UECS3223 Cloud Computing

Jan 2024

Group Project

**Group members**

|  |  |  |
| --- | --- | --- |
| Name | Student ID | Trimester and year of study |
| Andrew Woon Mun Hong | 2102584 | Y2T3 |
| Gan Wei Jia | 2102338 | Y2T3 |
| Lim Jun Hau | 2102629 | Y2T3 |
| Liew Chun Kin | 2103388 | Y2T3 |

**Marks breakdown**

|  |  |  |
| --- | --- | --- |
| Parts | Marks allocated | Comments |
| a) Proper instructions on how to set up the workflow for the data analysis and / or ETL operations so that they run properly | /20 |  |
| b) Implementation of MapReduce / Spark applications or Hive / Spark SQL queries that perform AT LEAST 3 types of different analysis on the selected dataset. | /30 |  |
| c) Complexity of analysis performed; as given by the analysis algorithm implemented in MapReduce / Spark or the queries executed in Hive / Spark SQL | /30 |  |
| d) Proper explanation of the selected analysis being performed, along with comments in source code to explain how the analysis is achieved. | /20 |  |
| Total | **/100** |  |

1. The raw dataset is a source code that was found on the website GitHub (https://github.com/thegr8dev/doctorpatientportal)
2. The source code dataset which consists of (.php) and (.js) type files is first loaded into the Scala platform to perform ETL or analysis operations with Spark (in Scala)
3. The dataset is loaded into the DBFS of Databricks which can be used to perform Spark (in Scala) analysis. Then a notebook is created to perform the commands on the dataset. For finding the dependencies, we used the local PC's command prompt for scala commands and output the diagrams of dependencies by using graphwiz.
4. The analysis results obtained from the operations are the general information of the dataset, the source code metrics, the components and subcomponents of the dataset, and the relationship between modules, classes, functions, and data dependencies.
5. **Finding the General Informations**

* **First Step**

**// Define the file path**

**val filePath = "/FileStore/assignment"**

**// Read the content of the PHP file**

**val phpFileRDD = sc.textFile(filePath)**

**// Define the function to process each line**

**def processLine(line: String): String = {**

**// Regular expression to match text outside <>**

**val regex = "([^<]\*)<[^>]\*>".r**

**// Extract all matches and concatenate them with space as separator**

**var attributes = regex.findAllMatchIn(line).map(\_.group(1)).mkString("").trim**

**// Remove "echo" statements**

**attributes = attributes.replaceAll("echo", "")**

**// Remove attributes concatenated with "&"**

**attributes = attributes.replaceAll("&", "")**

**// Remove non-breaking space entities**

**attributes = attributes.replaceAll("nbsp;", "")**

**// Remove special characters except spaces**

**attributes.replaceAll("[^a-zA-Z0-9\\s]", "")**

**}**

**// Apply the function to each line in the RDD**

**val processedLinesRDD = phpFileRDD.map(processLine)**

**val filteredCleanedArray = processedLinesRDD**

**.filter(!\_.contains("jQuery"))**

**.filter(!\_.contains("while"))**

**.filter(!\_.contains("opt"))**

**.filter(!\_.contains("tools"))**

**.filter(!\_.contains("print"))**

**.filter(!\_.contains("alert"))**

**.filter(!\_.contains("table"))**

**.filter(!\_.contains("html"))**

**.filter(!\_.contains("prevlaquokdatasettingsnif0atotalreturn "))**

**.filter(!\_.contains("echo")) // Remove echo statements**

**.filter(!\_.contains("&"))    // Remove attributes concatenated with "&"**

**.filter(!\_.contains("nbsp;")) // Remove non-breaking space entities**

**.filter(!\_.matches(".\*\\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\\.[A-Z|a-z]{2,}\\b.\*")) // Remove email addresses**

**.filter(!\_.matches(".\*\\b\\d{2,4}(-|\\/|\\.)\\d{1,2}\\1\\d{1,4}\\b.\*")) // Remove dates**

**val finalCleanedArray = filteredCleanedArray.filter(\_.replaceAll("\\s", "").matches("\\p{Print}+")).distinct()**

**// Print each processed line row by row**

**finalCleanedArray.collect()**

This code first reads the php file in the selected folder. Then it process each line by matching the text outside <> HTML tags and then capture text content within each line  
It also removes the echo, &, nbsp and the special characters. This process is then apply to phpFileRDD using the map function. Lastly, we filter out some of the remaining details that are not useful.

* **Find Portal**

**val portalInfoRDD = finalCleanedArray.filter { line =>**

**line.toLowerCase.contains("portal") || // Look for lines containing the word "portal"**

**line.toLowerCase.contains("about us") || // Look for lines containing "about us"**

**line.toLowerCase.contains("contact us") || // Look for lines containing "contact us"**

**line.toLowerCase.contains("help") || // Look for lines containing "help"**

**line.toLowerCase.contains("services") || // Look for lines containing "services"**

**line.toLowerCase.contains("benefits") || // Look for lines containing "benefits"**

**line.toLowerCase.contains("testimonials") // Look for lines containing "testimonials"**

**}**

**// Print each line containing portal information**

**portalInfoRDD.collect()**

Later we apply this remaining code to capture information related to portal, about us, contact us, help, services, benefits, testimonials from the code we filter just now

This code finds the information about the portal after we filter out most of the functions and just leave behind some text. We learn that this php code is a Doctor Patient Portal is a basic portal that helps those patients who stand in line for hours to take appointments with doctor in clinic or hospitals, This portal help to reduce lines in hospital a simple registration which is free and with lifetime membership for both doctor and patient Verified doctors with good reputation make our portal more trusty It provides free services like their own dashboard to manage appointments and other things A Personal message system which helps to take precaution advice from a doctor or for other health talk Its also a move in digital india to make paperwork less and work digitally.

The user also can log in and register to the portal and contact the doctor using the portal. It comes with the login form and register form. It also provides help to the user.

* **Find Developer**

**val developerInfoRDD = finalCleanedArray.filter { line =>**

**line.toLowerCase.contains("design") || // Look for lines containing the word "design"**

**line.toLowerCase.contains("idea")      // Look for lines containing the word "idea"**

**}**

**// Print each line containing portal information**

**portalInfoRDD.collect()**

Now we use the first code to find out the developer of this php code. We find using the key word of design and idea and get the information that this php code is **designed By  Ankit Kabra, and the idea behind this is Vardan Sharma.**

* **Find Appointment**

**val appointmentInfoRDD = finalCleanedArray.filter { line =>**

**line.toLowerCase.contains("appointment")  // Look for lines containing the word "appointment"**

**}**

**val appointmentInfoRDDFinal = appintmentInfoRDD.filter(!\_.contains("portal")) //filter out the portal information that already retrieve**

**// Print each line containing portal information**

**appointmentInfoRDDFinal.collect()**

Now we want to get the information about the appointment. We filter out the information on the portal we already retrieve and find a line containing the appointment. We learn that the portal stores total appointments, today's appointments, and upcoming appointments. It also records the appointment status and time. It also has a form to allow the user to fill out the appointments, and allow the patient to pay for the appointments.

* **Find Doctor Profession**

**// Filter lines containing information about different professions**

**val professionInfoRDD = finalCleanedArray.filter { line =>**

**line.toLowerCase.contains("denitst") ||       //find dentist**

**line.toLowerCase.contains("cardiologist") ||  //find cardiologist**

**line.toLowerCase.contains("allergist") ||   //find allergist**

**line.toLowerCase.contains("physcit") ||   //find physician**

**line.toLowerCase.contains("gynecologist")   //find gyneoclogist**

**}**

**// Print each line containing profession information**

**professionInfoRDD.collect()**

In this code we want to filter out all the unnecessary details and just leave behind the doctor profession of dentist, cardiologist, allergist, physician and gynecologist.

