Comenzado el	jueves, 4 de abril de 2024, 13:17
Estado	Finalizado
Finalizado en	jueves, 4 de abril de 2024, 15:53
Tiempo empleado	2 horas 35 minutos
Puntos	1,00/1,00
Calificación	**50,00** de 50,00 (**100**%)

PREGUNTA 1

Finalizado

Se puntúa 1,00 sobre 1,00

ENUNCIADO

Los algoritmos de búsqueda son esenciales para desarrollo de las ciencias de la computación. En este ejercicio los estudiantes crearán una solución web que explora dos algortimos de búsqueda: la búsqueda lineal y la búsqueda binaria.

Diseñe, construya y despliegue un aplicación web para investigar los dos algoritmos de búsqueda. El programa debe estar desplegado en tres máquinas virtuales de EC2 de AWS como se describe abajo. Las tecnologías usadas en la solución deben ser maven, git, github, sparkjava, html5, y js. No use liberías adicionales.

NO COPIE CÓDIGO EXISTENTE NI SUYO NI DE OTRAS FUENTES. TODO EL ENTREGABLE DEBE SER ORIGINAL CREADO EN LA CLASE POR USTED.

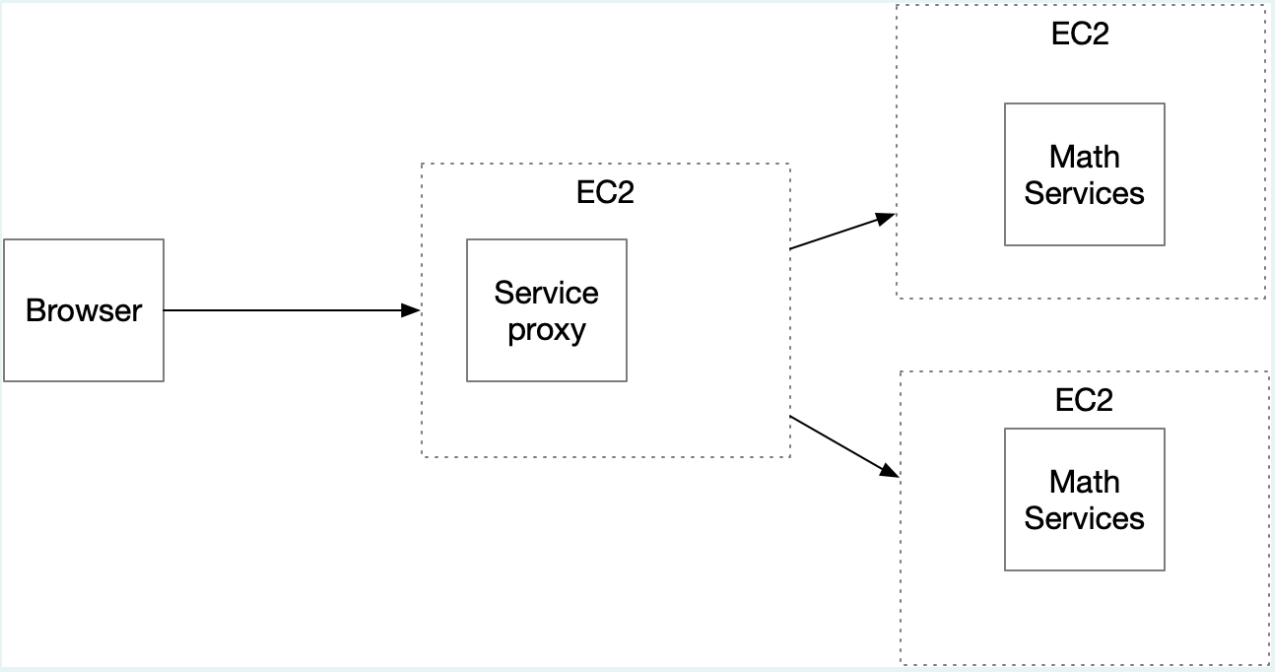
Solo use la documentación autorizada por el profesor.

