Harshil Shah – SEC01 (NUID 002780887)

Big Data System Engineering with Scala
Spring 2023
Assignment - 6 Web Crawler

- GitHub Repo URL - https://github.com/harshilshahneu/CSYE7200-Harshil-Shah/tree/Fall2022

- List of Tasks Implemented

Implement the primitive web crawler that is partly complete in the crawler package on our class repo (you must use the Fall2022 branch).

There are three TO BE IMPLEMENTED to complete, with a total point value of 32. You may also earn up to 10 bonus points for suggestions on how to improve the web crawler (detailed code not required but you do need to explain in words what you would do). Two of these are in WebCrawler.scala. The other is in MonadOps.scala as follows:

Please ensure that you pull the latest versions of WebCrawler.scala, HTMLParser.scala and MonadOps.scala. You can get these from the class repo (see Course Material/Resources/Class Repository), the module name for this assignment is assignment-web-crawler.

For the processing of a Node, we will need to refer back to the way Scala processes XML documents which we covered in Serialization (that's the purpose of the tagsoup library). Hint: you can get all of the anchor, viz. the "a" nodes using

ns \\ "a"

You can get the "href" property from these nodes using "\" and "@href".

There is also the main program but if you run that, you will need to provide Program arguments consisting of URL(s) at which to start crawling.

I strongly advise you to take advantage of the various hints. Also, make sure you know the types of any intermediate results that you derive: either by adding type annotation (in IDEA, just do option/alt + return/enter and it will give you the option of adding a type annotation) or by just selecting an identifier and displaying its type (in IDEA, that would be ctrl/shift/P). Any time you know the type you're starting with--and you know the type

of result you need--now you just have to find existing methods to convert from one to the other.

- Code

WebCrawler.scala

```scala
/**
 * Method to get a list of URLs referenced by the given URL.
 *
 * @param url a URL.
 * @return a Future of Seq[URL] which corresponds to the various A links in the HTML.
 */
def wget(url: URL)(implicit ec: ExecutionContext): Future[Seq[URL]] = {
    // Hint: write as a for-comprehension, using the method createURL(Option[URL], String) to get the appropriate URL for relative links
    // 16 points.
    def getURLs(ns: Node): Seq[Try[URL]] = {
        for {n <- ns \\ "a"; nh <- n \ "@href"} yield validateURL(createURL(Some(url), nh.toString))
    }

    def getLinks(g: String): Try[Seq[URL]] = {
        val newY: Try[Node] = HTMLParser.parse(g) recoverWith { case f => Failure(new RuntimeException(s"parse problem with URL $url: $f")) }
        for (n <- newY; uys = getURLs(n); us <- MonadOps.sequenceForgiveSubsequent(uys) { case _: WebCrawlerProtocolException => true; case _ => false }) yield us
    }
    // Hint: write as a for-comprehension, using getURLContent (above) and getLinks above. You will also need MonadOps.asFuture
    // 9 points.
    for (
        urlContent <- getURLContent(url);
        us <- MonadOps.asFuture(getLinks(urlContent))
    ) yield us
```

```scala
/**
 * Method to extract an Option[X] from an Either[Throwable, X].
 *
 * Hint: this one is easy: just look for a method which turns an Either into a Option.
 * 7 points.
 *
 * @param xe an Either[Throwable, X].
 * @tparam X the underlying type.
 * @return if xe is a Right(x) then Some(x) else None.
 */
def asOption[X](xe: Either[Throwable, X]): Option[X] = xe.toOption
```

