Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación
Área en Ingeniería en Computadores

Primer Proyecto Programado
SpiderSearch Engine: Stage 1

Profesor:
Nereo Campos Araya

Elaborado por:
Gerald Francisco Mora Mora, 2014064955.
Jairo Daniel Ortega Calderón, 2014043224.

Curso:
CE-1103

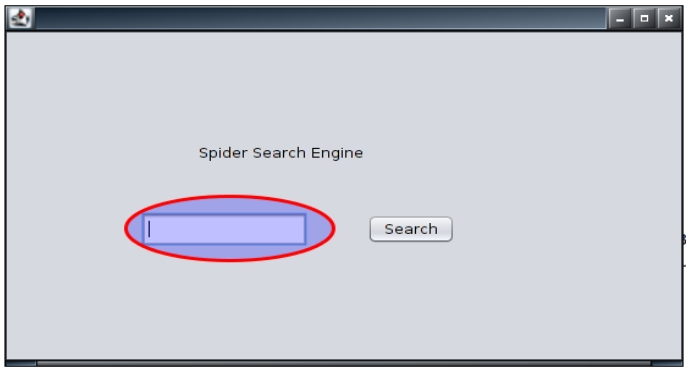
Grupo 1

Cartago, I Semestre 2015.

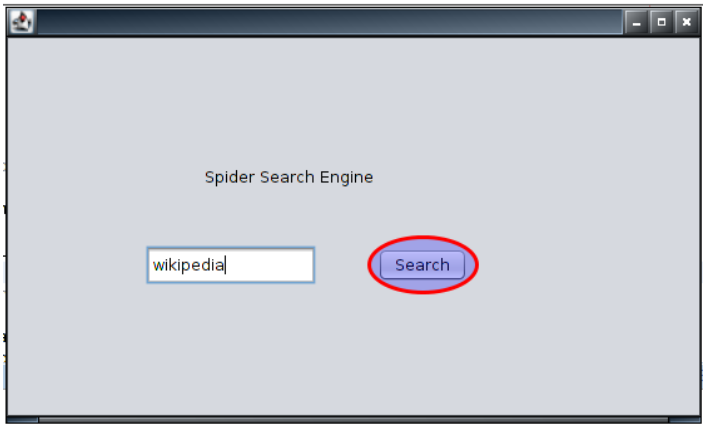
Índice

Manual de Usuario.

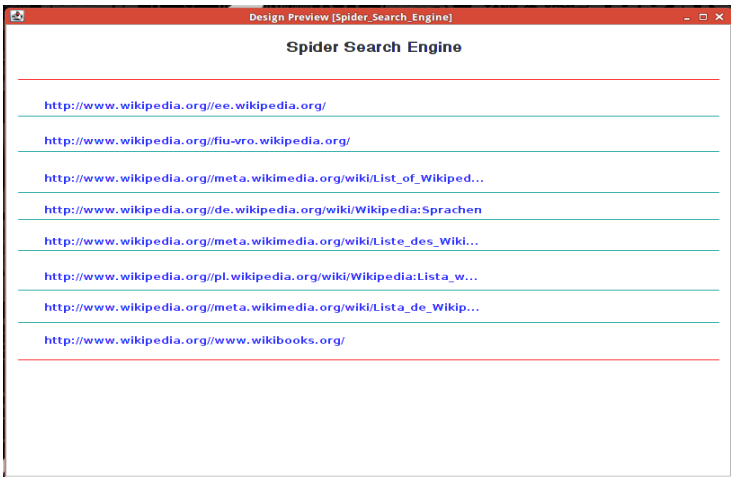
1. Proceda a escribir lo que desea en el espacio correspondiente.



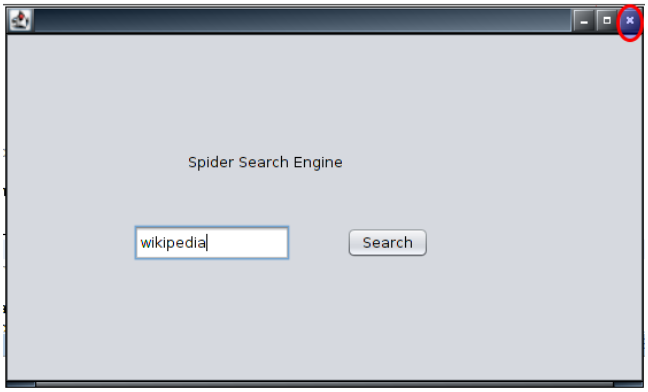
2. Posterior a la escritura de lo que desea buscar, pulse el botón “Search”.