1. **Finding the Source code metrics**

|  |  |
| --- | --- |
| Code Metrics (.php) & (.js) | |
| Total functions | 2876 |
| Total LOC | 32538 |
| Total files | 52 |
| Total variables | 1761 |
| Total classes | 1636 |
| Total includes | 66 |
| Total divisions (div) | 1357 |

* **Extracting the source code metrics: class, div, var, includes, and functions**

import org.apache.spark.sql.SparkSession

// Read all .php files in the directory

val filePath = "/FileStore/assignment"

val filesRDD = spark.sparkContext.wholeTextFiles(filePath)

// Count occurrences of specific words in .php files

val specificWordsCount = filesRDD.flatMap { case (\_, content) =>

  // Split content into words and filter specific words

  val words = content.toLowerCase.split("\\W+")

  words.filter(word => Set("class", "div", "var", "includes", "function").contains(word))

}.map(word => (word, 1)).reduceByKey(\_ + \_)

// Print the count of specific words

specificWordsCount.collect().foreach { case (word, count) =>

  println(s"Occurrences of '$word': $count")

}

First the files are loaded into filesRDD, then flatten to filter the class, div, var, includes, function and count the occurrence of the source code metrics.

* **Extracting the LOC, and number of files in the source code dataset.**

// Load .js and .php files from the specified directory

val filesRDD = spark.sparkContext.wholeTextFiles("/FileStore/assignment/\*.{js,php}")

// Calculate total lines of code (LOC) and number of files for JavaScript files

val (jsTotalLOC, jsNumFiles) = (filesRDD.filter { case (path, \_) => path.endsWith(".js") }

                                    .map { case (\_, content) => content.split("\n").length }

                                    .sum(),

                                 filesRDD.filter { case (path, \_) => path.endsWith(".js") }.count())

// Calculate total lines of code (LOC) and number of files for PHP files

val (phpTotalLOC, phpNumFiles) = (filesRDD.filter { case (path, \_) => path.endsWith(".php") }

                                     .map { case (\_, content) => content.split("\n").length }

                                     .sum(),

                                  filesRDD.filter { case (path, \_) => path.endsWith(".php") }.count())

// Combine the results

val totalLOC = jsTotalLOC + phpTotalLOC

val totalNumFiles = jsNumFiles + phpNumFiles

// Output combined results

println("Combined Metrics:")

println(s"Total Lines of Code: $totalLOC")

println(s"Total Number of Files: $totalNumFiles")

From the code above, we can filter and extract the line of codes that is in the source code and the number of files that are used for the analysis.

1. Finding the Components and Subcomponents

* **Extracted Class code**

**// Define the file path and load the RDD**

**val filePaths = "/FileStore/assignment"**

**val combinedRDD = sc.textFile(filePaths)**

**// Define a regex pattern to extract class attributes within double quotes**

**val classPattern = """class="([^"]\*)"""".r**

**// Filter lines containing "class" attributes and extract them**

**val extractedClasses = combinedRDD**

**.flatMap { line =>**

**classPattern.findAllMatchIn(line).flatMap { matchResult =>**

**val classAttributeValue = matchResult.group(1).trim**

**if (!classAttributeValue.contains("'")) {**

**val classes = classAttributeValue.split("\\s+").filter(\_.nonEmpty)**

**classes.map(classAttr => (classAttr, 1))**

**} else {**

**Iterator.empty**

**}**

**}**

**}**

**.reduceByKey(\_ + \_) // Reduce by key to aggregate counts for each class attribute**

**// Collect the results and sort the extracted class attributes by count in descending order**

**val sortedClassCounts = extractedClasses**

**.map { case (classAttr, count) => (count, classAttr) } // Swap (classAttr, count) to (count, classAttr) for sorting**

**.sortByKey(false) // Sort by count in descending order**

**.collect()**

**// Print the class attributes sorted by count in descending order**

**sortedClassCounts.foreach { case (count, classAttr) =>**

**println(s"Class attribute '$classAttr' occurred $count times")**

**}**

**// Calculate the sum of counts for all class attributes**

**val totalCountClassAttributes = sortedClassCounts.foldLeft(0)((sum, tuple) => sum + tuple.\_1)**

**println(s"\nTotal count of extracted class attributes: $totalCountClassAttributes")**

This code is to perform finding the “class” and the attributes inside of the class. First, we need to verify the class pattern which helps to find attributes on the “class=”. Then we perform splitting and group the attributes using reduceByKey. After that, we print the result as descending order and the total extracted count.

****

****

* **Extracted Div code**

**// Define the file path and load the RDD**

**val filePaths = "/FileStore/assignment"**

**val combinedRDD = sc.textFile(filePaths)**

**// Filter lines containing "div", extract div names, and count occurrences**

**val extractedDivs = combinedRDD**

**.filter(\_.contains("div")) // Filter lines containing "div"**

**.flatMap { line =>**

**val words = line.split(" ")**

**if (words.nonEmpty) {**

**// Remove special characters from the last word**

**val lastWord = words.last.replaceAll("[\\$<>?;&\\[\\]/()''.]", "")**

**Some(lastWord)**

**} else {**

**None**

**}**

**}**

**.flatMap(\_.split("\\s+")) // Split by whitespace**

**.filter(\_.startsWith("div")) // Filter tokens starting with "div"**

**.map(\_.replaceAll("[\",]", "")) // Extract and clean div names**

**.map((\_, 1)) // Map each div name to (divName, 1) for counting**

**.reduceByKey(\_ + \_) // Count occurrences of each div name**

**// Collect the results and sort the grouped div counts by count in descending order**

**val sortedDivCounts = extractedDivs**

**.map { case (divName, count) => (count, divName) } // Swap (divName, count) to (count, divName) for sorting**

**.sortByKey(false) // Sort by count in descending order**

**.collect()**

**// Print the div counts sorted by count in descending order**

**sortedDivCounts.foreach { case (count, divName) =>**

**println(s"Div '$divName' occurred $count times")**

**}**

**// Print the count of 'div' occurrences after filtering**

**val divOccurrencesCount = extractedDivs.count()**

**println(s"Number of occurrences of 'div' after filtering: $divOccurrencesCount")**

**// Sum up the counts of all div occurrences**

**val totalDivCount = extractedDivs.map(\_.\_2).reduce(\_ + \_)**

**println(s"Total count of div occurrences: $totalDivCount")**

This code is to perform finding the “div” and the attributes inside of the it First, we need to search the “div” by using contains method. Then we perform splitting and group the attributes using reduceByKey. After that, we print the result as descending order and the total extracted count.

