  "prefixFun": [[0,-1],[1,0],[2,0],[3,0],
    [4,0],[5,1],[6,2],[7,0]],
  "steps": [
    { "offset": 0, "start": 0, "end": 3 },
    { "offset": 3, "start": 0, "end": 0 },
    { "offset": 4, "start": 0, "end": 6 },
    { "offset": 8, "start": 2, "end": 2 },
    { "offset": 10, "start": 0, "end": 0 },
    { "offset": 11, "start": 0, "end": 6 },
    { "offset": 15, "start": 2, "end": 6,
    "match": "true" },
    { "offset": 22, "start": 0, "end": 0 }
  ]
  "comparisons": 27
}
```

JSON sample for BM (Lab04)

```
{
  "algorithm": "BM",
  "pattern": "ABCDABD",
  "text": "ABC ABCDAB ABCDABCDABDE",
  "goodSuffixFun": [[0,7],[1,7],[2,7],
      [3,7],[4,7],[5,7],[6,3],[7,1]],
  "badCharFun": [["A",4],["B",5],["C",2],["D",6]],
  "steps": [
    { "offset": 0, "start": 6, "end": 6 },
    { "offset": 4, "start": 6, "end": 6 },
    { "offset": 11, "start": 6, "end": 6 },
    { "offset": 15, "start": 6, "end": 0,
        "match": "true" }
  ],
  "comparisons": 10
}
```