3. Espere un momento mientras se realiza la búsqueda deseada y seleccione la página de su preferencia.



4. Si desea cerrar el programa, únicamente presione en el botón de la “x” (parte superior derecha) y automáticamente se le cerrara.



Descripción de las bibliotecas y funciones utilizadas.

Bibliotecas

- **java.io.FileNotFoundException:** Para capturar las excepciones que se producen al abrir un archivo por medio de una ruta especificada.
- **java.io.IOException:** Utilizada para hacer la captura de las excepciones que ocurren a la hora de hacer operaciones de entrada y salida.
- **javax.xml.parsers.ParserConfigurationException:** Verifica la escritura del archivo XML para localizar ciertas excepciones que no corresponden a un documento XML bien escrito.
- **org.xml.sax.SAXException:** Tiene la función de verificar la correcta escritura de un XML. Detecta excepciones por errores dentro del archivo XML.
- **java.io.BufferedReader:** Lee el texto de un stream de caracteres y prevé una lectura eficiente de los caracteres.
- **java.io.InputStream:** Es una superclase de todas las clases que presentan un flujo de entrada de bytes.
- **java.io.InputStreamReader:** Es un puente de flujo de datos en un stream. Lee caracteres de bytes y los de codifica en caracteres específicos.
- **java.net.MalformedURLException:** Se utiliza para alertar al programador cuando un enlace URL esta mal escrito o no cumple con el formato necesario.
- **java.net.URL:** Esta biblioteca es especial para la manipulación de URL's en Java
- **java.net.URLConnection:** Representa los diversos tipos de conexión de los URL's, como HttpURLConnection (Utilizado en este proyecto) o JarURLConnection(Representa a un archivo JAR).
- j**ava.util.regex:** Matcher: Es el motor que interpreta al patrón. Realiza operaciones sobre las cadenas de entrada.
- **java.util.regex.Pattern:** Es una representación compilada de una expresión regular.
- **javax.xml.parsers:** DocumentBuilder: Obtiene la API de las instancias de DOM del documento XML.
- j**avax.xml.parsers:** DocumentBuilderFactory: Permite a las aplicaciones obtener un analizador que produce árboles de objetos DOM de documentos XML.
- **org.w3c.dom:** Document: Crea un Documento sobre el cual se escribirá el texto que posee el archivo XML.
- **org.w3c.dom.Element:** Es para obtener el elemento raíz del documento XML.
- **org.w3c.dom.Node:** Nodo creado para mantener los datos encontrados en el Elemento raíz.

- **org.w3c.dom.NodeList:** Guarda los elemento de la raíz.
- **org.xml.sax.SAXException:** Detecta como excepción la introducción de un archivo el cual no es del tipo XML.

Funciones

- **openConnection():** Función que abre conexión con el URL solicitado.
- **getInputStream():** Obtiene datos que entran codificados a la aplicación cuando se utiliza la biblioteca InputStream.
- **ReadLine():** Readline se utiliza para leer linea por linea una entrada de datos, ya sea por entrada de datos, o por lectura de un documento o archivo, Se aplica junto con la biblioteca BurreferedReader.
- **Compile().** Se utiliza para el métodoo Pattern de REGEX, Este toma un string con caracteres y los compila para ser utilizados como patrón.
- **Matcher():** Se utiliza para el método Matcher de REGEX, Este toma el patrón, lo interpreta y realiza acciones sobre el el.
- **NewDocumentBuilder():** Toma un documento en blanco y lo prepara para agregar los datos.
- **Parse():** Toma un archivo y lo transforma al formato adecuado para ser agregado al nuevo documento.
- **GetDocumentElement():** Entra al archivo XML y obtiene la etiqueta raíz.
- **GetChildNodes():** Entra al archivo XML y obtiene las etiquetas dentro de la raíz.
- **Item():** Obtiene los datos dentro de las Etiquetas que están dentro de la etiqueta raíz en un archivo XML.

Descripción de los métodos implementados.

- Paquete Logic: En este paquete se maneja las estructuras de datos lineales, los algoritmos de ordenamiento, en fin toda la lógica del motor de búsqueda se encuentra aquí.
    - Clase CircularList: Clase para crear las listas circulares.
        - CircularList: Constructor de la clase.
        - setHead: Método para modificar el head de la lista.
        - insertHead: Método para insertar por el head en la lista.
        - insertTail: Método para insertar por el tail en la lista.
        - insertInOrder: Método para insertar en orden a la lista
        - print: Método para imprimir la lista.
        - getHead: Método para obtener el head de la lista.
        - delete: Método para eliminar un dato de la lista.
        - find: para verificar si esta un elemento específico en la lista.