**A computer screen shot of a computer code

Description automatically generated**

* **Extracted attributes code**

// Define the file path and load the RDD

val filePaths = "/FileStore/assignment"

val combinedRDD = sc.textFile(filePaths)

// Function to extract PHP variables and their attributes from a line of PHP code

def extractVariableAttributes(line: String): Seq[String] = {

  // Regular expression to match PHP variable assignments starting with $

  val variableRegex = """\$(\w+)\s\*=\s\*\$?([a-zA-Z\_\x7f-\xff][a-zA-Z0-9\_\x7f-\xff\[\]'"]\*)""".r

  // Find all matches of variable assignments in the line

  variableRegex.findAllMatchIn(line)

    .flatMap { matchResult =>

      // Extract the attribute value from the match

      if (matchResult.groupCount >= 2) {

        Some(matchResult.group(1)) // Return the variable name with $

      } else {

        None

      }

    }

    .toList

}

// Extract variable attributes from each line of text and flatten the result

val attributeRDD = combinedRDD

  .flatMap(extractVariableAttributes) // Extract variable attributes

// Count occurrences of each attribute

val attributeCountsRDD = attributeRDD

  .map(attribute => (attribute, 1)) // Map each attribute to (attribute, count)

  .reduceByKey(\_ + \_) // Reduce by key to count occurrences

// Collect the result as a list of (attribute, count) tuples and sort by count in descending order

val attributeCounts = attributeCountsRDD

  .map { case (attribute, count) => (count, attribute) } // Swap (attribute, count) to (count, attribute) for sorting

  .sortByKey(false) // Sort by count in descending order

  .collect()

// Print the result with modified attribute formatting and sorted by count

attributeCounts.foreach { case (count, attribute) =>

  println(s"Attribute inside php: $$${attribute}, Count: $count")

}

// Calculate the sum of attribute counts

val totalAttributeCount = attributeCountsRDD.map(\_.\_2).reduce(\_ + \_)

println(s"Total count of extracted attributes: $totalAttributeCount")

This code is to perform finding the “$” and the attributes inside. First, we define the “$” pattern to find the attribute that link together with “$”. Then we perform splitting and group the attributes using reduceByKey. After that, we print the result as descending order and the total extracted count.

A group of black text

Description automatically generated



* **Extracted function() code**

**import org.apache.spark.rdd.RDD**

**// Define the file path and load the RDD**

**val filePaths = "/FileStore/assignment"**

**val combinedRDD = sc.textFile(filePaths)**

**// Define a regular expression pattern to match functions with options inside parentheses**

**val functionWithOptionsPattern = """function\s\*\((.\*?)\)""".r**

**// Filter lines containing the desired function pattern and extract functions with options**

**val extractedFunctions = combinedRDD**

**.flatMap { line =>**

**// Find all matches of the function pattern within the line**

**functionWithOptionsPattern.findAllMatchIn(line).map { matchResult =>**

**// Extract the options inside the parentheses**

**val options = matchResult.group(1).trim**

**options**

**}**

**}**

**// Count occurrences of each function with options**

**val functionCounts = extractedFunctions**

**.map(options => (options, 1)) // Map each function option to (options, 1) for counting**

**.reduceByKey(\_ + \_) // Reduce by key to aggregate counts**

**// Sort the function counts by count in descending order**

**val sortedFunctionCounts = functionCounts**

**.map { case (options, count) => (count, options) } // Swap (options, count) to (count, options) for sorting**

**.sortByKey(false) // Sort by count in descending order**

**.persist() // Cache the sorted counts for efficient reuse**

**// Collect the sorted results as an array**

**val sortedFunctions = sortedFunctionCounts.collect()**

**// Print the count of each function with options in descending order**

**println("Functions with options (sorted by count in descending order):")**

**sortedFunctions.foreach { case (count, options) =>**

**println(s"Function($options), Count: $count")**

**}**

**// Calculate the sum of all function counts**

**val totalCountFunctions = sortedFunctionCounts.map(\_.\_1).sum()**

**println(s"\nTotal count of functions with options: $totalCountFunctions")**

This code is to perform finding the “function()” and the attributes inside the bracket. First, we define the “function()” pattern to find the attribute that link together with “function()”. Then we perform splitting and group the attributes using reduceByKey. After that, we print the result as descending order and the total extracted count.

**A white background with black text

Description automatically generated**

* **Extracted for loop code**

// Define the file path and load the RDD

val filePaths = "/FileStore/assignment"

val combinedRDD = sc.textFile(filePaths)

// Define regular expression pattern to match `for` loop constructs

val forPattern = """\bfor\s\*\((.\*?)\)""".r

// Filter lines containing `for` loop constructs and extract them with conditions/attributes

val extractedForLoops = combinedRDD

  .flatMap { line =>

    forPattern.findAllMatchIn(line).map(\_.group(1))

  }

// Count occurrences of each `for` loop construct with conditions/attributes

val forLoopCounts = extractedForLoops

  .map(content => (s"for($content)", 1))

  .reduceByKey(\_ + \_)

// Sort the `for` loop counts by count in descending order

val sortedForLoopCounts = forLoopCounts

  .map { case (info, count) => (count, info) }

  .sortByKey(false)

// Collect and print the sorted results

val sortedForLoops = sortedForLoopCounts.collect()

println("For Loop Constructs with Conditions (sorted by count in descending order):")

sortedForLoops.foreach { case (count, info) =>

  println(s"$info occurred $count times")

}

// Calculate the sum of counts for `for` loop constructs

val totalCountForLoops = sortedForLoopCounts.map(\_.\_1).sum()

println(s"\nTotal count of 'for' loop constructs: $totalCountForLoops")

This code is to perform finding the “for()” loop and the attributes inside the loop bracket. First, we define the “for()” pattern to find the attribute that link together with “for()”. Then we map the attributes using reduceByKey. After that, we print the result as descending order and the total extracted count.

**A white background with black text

Description automatically generated**

****

****

* **Extracted switch code**

**// Define regular expression pattern to match `switch` statement constructs**

**val switchPattern = """\bswitch\s\*\((.\*?)\)""".r**

**// Filter lines containing `switch` statement constructs and extract them with conditions/attributes**

**val extractedSwitchStatements = combinedRDD**

**.flatMap { line =>**

**switchPattern.findAllMatchIn(line).map(\_.group(1))**

**}**

**// Count occurrences of each `switch` statement construct with conditions/attributes**

**val switchStatementCounts = extractedSwitchStatements**

**.map(content => (s"switch($content)", 1))**

**.reduceByKey(\_ + \_)**

**// Sort the `switch` statement counts by count in descending order**

**val sortedSwitchStatementCounts = switchStatementCounts**

**.map { case (info, count) => (count, info) }**

**.sortByKey(false)**

**// Collect and print the sorted results**

**val sortedSwitchStatements = sortedSwitchStatementCounts.collect()**

**println("Switch Statement Constructs with Conditions (sorted by count in descending order):")**

**sortedSwitchStatements.foreach { case (count, info) =>**

**println(s"$info occurred $count times")**

**}**

**// Calculate the sum of counts for `switch` constructs**

**val totalCountForswitch = sortedSwitchStatementCounts.map(\_.\_1).sum()**

**println(s"\nTotal count of 'switch' constructs: $sortedSwitchStatementCounts")**

This code is to perform finding the “switch()” and the attributes inside the bracket. First, we define the “switch()” pattern to find the attribute that link together with “switch()”. Then we map the attributes using reduceByKey. After that, we print the result as descending order and the total extracted count.

