Rocker

Backend Developer Coding Exercise

Overview

The purpose of this exercise is to test your knowledge and ease of use with Scala. We appreciate that you have a life outside of this process, but hope this exercise is a little fun for you, too. If this exercise takes more than a few days, please let us know. Thank you for your time and effort you are putting forward to interview with us!

Deliverables

- ✔ Please share a Github code repository with rockerrecruit when you feel you have a good solution.
- ✔ Code to implement a version of the below instructions.
- ✔ A README containing instructions so that we know how to build and run your code.

Instructions

The exercise is to write a command line driven text search engine, with usage being:

```
sbt
> runMain test.SimpleSearch directoryContainingTextFiles
```

This should read all the text files in the given directory, building an *in memory* representation of the files and their contents, and then give a command prompt at which interactive searches can be performed.

- An example session might look like:

```
$ runMain test.SimpleSearch /foo/bar
14 files read in directory /foo/bar
search> to be or not to be
file1.txt : 100% file2.txt : 90%
search>  cats
no matches found
search> :quit
```

The search should take the words given on the prompt and return a list of the top 10 (maximum) matching filenames in rank order, giving the rank score against each match.

Note: We would like to see a working console application, please focus your attention on the search algorithm and the basic console functionality (you can assume that the input strings are the same).

Ranking

- The rank must be 100% if a file contains all the words
- It must be 0% if it contains none of the words
- It should be between 0 and 100 if it contains only some of the words but the exact ranking formula is up to you to choose and implement

Things to consider in your implementation (please read carefully)

- We are expecting 'production like' code, i.e. fully tested and well structured code, but not necessarily feature complete
- You should be spending 3-4 hours on the task, so basic functionality is better than a complex scoring algorithm and no console application.
- External libraries are forbidden, other than test/mocking frameworks
- Ranking score design — start with something basic, then iterate as time allows
- The in memory representation — searches should be relatively efficient
- What constitutes a word?
- What constitutes two words being equal (and matching)?

Resources

- If you need text files for testing, you can find an archive here: https://drive.google.com/drive/folders/1FWkAQsezG6SRKhgeYTnwC2XutAbf590U?usp=sharing
- **Example starting point, this may not be as testable as you would want.**

```scala
import java.io.File


import scala.util.Try


object Main extends App {
 Program
   .readFile(args)
   .fold(
     println,
     file => Program.iterate(Program.index(file))
   )
}


object Program {
 import scala.io.StdIn.readLine


 case class Index() // TODO: Implement this


 sealed trait ReadFileError


 case object MissingPathArg extends ReadFileError
 case class NotDirectory(error: String) extends ReadFileError
 case class FileNotFound(t: Throwable) extends ReadFileError


 def readFile(args: Array[String]): Either[ReadFileError, File] = {
   for {
     path <- args.headOption.toRight(MissingPathArg)
     file <- Try(new java.io.File(path))
       .fold(
         throwable => Left(FileNotFound(throwable)),
         file =>
           if (file.isDirectory) Right(file)
           else Left(NotDirectory(s"Path [$path] is not a directory"))
       )
   } yield file
```

```
  }


  // TODO: Index all files in the directory
  def index(file: File): Index = ???


  def iterate(indexedFiles: Index): Unit = {
    print(s"search> ")
    val searchString = readLine()
    // TODO: Make it print the ranking of each file and its corresponding score
    iterate(indexedFiles)
  }
}
```