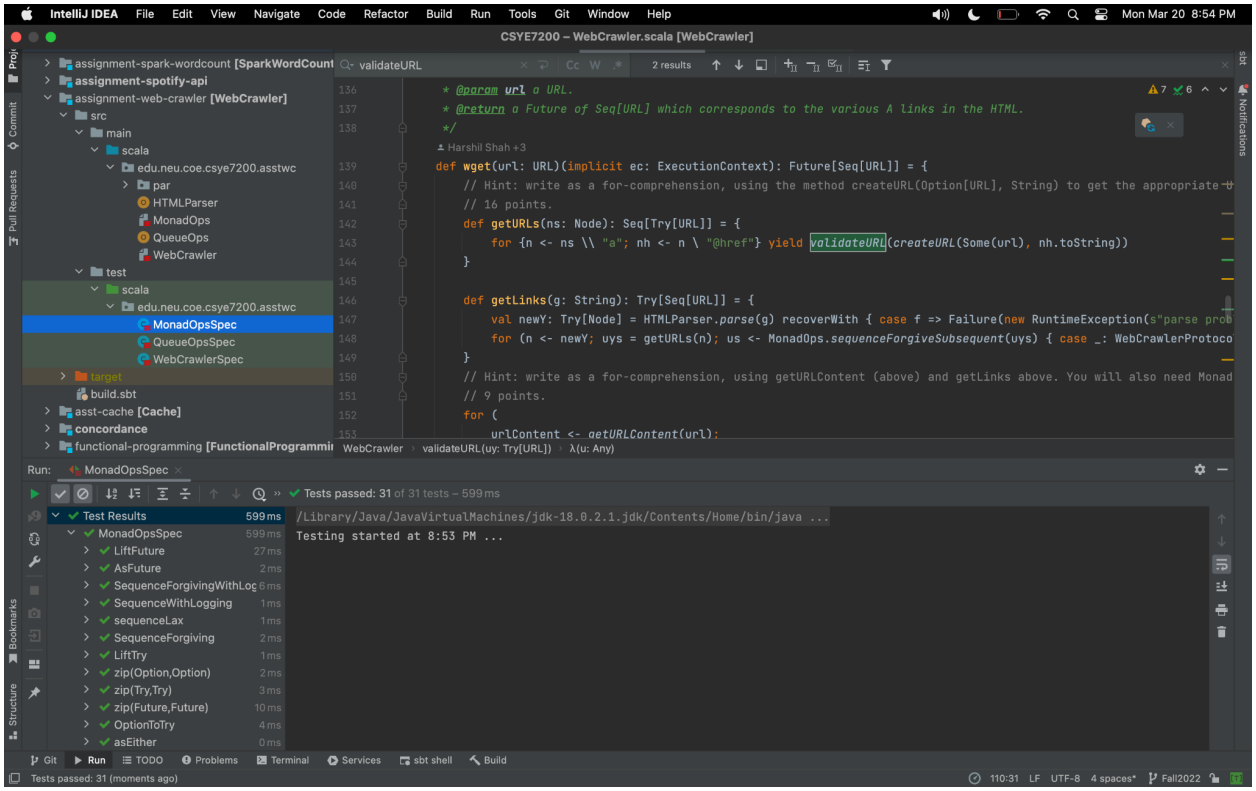
- Improvements

1. Implement caching: The current implementation does not cache crawled pages, which means that pages may be requested multiple times if they are linked from multiple sources. Adding caching can help reduce the number of requests and improve crawl speed.
2. Optimize the crawling process: The current implementation uses a simple recursive algorithm to crawl web pages, which may not be efficient for large-scale crawls or when dealing with slow servers. An alternative approach would be to

use a concurrent or distributed crawler, which can perform multiple requests in parallel or distribute the workload across multiple machines.

Execution Screenshots

-MonadOpsSpec



-WebCrawlerSpec

IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window Help Mon Mar 20 8:54 PM

CSYE7200 – WebCrawler.scala [WebCrawler]

```scala
    * @param url a URL.
    * @return a Future of Seq[URL] which corresponds to the various A links in the HTML.
    */
    Harshil Shah +3
    def wget(url: URL)(implicit ec: ExecutionContext): Future[Seq[URL]] = {
        // Hint: write as a for-comprehension, using the method createURL(Option[URL], String) to get the appropriate U
        // 16 points.
        def getURLs(ns: Node): Seq[Try[URL]] = {
            for {n <- ns \\ "a"; nh <- n \ "@href"} yield validateURL(createURL(Some(url), nh.toString))
        }

        def getLinks(g: String): Try[Seq[URL]] = {
            val newY: Try[Node] = HTMLParser.parse(g) recoverWith { case f => Failure(new RuntimeException(s"parse prob
            for (n <- newY; uys = getURLs(n); us <- MonadOps.sequenceForgiveSubsequent(uys) { case _: WebCrawlerProtoco
        }
        // Hint: write as a for-comprehension, using getURLContent (above) and getLinks above. You will also need Monad
        // 9 points.
        for (
            urlContent <- getURLContent(url);
```

Tests passed: 10 of 10 tests – 15 sec 792 ms

```
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java ...
Testing started at 8:54 PM ...
```