**A white background with black text

Description automatically generated**

* **Extracted case code**

**// Define regular expression pattern to match `case` clauses**

**val casePattern = """\bcase\s+['"]?(.\*?)['"]?\s\*:""".r**

**// Filter lines containing `case` clauses and extract them with conditions/attributes**

**val extractedCaseClauses = combinedRDD**

**.flatMap { line =>**

**casePattern.findAllMatchIn(line).map(\_.group(1))**

**}**

**// Count occurrences of each `case` clause with conditions/attributes**

**val caseClauseCounts = extractedCaseClauses**

**.map(content => (s"case: $content", 1))**

**.reduceByKey(\_ + \_)**

**// Sort the `case` clause counts by count in descending order**

**val sortedCaseClauseCounts = caseClauseCounts**

**.map { case (info, count) => (count, info) }**

**.sortByKey(false)**

**// Collect and print the sorted results**

**val sortedCaseClauses = sortedCaseClauseCounts.collect()**

**println("Case Clauses with Conditions (sorted by count in descending order):")**

**sortedCaseClauses.foreach { case (count, info) =>**

**println(s"$info occurred $count times")**

**}**

**// Calculate the sum of counts for `case` constructs**

**val totalCountForcase = sortedCaseClauseCounts.map(\_.\_1).sum()**

**println(s"\nTotal count of 'case' constructs: $sortedCaseClauseCounts")**

This code is to perform finding the “case” and the attributes inside the bracket. First, we define the “case” pattern to find the attribute that link together with “case”. Then we map the attributes using reduceByKey. After that, we print the result as descending order and the total extracted count.

**A white background with black text

Description automatically generated**

* **Extracted if condition code**

**// Define regular expression pattern to match `if` statement constructs**

**val ifPattern = """\bif\s\*\((.\*?)\)""".r**

**// Filter lines containing `if` statement constructs and extract them with conditions/attributes**

**val extractedIfStatements = combinedRDD**

**.flatMap { line =>**

**ifPattern.findAllMatchIn(line).map(\_.group(1))**

**}**

**// Count occurrences of each `if` statement construct with conditions/attributes**

**val ifStatementCounts = extractedIfStatements**

**.map(content => (s"if($content)", 1))**

**.reduceByKey(\_ + \_)**

**// Sort the `if` statement counts by count in descending order**

**val sortedIfStatementCounts = ifStatementCounts**

**.map { case (info, count) => (count, info) }**

**.sortByKey(false)**

**// Collect and print the sorted results**

**val sortedIfStatements = sortedIfStatementCounts.collect()**

**println("If Statement Constructs with Conditions (sorted by count in descending order):")**

**sortedIfStatements.foreach { case (count, info) =>**

**println(s"$info occurred $count times")**

**}**

**// Calculate the sum of counts for `if` constructs**

**val totalCountForIf = sortedIfStatementCounts.map(\_.\_1).sum()**

**println(s"\nTotal count of 'if' constructs: $sortedIfStatementCounts")**

This code is to perform finding the “if()” condition and the attributes inside the bracket. First, we define the “if()” pattern to find the attribute that link together with “if()”. Then we map the attributes using reduceByKey. After that, we print the result as descending order and the total extracted count.A white background with black text

Description automatically generated

**Extracted else condition code**

**// Define regular expression pattern to match `else` statement constructs**

**val elsePattern = """\belse\b""".r**

**// Filter lines containing `else` statement constructs**

**val extractedElseStatements = combinedRDD**

**.filter(line => elsePattern.findFirstIn(line).isDefined)**

**.map(\_ => ("else", 1))**

**// Count occurrences of `else` statement constructs**

**val elseStatementCounts = extractedElseStatements**

**.reduceByKey(\_ + \_)**

**// Collect and print the results**

**val sortedElseStatements = elseStatementCounts.collect()**

**println("Else Statement Constructs (sorted by count in descending order):")**

**sortedElseStatements.foreach { case (info, count) =>**

**println(s"$info occurred $count times")**

**}**

This code is to perform finding the “else()” condition and the attributes inside the bracket. First, we define the “else()” pattern to find the attribute that link together with “else()”. Then we map the attributes using reduceByKey. After that, we print the result as descending order and the total extracted count.

**A white background with black text

Description automatically generated**

* **Extracted while loop code**

**// Define regular expression pattern to match `while` loop constructs**

**val whilePattern = """\bwhile\s\*\((.\*?)\)""".r**

**// Filter lines containing `while` loop constructs and extract them with conditions/attributes**

**val extractedWhileLoops = combinedRDD**

**.flatMap { line =>**

**whilePattern.findAllMatchIn(line).map(\_.group(1))**

**}**

**// Count occurrences of each `while` loop construct with conditions/attributes**

**val whileLoopCounts = extractedWhileLoops**

**.map(content => (s"while($content)", 1))**

**.reduceByKey(\_ + \_)**

**// Sort the `while` loop counts by count in descending order**

**val sortedWhileLoopCounts = whileLoopCounts**

**.map { case (info, count) => (count, info) }**

**.sortByKey(false)**

**// Collect and print the sorted results**

**val sortedWhileLoops = sortedWhileLoopCounts.collect()**

**println("While Loop Constructs with Conditions (sorted by count in descending order):")**

**sortedWhileLoops.foreach { case (count, info) =>**

**println(s"$info occurred $count times")**

**}**

**// Calculate the sum of counts for `while` loop constructs**

**val totalCountForWhile = sortedWhileLoopCounts.map(\_.\_1).sum()**

**println(s"\nTotal count of 'while' loop constructs: $sortedWhileLoopCounts")**

This code is to perform finding the “while()” loop and the attributes inside the bracket. First, we define the “while()” pattern to find the attribute that link together with “while()”. Then we map the attributes using reduceByKey. After that, we print the result as descending order and the total extracted count.