DOCUMENTACIÓN AUTORIZADA:

- https://docs.oracle.com/javase/8/docs/api/
- https://sparkjava.com/
- https://docs.aws.amazon.com/corretto/latest/corretto-8-ug/amazon-linux-install.html
- https://mvnrepository.com/
- AWS Academy
- AWS

PROBLEMA:

Diseñe un prototipo de sistema de microservicios que tenga un servicio (En la figura se representa con el nombre Math Services) para computar las funciones de ordenamiento. El servicio de las funciones de ordenamiento debe estar desplegado en al menos dos instancias virtuales de EC2. Adicionalmente, debe implementar un service proxy que reciba las solicitudes de llamado desde los clientes y se las delega a las dos instancias del servicio de ordenamiento usando un algoritmo de **round-robin**. El proxy deberá estar desplegado en otra máquina EC2. Asegúrese que se pueden configurar las direcciones y puertos de las instancias del servicio en el proxy usando variables de entorno del sistema operativo. Finalmente, construya un cliente Web mínimo con un formulario que reciba el valor y de manera **asíncrona** invoke el servicio en el PROXY. Puede hacer un formulario para cada una de las funciones. El cliente debe ser escrito en HTML y JS.



SOBRE LAS FUNCIONES DE ORDENAMIENTO:

Sus servicios deben incluir dos funciones.

- Uno recibe una lista de cadenas y un valor a buscar e implementa la búsqueda lineal : linearSearch(lista, valor) retorna un json con el índice de la primera aparición del valor o con -1 si no encuentra el valor
- Uno recibe una lista ordenada de cadenas y un valor a buscar e implementa la búsqueda binaria de manera **recursiva** : binarySearch(n), retorna un json con el índice de la primera aparición del valor o con -1 si no encuentra el valor.

PARA AMBAS IMPLEMENTACIONES ESCRIBA EL ALGORITMO. Usted debe implementar las dos funciones, no debe usar funciones de una librería o del API (si ya existen).

La búsqueda lineal y la búsqueda binaria son dos algoritmos fundamentales utilizados para encontrar un elemento específico en un conjunto de datos. Aquí te describo en detalle cómo funcionan estos dos algoritmos:

BÚSQUEDA LINEAL

La búsqueda lineal, también conocida como búsqueda secuencial, es un método simple y directo para encontrar un elemento en un conjunto de datos. Funciona de la siguiente manera:

1. **Inicio:** Comenzar desde el primer elemento del conjunto de datos.
2. **Comparación:** Comparar cada elemento con el valor buscado.
3. **Resultado:**
    - Si el elemento actual es igual al valor buscado, se retorna la posición de ese elemento, indicando así que el elemento ha sido encontrado.
    - Si el elemento actual no es igual al valor buscado, se pasa al siguiente elemento.
4. **Finalización:** Este proceso continúa hasta que se encuentra el elemento o se ha recorrido todo el conjunto de datos.
5. **No encontrado:** Si se llega al final del conjunto sin encontrar el valor, se indica que el elemento no está presente.

La búsqueda lineal no requiere que los datos estén ordenados y es efectiva para conjuntos de datos pequeños, pero su eficiencia disminuye a medida que el tamaño del conjunto de datos aumenta, ya que en el peor caso, se deben comparar todos los elementos.