    - Clase List: Clase para crear las listas doblemente enlazadas.
        - List: Constructor de la clase.
        - setHead: Método para modificar el head de la lista.
        - setTail: Método para modificar el tail de la lista.
        - insertHead: Método para insertar por el head en la lista.
        - insertTail: Método para insertar por el tail en la lista.
        - insertInOrder: Método para insertar en orden en la lista.
        - print: Método para imprimir la lista.
        - find: Método para verificar si esta un elemento específico en la lista.
        - delete: Método para eliminar un dato de la lista.
        - getTail: Método para obtener el tail de la lista.
        - getHead: Método para obtener el head de la lista.

    - Clase Main:
        - getPrevNode: Método para obtener el anterior nodo.
        - getNextNode: Método para obtener el siguiente nodo.
        - getData: Método para obtener el dato de un elemento.

    - Clase Node: Clase para crear los nodos.
        - Node: Constructor de la clase.
        - setData: Método para modificar el dato.
        - setNextNode: Método para modificar el siguiente nodo.
        - setPrevNode: Método para modificar el nodo previo.

        - dequeue: Método para desencolar.
        - QueueList: Clase para manipular la cola.

    - Clase queueList: Clase para manipular la cola.
        - QueueList: Constructor de la clase.
        - enqueue: Método para encolar.

    - Clase SpiderBot: Clase para realizar el SpiderBot.
        - SpiderBot: Constructor de la clase.
        - obtenerurl: Metodo para obtener los datos de un xml, en este caso,el de "targets".

    - Clase extraerLinks:
        - extraerLinks: Constructor de la clase.
        - extraerTexto: Extrae todo el texto HTML.
        - extraerURL: Extrae links del texto HTML.
        - verificar: Verifica que los links están compuestos correctamente.

    - Clase keyBykey:
        - keyByKey: Constructor de la clase.
        - procesar: Funciona igual que un Tokenaizer, osea me separa un texto en palabras.

    - Clase keywordsReferencia:
        - keywordsReferencia: Constructor de la clase.
        - setURL: Es para setear el URL.
        - setName: Es par setear la palabra.
        - getURL: Obtiene el Url que lleva asociada la Keyword.
        - getName: Obtiene un keyword.

    - Clase leerxml:
        - leer: Lee toda la configuración de un XML.

    - Clase paginas:
        - paginas: Constructor de la clase.
        - setTrustworthy: Modificar el Trustworthy.
        - getTrustworthy: Obtiene el valor de Trustworthy.

    - Clase url:
        - url: Constructor de la clase.
        - setNumAsoc: Modifica el número de asociado.

    - setDireccion: Setea la dirección.
    - getDireccion: Obtiene la dirección del objeto URL.
    - getNumAsoc: Obtiene el número asociado del objeto URL.

Descripción de las estructuras desarrolladas.

- List: Esta estructura es una lista doblemente enlazada, es decir cada objeto Node tiene una referencia hacia “adelante” y hacia “atrás”; además cuenta con la referencia en el primer elemento (head) y el último (tail). Dicha estructura es utilizada para los keywords y las referencias de las palabras hacia la lista circular de url’s.

- CircularList: Estructura del tipo lista circular doblemente enlazada. Cuenta con la referencia head, que es por donde se ingresa un nuevo elemento, este se puede ubicar en cualquier Node; utilizada para los Urls.

- QueueList: Estructura constituida de Node’s, con la utilidad de funcionar como cola (FIFO). Se utiliza para procesar los urls, según su nivel de profundidad.

Descripción detallada de los algoritmos desarrollados.