**A white background with black text

Description automatically generated**

* **Table of extracted code and count**

**val totalCountForLoops = sortedForLoopCounts.map(\_.\_1).sum**

**val totalCountForswitch = sortedSwitchStatementCounts.map(\_.\_1).sum**

**val totalCountForcase = sortedCaseClauseCounts.map(\_.\_1).sum**

**// Calculate the total count for `if` constructs**

**val totalCountForIf = sortedIfStatements.map(\_.\_1).sum**

**// Calculate the total count for `else` constructs**

**val totalCountForElse = sortedElseStatements.map(\_.\_2).sum**

**val totalCountForWhile = sortedWhileLoopCounts.map(\_.\_1).sum**

**val totalCountClassAttributes = sortedClassCounts.foldLeft(0)((sum, tuple) => sum + tuple.\_1)**

**val totalCountFunctions = sortedFunctionCounts.map(\_.\_1).sum**

**val totalDivCount = extractedDivs.map(\_.\_2).reduce(\_ + \_)**

**val totalAttributeCount = attributeCountsRDD.map(\_.\_2).reduce(\_ + \_)**

**// Create an HTML table**

**val htmlTable = s"""**

**|<input type="text" id="keywordInput" placeholder="Search for a keyword..">**

**|<button id="searchButton">Search</button>**

**|<script defer>**

**|function searchKeyword() {**

**|  var input, filter, table, tr, td, i, txtValue;**

**|  input = document.getElementById("keywordInput");**

**|  filter = input.value.toUpperCase();**

**|  table = document.getElementById("dataTable");**

**|**

**|  // Check if the table exists**

**|  if (table) {**

**|    tr = table.getElementsByTagName("tr");**

**|    for (i = 0; i < tr.length; i++) {**

**|      td = tr[i].getElementsByTagName("td")[0];**

**|      if (td) {**

**|        txtValue = td.textContent || td.innerText;**

**|        if (txtValue.toUpperCase().indexOf(filter) > -1) {**

**|          tr[i].style.display = "";**

**|        } else {**

**|          tr[i].style.display = "none";**

**|        }**

**|      }**

**|    }**

**|  }**

**|}**

**|**

**|function searchButtonClick() {**

**|  searchKeyword();**

**|  var filteredData = [];**

**|  var table = document.getElementById("dataTable");**

**|  var rows = table.getElementsByTagName("tr");**

**|  for (var i = 0; i < rows.length; i++) {**

**|    var cells = rows[i].getElementsByTagName("td");**

**|    var rowData = [];**

**|    for (var j = 0; j < cells.length; j++) {**

**|      rowData.push(cells[j].innerText);**

**|    }**

**|    filteredData.push(rowData);**

**|  }**

**|  fetch('/submitFilteredData', {**

**|    method: 'POST',**

**|    body: JSON.stringify(filteredData),**

**|    headers: {**

**|      'Content-Type': 'application/json'**

**|    }**

**|  });**

**|}**

**|**

**|document.getElementById("searchButton").addEventListener("click", searchButtonClick);**

**|</script>**

**|<table border='1' id="dataTable">**

**|  <tr>**

**|    <th>Statement Type</th>**

**|    <th>Total Count</th>**

**|    <th>Code Link</th>**

**|  </tr>**

**|  <tr>**

**|    <td>For loop</td>**

**|    <td>$totalCountForLoops</td>**

**|    <td><a href='#code\_link'>Link to Code</a></td>**

**|  </tr>**

**|  <tr>**

**|    <td>Switch</td>**

**|    <td>$totalCountForswitch</td>**

**|    <td><a href='#code\_link'>Link to Code</a></td>**

**|  </tr>**

**|  <tr>**

**|    <td>Case</td>**

**|    <td>$totalCountForcase</td>**

**|    <td><a href='#code\_link'>Link to Code</a></td>**

**|  </tr>**

**|  <tr>**

**|    <td>if</td>**

**|    <td>$totalCountForIf</td>**

**|    <td><a href='#code\_link'>Link to Code</a></td>**

**|  </tr>**

**|  <tr>**

**|    <td>else</td>**

**|    <td>$totalCountForElse</td>**

**|    <td><a href='#code\_link'>Link to Code</a></td>**

**|  </tr>**

**|  <tr>**

**|    <td>While loop</td>**

**|    <td>$totalCountForWhile</td>**

**|    <td><a href='#code\_link'>Link to Code</a></td>**

**|  </tr>**

**|  <tr>**

**|    <td>Class</td>**

**|    <td>$totalCountClassAttributes</td>**

**|    <td><a href='#code\_link'>Link to Code</a></td>**

**|  </tr>**

**|  <tr>**

**|    <td>Function()</td>**

**|    <td>$totalCountFunctions</td>**

**|    <td><a href='#code\_link'>Link to Code</a></td>**

**|  </tr>**

**|  <tr>**

**|    <td>Div</td>**

**|    <td>$totalDivCount</td>**

**|    <td><a href='#code\_link'>Link to Code</a></td>**

**|  </tr>**

**|  <tr>**

**|    <td>Attribute</td>**

**|    <td>$totalAttributeCount</td>**

**|    <td><a href='#code\_link'>Link to Code</a></td>**

**|  </tr>**

**|</table>**

**""".stripMargin**

**displayHTML(htmlTable)**

This code is to perform finding the total number of counts for each extracted class by using html table. The code link which can be clicked and redirected to the function of the code. It can directly search the keyword to find the extracted name and the total count.

**A screenshot of a search engine

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**4. Finding relationships between modules, classes, functions and data dependencies.**

* **Scala code**

**import scala.io.Source**

**import java.io.PrintWriter**

**val filePaths = "C:\\Users\\user\\newdoctorpatientportal-master"**

**val fileContents = Source.fromFile(filePaths).getLines().toList**

**// Function to extract dependencies from different components**

**def extractDependenciesFromLine(line: String): List[String] = {**

**val modulePattern = """module\s+(\w+)""".r**

**val classPattern = """class\s+(\w+)""".r**

**val functionPattern = """def\s+(\w+)""".r**

**val dataPattern = """val\s+(\w+)""".r**

**val moduleDependencies = modulePattern.findAllIn(line).toList**

**val classDependencies = classPattern.findAllIn(line).toList**

**val functionDependencies = functionPattern.findAllIn(line).toList**

**val dataDependencies = dataPattern.findAllIn(line).toList**

**moduleDependencies ++ classDependencies ++ functionDependencies ++ dataDependencies**

**}**

**// Function to extract dependencies from a function options string**

**def extractDependenciesFromOptions(options: String): List[String] = {**

**options.split("\\s\*,\\s\*").toList**

**}**

**// Extract and process dependencies from input**

**println("Enter the code:")**

**val userInput = scala.io.StdIn.readLine()**

**val dependencies = extractDependenciesFromLine(userInput)**

**// Construct dependency edges in DOT format**

**val dependencyEdges = dependencies.filter(\_.nonEmpty).map( case (userInput, targets)=> s"""  "$userInput" -> "$target";""")**

**// Generate the complete DOT representation**

**val dotContent =**

**s"""|digraph G {**

**|${dependencyEdges.mkString("\n")}**

**|}**

**|""".stripMargin**

**println(dotContent)**

**// Define the file path where you want to save the .dot file**

**val filePath = "C:\\output.dot"**

**val writer = new PrintWriter(filePath)**

**writer.write(dotContent)**

**writer.close()**

After placing all the files on the correct path or amend the path based on your selection

scala -classpath C:\spark\jars\spark-core\_2.12-3.5.1.jar C:\Users\user\Desktop\Output.scala "C:\Users\user\newdoctorpatientportal-master"

The above code saves as Output.scala and runs on command prompt.

After input the component to search,the command prompt will prompt out the digraph. The following is all the relationships found on the file.