BÚSQUEDA BINARIA

La búsqueda binaria es un método más eficiente que la búsqueda lineal, pero requiere que el conjunto de datos esté ordenado previamente. Su proceso se describe de la siguiente manera:

1. **Inicio:** Determinar los índices de inicio y fin del conjunto de datos, que inicialmente son el primer y último elemento, respectivamente.
2. **División:** Calcular el índice medio del conjunto de datos actual y comparar el elemento en esta posición con el valor buscado.
3. **Comparación:**
    - Si el elemento medio es igual al valor buscado, se retorna la posición de este elemento, indicando que se ha encontrado.
    - Si el elemento medio es mayor que el valor buscado, se descarta la mitad superior del conjunto y se repite el proceso con la mitad inferior.
    - Si el elemento medio es menor que el valor buscado, se descarta la mitad inferior del conjunto y se repite el proceso con la mitad superior.
4. **Iteración:** El proceso se repite, reduciendo a la mitad el tamaño del conjunto de datos en cada paso.
5. **Finalización:** Este proceso continúa hasta que se encuentra el valor o el subconjunto se reduce a cero.

La búsqueda binaria es muy eficiente en conjuntos de datos grandes, ya que reduce significativamente el número de comparaciones necesarias en comparación con la búsqueda lineal, logrando una complejidad temporal de «math xmlns=¨http://www.w3.org/1998/Math/MathML¨»«semantics»«mrow»«mi»O«/mi»«mo stretchy=¨false¨»(«/mo»«mi»log«/mi»«mo»§#8289;«/mo»«mi»n«/mi»«mo stretchy=¨false¨»)«/mo»«/mrow»«/semantics»«/math»O(logn), donde «math xmlns=¨http://www.w3.org/1998/Math/MathML¨»«semantics»«mrow»«mi»n«/mi»«/mrow»«/semantics»«/math»n es el número de elementos en el conjunto de datos.

DETALLES ADICIONALES DE LA ARQUITECTURA Y DEL API

Implemente los servicios para responder al método de solicitud HTTP GET. Deben usar el nombre de la función especificado los parámetros deben ser pasados en variables de query con nombres "list" y "value".

El proxy debe delegar el llamado a los servicios de backend. El proxy y los servicios se deben implementar en Java usando Spark.

Ejemplo 1 de un llamado:

https://amazonxxx.x.xxx.x.xxx:{port}/linearsearch?list=10,20,13,40,60&value=13

Salida. El formato de la salida y la respuesta debe ser un JSON con el siguiente formato

```
{
 "operation": "linearSearch",
 "inputlist": "10,20,13,40,60",
 "value": "13"
 "output":  "2"
}
```

Ejemplo 2 de un llamado:

https://amazonxxx.x.xxx.x.xxx:{port}/linearsearch?list=10,20,13,40,60&value=99

Salida. El formato de la salida y la respuesta debe ser un JSON con el siguiente formato

```
{
 "operation": "linearSearch",
 "inputlist": "10,20,13,40,60",
 "value": "99"
 "output":  "-1"
}
```

Ejemplo 3 de un llamado:

https://amazonxxx.x.xxx.x.xxx:{port}/binarysearch?list=10,20,13,40,60&value=13

Salida. El formato de la salida y la respuesta debe ser un JSON con el siguiente formato

```
{
 "operation": "binarySearch",
 "inputlist": "10,20,13,40,60",
 "value": "13"
 "output":  "2"
}
```

Entregable:

1. Proyecto actualizado en github (un solo repositorio)

2. Descripción del proyecto enel README con pantallazos que muestren el funcionamiento.

3. Descripción de como correrlo en EC2.

4. Video de menos de un minuto del funcionamiento (lo puede tomar con el celular una vez funcione)

EN EL CAMPO DE TEXTO ESCRIBA LA DIRECIÓN DE SU REPOSITORIO GITHUB.