**digraph G {**

**"chkuserfun" -> "val";**

**"a" -> "b";**

**"a" -> "c";**

**"d" -> "a";**

**"e" -> "a";**

**"e" -> "c";**

**"e" -> "d";**

**"f" -> "a";**

**"g" -> "a";**

**"g" -> "c";**

**"h" -> "a";**

**"i" -> "a";**

**"i" -> "b";**

**"j" -> "a";**

**"j" -> "c";**

**"j" -> "d";**

**"k" -> "a";**

**"k" -> "b";**

**"k" -> "c";**

**"k" -> "d";**

**"m" -> "a";**

**"m" -> "b";**

**"Plugin" -> "option";**

**"Plugin" -> "option";**

**"Plugin" -> "option";**

**"getTargetFromTrigger" -> "$trigger";**

**"Plugin" -> "option";**

**"getParent" -> "$this";**

**"clearMenus" -> "e";**

**"Plugin" -> "option";**

**"Plugin" -> "option";**

**"Plugin" -> "\_relatedTarget";**

**"Plugin" -> "option";**

**"Plugin" -> "option";**

**"ScrollSpy" -> "element";**

**"ScrollSpy" -> "options";**

**"Plugin" -> "option";**

**"Plugin" -> "option";**

**"Plugin" -> "option";**

**"s" -> "a";**

**"fa" -> "a";**

**"fa" -> "b";**

**"fa" -> "d";**

**"fa" -> "e";**

**"b" -> "c";**

**"b" -> "e";**

**"ha" -> "a";**

**"ia" -> "a";**

**"ja" -> "a";**

**"ja" -> "b";**

**"ka" -> "a";**

**"ka" -> "b";**

**"la" -> "a";**

**"ma" -> "a";**

**"na" -> "a";**

**"oa" -> "a";**

**"qa" -> "a";**

**"ra" -> "a";**

**"ra" -> "b";**

**"ra" -> "c";**

**"sa" -> "a";**

**"ta" -> "a";**

**"ta" -> "b";**

**"ta" -> "c";**

**"ua" -> "a";**

**"ua" -> "b";**

**"ua" -> "c";**

**"ua" -> "d";**

**"ua" -> "e";**

**"va" -> "a";**

**"va" -> "b";**

**"va" -> "c";**

**"va" -> "d";**

**"va" -> "e";**

**"va" -> "f";**

**"wa" -> "a";**

**"xa" -> "a";**

**"xa" -> "b";**

**"z" -> "a";**

**"z" -> "b";**

**"z" -> "c";**

**"F" -> "a";**

**"F" -> "b";**

**"H" -> "a";**

**"d" -> "b";**

**"P" -> "a";**

**"P" -> "b";**

**"P" -> "c";**

**"Q" -> "a";**

**"R" -> "a";**

**"R" -> "b";**

**"R" -> "d";**

**"R" -> "e";**

**"S" -> "a";**

**"S" -> "b";**

**"S" -> "c";**

**"X" -> "a";**

**"X" -> "b";**

**"X" -> "c";**

**"X" -> "d";**

**"ca" -> "a";**

**"ea" -> "a";**

**"ea" -> "b";**

**"fa" -> "a";**

**"fa" -> "b";**

**"ia" -> "a";**

**"ja" -> "a";**

**"ja" -> "b";**

**"ja" -> "c";**

**"ja" -> "d";**

**"ja" -> "e";**

**"sa" -> "a";**

**"sa" -> "b";**

**"sa" -> "c";**

**"sa" -> "d";**

**"sa" -> "e";**

**"sa" -> "f";**

**"Ca" -> "a";**

**"Ca" -> "b";**

**"Da" -> "a";**

**"Ea" -> "a";**

**"Fa" -> "a";**

**"Fa" -> "b";**

**"Ga" -> "a";**

**"Ga" -> "b";**

**"Ha" -> "a";**

**"Ha" -> "b";**

**"Ha" -> "c";**

**"Ha" -> "d";**

**"Ia" -> "a";**

**"Ia" -> "b";**

**"Ia" -> "c";**

**"La" -> "a";**

**"La" -> "b";**

**"Ma" -> "a";**

**"Ua" -> "a";**

**"Ua" -> "b";**

**"bb" -> "a";**

**"cb" -> "a";**

**"cb" -> "b";**

**"db" -> "a";**

**"db" -> "b";**

**"db" -> "c";**

**"eb" -> "a";**

**"eb" -> "b";**

**"eb" -> "c";**

**"eb" -> "d";**

**"eb" -> "e";**

**"fb" -> "b";**

**"fb" -> "c";**

**"fb" -> "e";**

**"gb" -> "a";**

**"gb" -> "b";**

**"gb" -> "c";**

**"gb" -> "d";**

**"gb" -> "e";**

**"mb" -> "a";**

**"mb" -> "b";**

**"nb" -> "a";**

**"nb" -> "b";**

**"nb" -> "c";**

**"ob" -> "a";**

**"ob" -> "b";**

**"ob" -> "c";**

**"pb" -> "a";**

**"pb" -> "b";**

**"qb" -> "a";**

**"qb" -> "b";**

**"qb" -> "c";**

**"Bb" -> "a";**

**"Sb" -> "a";**

**"Tb" -> "a";**

**"Tb" -> "b";**

**"Tb" -> "c";**

**"Tb" -> "d";**

**"g" -> "h";**

**"Ub" -> "a";**

**"Ub" -> "b";**

**"Vb" -> "a";**

**"Vb" -> "b";**

**"Vb" -> "c";**

**"Wb" -> "a";**

**"Wb" -> "b";**

**"Wb" -> "c";**

**"Wb" -> "d";**

**"y" -> "b";**

**"y" -> "c";**

**"y" -> "d";**

**"y" -> "e";**

**"Xb" -> "a";**

**"Yb" -> "a";**

**"cc" -> "a";**

**"cc" -> "b";**

**"cc" -> "c";**

**"cc" -> "d";**

**"lc" -> "a";**

**"b" -> "b";**

**"b" -> "b";**

**"b" -> "b";**

**"b" -> "b";**

**"c" -> "b";**

**"b" -> "b";**

**"c" -> "c";**

**"d" -> "b";**

**"b" -> "b";**

**"b" -> "d";**

**"b" -> "b";**

**"b" -> "b";**

**"b" -> "c";**

**"b" -> "d";**

**"c" -> "c";**

**"b" -> "b";**

**"b" -> "b";**

**"letters" -> "input";**

**"letters" -> "input";**

**"letters1" -> "input";**

**"getId" -> "val";**

**"getIdd" -> "val";**

**"letters" -> "input";**

**"j" -> "e";**

**"j" -> "b";**

**"h" -> "e";**

**"calcOffset" -> "event";**

**"arguments\_callee1" -> "v";**

**"arguments\_callee4" -> "v";**

**"line\_time" -> "h";**

**"line\_time" -> "m";**

**"setMask" -> "options";**

**"HighlightedDate" -> "date";**

**"HighlightedDate" -> "desc";**

**"HighlightedDate" -> "style";**

**"handler" -> "event";**

**"shouldAdjustOldDeltas" -> "orgEvent";**

**"shouldAdjustOldDeltas" -> "absDelta";**

**"K" -> "N";**

**"K" -> "O";**

**"z" -> "E";**

**"z" -> "F";**

**"j" -> "E";**

**"j" -> "F";**

**"P" -> "U";**

**"P" -> "Z";**

**"P" -> "Y";**

**"P" -> "ad";**

**"P" -> "aa";**

**"P" -> "ac";**

**"S" -> "U";**

**"S" -> "Z";**

**"S" -> "Y";**

**"S" -> "ad";**

**"S" -> "aa";**

**"S" -> "ac";**

**"c" -> "H";**

**"i" -> "F";**

**"i" -> "E";**

**"H" -> "I";**

**"H" -> "J";**

**"t" -> "F";**

**"t" -> "E";**

**"F" -> "J";**

**"a" -> "g";**

**"a" -> "h";**

**"a" -> "f";**

**"a" -> "f";**

**"a" -> "g";**

**"o" -> "u";**

**"a" -> "o";**

**"a" -> "m";**

**"l" -> "t";**

**"a" -> "g";**

**"a" -> "k";**

**"c" -> "h";**

**"c" -> "g";**

**"\_superApply" -> "args";**

**"processClassString" -> "classes";**

**"processClassString" -> "checkOption";**

**"getOffsets" -> "offsets";**

**"getOffsets" -> "width";**

**"getOffsets" -> "height";**

**"parseCss" -> "element";**

**"parseCss" -> "property";**

**"getDimensions" -> "elem";**

**"clamp" -> "value";**

**"clamp" -> "prop";**

**"clamp" -> "allowEmpty";**

**"stringParse" -> "string";**

**"hue2rgb" -> "p";**

**"hue2rgb" -> "q";**

**"hue2rgb" -> "h";**

**"getElementStyles" -> "elem";**

**"styleDifference" -> "oldStyle";**

**"styleDifference" -> "newStyle";**

**"\_normalizeArguments" -> "effect";**

**"\_normalizeArguments" -> "options";**

**"\_normalizeArguments" -> "speed";**

**"\_normalizeArguments" -> "callback";**

**"standardAnimationOption" -> "option";**

**"run" -> "next";**

**"parseClip" -> "str";**

**"parseClip" -> "element";**

**"visible" -> "element";**

**"reduce" -> "elem";**

**"reduce" -> "size";**

**"reduce" -> "border";**

**"reduce" -> "margin";**

**"datepicker\_getZindex" -> "elem";**

**"datepicker\_bindHover" -> "dpDiv";**

**"datepicker\_extendRemove" -> "target";**

**"datepicker\_extendRemove" -> "props";**

**"filteredUi" -> "ui";**

**"filteredUi" -> "ui";**

**"isOverAxis" -> "x";**

**"isOverAxis" -> "reference";**

**"isOverAxis" -> "size";**

**"delayEvent" -> "type";**

**"delayEvent" -> "instance";**

**"delayEvent" -> "container";**

**"spinnerModifer" -> "fn";**

**"position" -> "event";**

**"letters" -> "input";**

**"letters" -> "input";**

**"letters1" -> "input";**

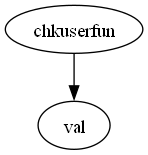
**"getId" -> "val";**

**"getIdd" -> "val";**

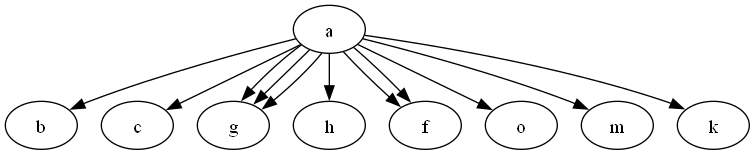
**"generateRandomString" -> "$length = 12";**

**The code will save the input graph into the output.dot file. We will need to run the dot file in command prompt by dot -Tpng C:\output.dot -o output.png.**

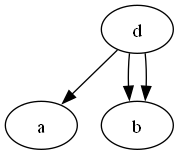
**The following are all the relationship dependency graphs found on our file.**