Ayuda

1. Para invocar un servicios get desde java puede hacerlo de manera fácil con:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;

public class HttpConnectionExample {

    private static final String USER_AGENT = "Mozilla/5.0";
    private static final String GET_URL = "https://www.alphavantage.co/query?
function=TIME_SERIES_DAILY&symbol=fb&apikey=Q1QZFVJQ21K7C6XM";

    public static void main(String[] args) throws IOException {

        URL obj = new URL(GET_URL);
        HttpURLConnection con = (HttpURLConnection) obj.openConnection();
        con.setRequestMethod("GET");
        con.setRequestProperty("User-Agent", USER_AGENT);

        //The following invocation perform the connection implicitly before getting the code
        int responseCode = con.getResponseCode();
        System.out.println("GET Response Code :: " + responseCode);

        if (responseCode == HttpURLConnection.HTTP_OK) { // success
            BufferedReader in = new BufferedReader(new InputStreamReader(
                    con.getInputStream()));
            String inputLine;
            StringBuffer response = new StringBuffer();

            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
            in.close();

            // print result
            System.out.println(response.toString());
        } else {
            System.out.println("GET request not worked");
        }
        System.out.println("GET DONE");
    }

}
```

2. Para invocar servicios rest de forma asíncrona desde un cliente JS

```
<!DOCTYPE html>
<html>
    <head>
        <title>Form Example</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h1>Form with GET</h1>
        <form action="/hello">
            <label for="name">Name:</label><br>
            <input type="text" id="name" name="name" value="John"><br><br>
            <input type="button" value="Submit" onclick="loadGetMsg()">
        </form>
        <div id="getrespmsg"></div>

        <script>
            function loadGetMsg() {
                let nameVar = document.getElementById("name").value;
                const xhttp = new XMLHttpRequest();
                xhttp.onload = function() {
                    document.getElementById("getrespmsg").innerHTML =
                    this.responseText;
                }
                xhttp.open("GET", "/hello?name="+nameVar);
                xhttp.send();
            }
        </script>

        <h1>Form with POST</h1>
        <form action="/hellopost">
            <label for="postname">Name:</label><br>
            <input type="text" id="postname" name="name" value="John"><br><br>
            <input type="button" value="Submit" onclick="loadPostMsg(postname)">
        </form>

        <div id="postrespmsg"></div>

        <script>
            function loadPostMsg(name){
                let url = "/hellopost?name=" + name.value;

                fetch (url, {method: 'POST'})
                    .then(x => x.text())
                    .then(y => document.getElementById("postrespmsg").innerHTML = y);
            }
        </script>
    </body>
</html>
```

3. Aplicación Spark mínima

```
public class SparkWebServer {

    public static void main(String... args){
          port(getPort());
          get("hello", (req,res) -> "Hello Docker!");
    }

    private static int getPort() {
        if (System.getenv("PORT") != null) {
            return Integer.parseInt(System.getenv("PORT"));
        }
        return 4567;
    }

}
```

4. Build con dependencias

```
<!-- build configuration -->
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-dependency-plugin</artifactId>
                <version>3.0.1</version>
                <executions>
                    <execution>
                        <id>copy-dependencies</id>
                        <phase>prepare-package</phase>
                        <goals>
                            <goal>copy-dependencies</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                        <configuration>
                            <archive>
                                <manifest>
                                    <mainClass>
                                        co.edu.escuelaing.securityprimerliveg2.HelloService
                                    </mainClass>
                                </manifest>
                            </archive>
                            <descriptorRefs>
                                <descriptorRef>jar-with-dependencies</descriptorRef>
                            </descriptorRefs>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

URL del proyecto en Github: https://github.com/ELS4NTA/AREP-PARCIAL-T2.git

Comentario:

Rúbrica Parcial 2 AREP	Referencia	Calificación Commit 1	Calificación Commit 2	
1. Proyecto actualizado en github y cumple requerimientos de calidad del repositorio		4	4	4
2. Implementación de reto algorítmico		7	7	7
3. Implementación del cliente asíncrono		3	3	3
4. Implementación del servicio REST retornando Json		3	3	3
4.1 Implementa proxy con Roundrobin		2	2	2
5. Cumple con la arquitectura solicitada (Múltiples instancias)		2	2	2
6. Corre de manera local		2	2	2
7. Funciona en AWS y está documentado en video.		4	4	4
Total		27	27	27
Procentaje		1	1	1
Nota		5	5	5