****

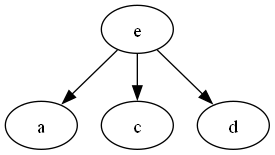
**function “chkuserfun”**

****

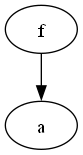
**function “a”**

****

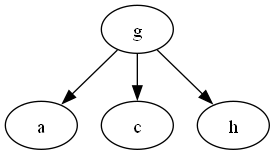
**function “d”**

****

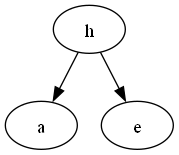
**function “e”**

****

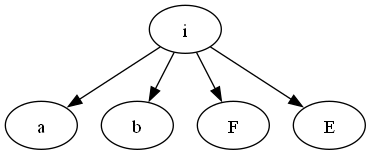
**function “f”**

****

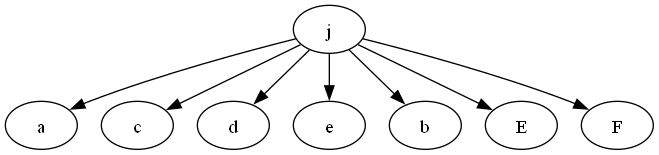
**function “g”**

****

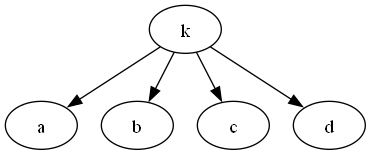
**function “h”**

****

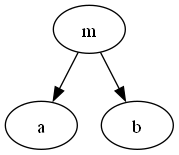
**function “i”**

****

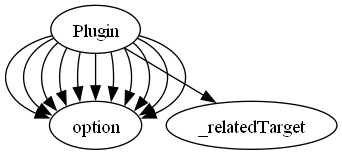
**function “j”**

****

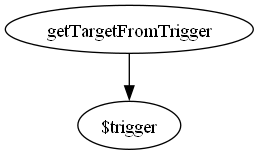
**function “k”**

****

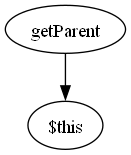
**function “m”**

****

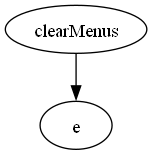
**function “Plugin”**

****

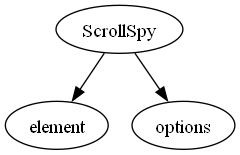
**function “getTargetFromTrigger”**

****

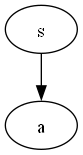
**function “getParent”**

****

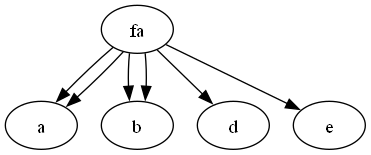
**function “clearMenus”**

****

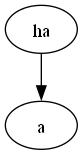
**function “ScrollSpy”**

****

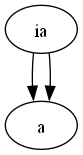
**function “s”**

****

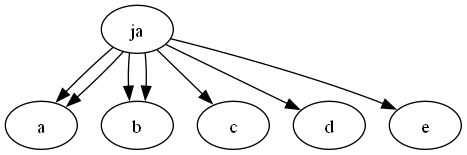
**function “fa”**

****

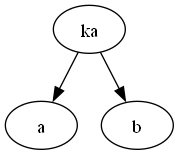
**function “ha”**

****

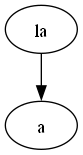
**function “ia”**

****

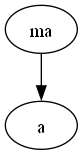
**function “ja”**

****

**function “ka”**

****

**function “la”**

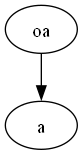
****

**function “ma”**

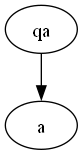
**A diagram of a diagram

Description automatically generated**

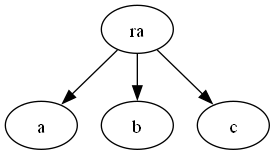
**function “na”**

****

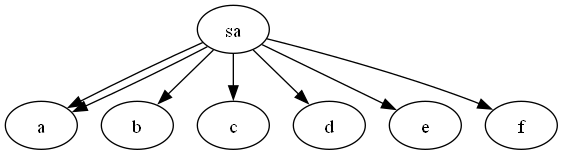
**function “oa”**

****

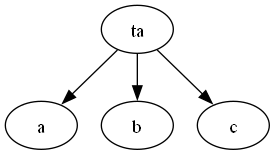
**function “qa”**

****

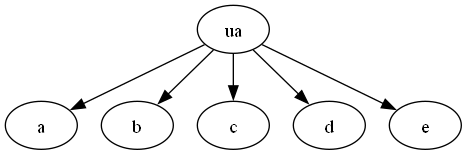
**function “ra”**

****

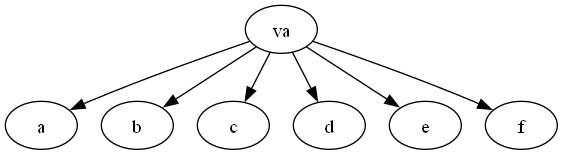
**function “sa”**

****

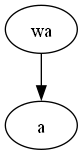
**function “ta”**

****

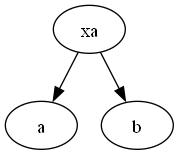
**function “ua”**

****

**function “va”**

****

**function “wa”**

****

**function “xa”**

**A diagram of a diagram

Description automatically generated**

**function “z”**

**A diagram of a diagram

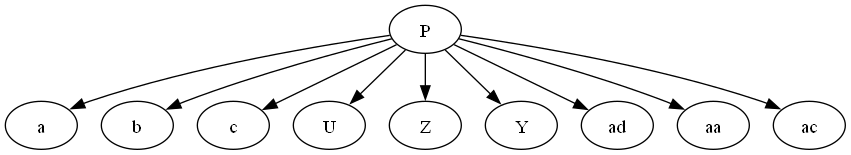
Description automatically generated**

**function “F”**

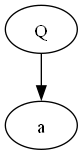
**A diagram of a structure

Description automatically generated**

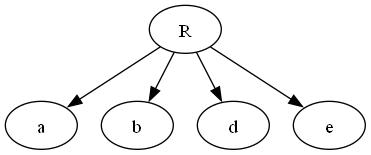
**function “H”**

****

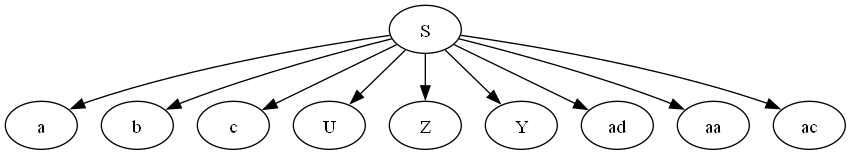
**function “P”**

****

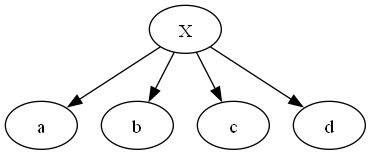
**function “Q”**

****

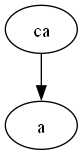
**function “R”**

****

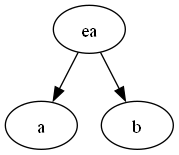
**function “S”**

****

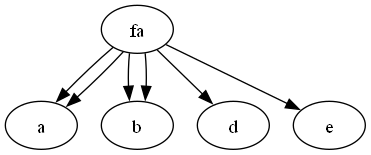
**function “X”**

****

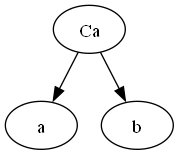
**function “ca”**

****

**function “ea”**

****

**function “fa”**

****

**function “Ca”**

**A diagram of a diagram

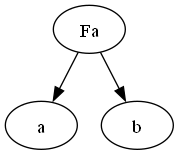
Description automatically generated**

**function “Da”**

**A diagram of a diagram

Description automatically generated**

**function “Ea”**

****

**function “Fa”**

**A diagram of a diagram

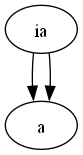
Description automatically generated**

**function “Ga”**

**A diagram of a structure

Description automatically generated**

**function “Ha”**

****

**function “ia”**

**A diagram of a diagram

Description automatically generated**

**function “Ia”**

**A diagram of a diagram

Description automatically generated**

**function “La”**

**A diagram of a diagram

Description automatically generated**

**function “Ua”**

**A diagram of a diagram

Description automatically generated**

**function “bb”**

**A diagram of a diagram

Description automatically generated**

**function “cb”**

**A diagram of a diagram

Description automatically generated**

**function “db”**

**A diagram of a diagram

Description automatically generated**

**function “eb”**

**A diagram of a diagram

Description automatically generated**

**function “fb”**

**A diagram of a diagram

Description automatically generated**

**function “gb”**

**A diagram of a diagram

Description automatically generated**

**function “mb”**

**A diagram of a network

Description automatically generated**

**function “nb”**

**A diagram of a diagram

Description automatically generated**

**function “ob”**

**A diagram of a diagram

Description automatically generated**

**function “pb”**

**A diagram of a diagram

Description automatically generated**

**function “qb”**

**A diagram of a diagram

Description automatically generated**

**function “Bb”**

**A diagram of a diagram

Description automatically generated**

**function “Sb”**

**A diagram of a diagram

Description automatically generated**

**function “Tb”**

**A diagram of a diagram

Description automatically generated**

**function “Ub”**

**A diagram of a diagram

Description automatically generated**

**function “Vb”**

**A diagram of a diagram

Description automatically generated**

**function “Wb”**

**A diagram of a diagram

Description automatically generated**

**function “y”**

**A diagram of a diagram

Description automatically generated**

**function “Xb”**

**A diagram of a diagram

Description automatically generated**

**function “Yb”**

**A diagram of a diagram

Description automatically generated**

**function “cc”**

**A diagram of a diagram

Description automatically generated**

**function “lc”**

**A diagram of a diagram

Description automatically generated**

**function “na”**

**A diagram of a diagram

Description automatically generated**

**function “b”**

**A diagram of a diagram

Description automatically generated**

**function “c”**

**A diagram of a diagram

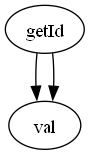
Description automatically generated**

**function “letters”**

**A diagram of a diagram

Description automatically generated**

**function “letters1”**

****

**function “getId”**

**A diagram of a flowchart

Description automatically generated**

**function “getIdd”**

**A diagram of a event

Description automatically generated**

**function “calcOffset”**

**A diagram of a diagram

Description automatically generated**

**function “arguments\_callee1”**

**A diagram of a diagram

Description automatically generated**

**function “arguments\_callee4”**

**A diagram of a line

Description automatically generated**

**function “line\_time”**

**A diagram of a process

Description automatically generated**

**function “setMask”**

**A diagram of a diagram

Description automatically generated**

**function “HighlightedDate"**

**A diagram of a event

Description automatically generated**

**function “handler"**

**A diagram of an event

Description automatically generated**

**function “shouldAdjustOldDeltas"**

**A diagram of a network

Description automatically generated**

**function “K”**

**A diagram of a diagram

Description automatically generated**

**function “t”**

**A diagram of a diagram

Description automatically generated**

**function “F”**

**A diagram of a diagram

Description automatically generated**

**function “l”**

**A diagram of a diagram

Description automatically generated**

**function “o”**

**A diagram of an args

Description automatically generated**

**function “\_superApply”**

**A diagram of a process

Description automatically generated**

**function “processClassString”**

**A diagram of a diagram

Description automatically generated**

**function “getOffsets”**

**A diagram of a diagram

Description automatically generated**

**function “parseCss”**

**A diagram of a diagram

Description automatically generated**

**function “getDimensions”**

**A diagram of a diagram

Description automatically generated**

**function “clamp”**

**A diagram of string parse

Description automatically generated**

**function “stringParse”**

**A diagram of a structure

Description automatically generated**

**function “hue2rgb”**

**A diagram of a diagram

Description automatically generated**

**function “getElementStyles”**

**A diagram of a style and news

Description automatically generated**

**function “styleDifference”**

**A diagram of a diagram

Description automatically generated**

**function “\_normalizeArguments”**

**A diagram of a diagram

Description automatically generated**

**function “standardAnimationOption”**

**A diagram of a diagram

Description automatically generated**

**function “run”**

**A diagram of a diagram

Description automatically generated**

**function “parseClip”**

**A diagram of a diagram

Description automatically generated**

**function “visible"**

**A diagram of a diagram

Description automatically generated**

**function “reduce”**

**A diagram of a diagram

Description automatically generated**

**function “datepicker\_getZindex”**

**A diagram of a server

Description automatically generated**

**function “datepicker\_bindHover”**

**A diagram of a diagram

Description automatically generated**

**function “datepicker\_extendRemove”**

**A diagram of a computer network

Description automatically generated**

**function “filteredUi”**

**A diagram of a reference

Description automatically generated**

**function “isOverAxis”**

**A diagram of a process

Description automatically generated**

**function “delayEvent”**

**A diagram of a spinner modifer

Description automatically generated**

**function “spinnerModifer”**

**A diagram of a diagram

Description automatically generated**

**function “position”**

**A diagram of a diagram

Description automatically generated**

**function “generateRandomString”**