Clase CircularList:

```
35      public void insertHead (G pData){
36          if (_head==null){
37              _head=new Node (pData, _head,_head );
38              _head.setNextNode(_head);
39              _head.setPrevNode(_head);
40          }
41          else {
42              _head=(new Node (pData, _head, _head.getPrevNode()));
43              _head.getNextNode().setPrevNode(_head);
44              _head.getPrevNode().setNextNode(_head);
45          }
46      }
```

```
93      public Node delete (G pData){
95          Node tmp=null;
96          if (_head==null)
97              return null;
98          else if ((Integer)_head.getData()==(Integer)pData && _head.getNextNode()==_head){
99              tmp=_head;
100             _head=null;
101         }
102         else{
103             tmp=_head;
105             _head=_head.getNextNode();
107             while(tmp!=_head &&(Integer) _head.getData()!=(Integer)pData)
108                 _head=_head.getNextNode();
109             if (tmp==_head && (Integer)_head.getData()!=(Integer)pData)
110                 tmp=null;
111             else{
112                 tmp=_head;
113                 tmp.getPrevNode().setNextNode(tmp.getNextNode());
114                 tmp.getNextNode().setPrevNode(tmp.getPrevNode());
115                 _head=tmp.getNextNode();
116                 tmp.setNextNode(null);
117                 tmp.setPrevNode(null);
118             }
119         }
120         return tmp;
121     }
```

```
124     public boolean find (G pData){
125         Node tmp = _head;
126         boolean condition = false;
127         while(tmp.getNextNode()!=_head){
                if ((Integer)tmp.getData()!= (Integer)pData)
129                 tmp=tmp.getNextNode();
130             else{
131                 condition=true;
132                 break;
133             }
134         }
            if ((Integer)tmp.getData()== (Integer)pData)
136             condition=true;
137         System.out.println(condition);
138         return condition;
139     }
```

Este algoritmo realiza lo siguiente:

En el método **InsertHead** se toma el URL obtenido de la cola y se agrega a la lista circular doble de URL's procesadas. **delete** recibe a un URL, el mismo será buscado entre los datos de la lista y se eliminará de ella. El método **find** recibe un URL y retorna falso o verdadero si el URL se encuentra o no agregado en la lista. El método **print** realizará una impresión de toda la lista cuando el programador la solicite.

Clase List:

```
51      public void insertHead(G pData){
52          if (_head==null){
53              _head=(new Node (pData, _head, _tail));
54              _tail=_head;
55          }
56          else{
57              _head=(new Node (pData, _head, null));
58              _head.getNextNode().setPrevNode(_head);
59          }
60      }
61      /**
62       * Metodo para insertar por el tail en la lista.
63       * @param pData. Dato a insertar.
64       */
65      public void insertTail (G pData){
66          if(_head==null){
67              _head= new Node(pData, _head, _tail);
68              _tail=_head;
69          }
70          else{
71              Node tmp = _head;
72              while(tmp.getNextNode()!=null)
73                  tmp=tmp.getNextNode();
74              tmp.setNextNode(new Node(pData, null, tmp));
75              _tail=tmp.getNextNode();
76          }
77      }
```

```
109     public Node delete (G pData){
            Node tmp = null;
111         if (_head == null)
112             return _head;
            else if ((Integer)_head.getData()==(Integer)pData){
114             tmp=_head;
115             if (_head.getNextNode()==null){
116                 _head=null;
117             }
118             else{
119                 tmp.getNextNode().setPrevNode(null);
120                 _head=_head.getNextNode();
121                 tmp.setNextNode(null);
122             }
123         }
124         else{
125             tmp = _head.getNextNode();
                while(tmp!=null && (Integer)tmp.getData()!=(Integer)pData)
127             tmp=tmp.getNextNode();
128         if (tmp==null)
129             return null;
130         else{
131             if (tmp.getNextNode()==null){
132                 tmp.getPrevNode().setNextNode(null);
133                 _tail=tmp.getPrevNode();
134                 tmp.setPrevNode(null);
135             }
136             else {
137                 tmp.getPrevNode().setNextNode(tmp.getNextNode());
138                 tmp.getNextNode().setPrevNode(tmp.getPrevNode());
139                 tmp.setNextNode(null);
140                 tmp.setPrevNode(null);
141             }
```

```
151     public boolean find (G pData){
152         Node tmp = _head;
153         boolean condition = false;
154         while(tmp!=null){
                if ((Integer)tmp.getData()!=(Integer)pData)
156                 tmp=tmp.getNextNode();
157             else{
158                 condition=true;
159                 break;
160             }
161         }
162         return condition;
163     }
```

Este algoritmo realiza lo siguiente:

En el método **InsertHead** se toma una palabra obtenido de la lectura del texto HTML y se agrega a la lista lista doble de Keywords. **delete** recibe a una palabra, la misma será buscada entre los datos de la lista y se eliminará de ella. El método **find** recibe una palabra y retorna falso o verdadero si la palabra se encuentra o no agregada en la lista. El método **print** realizará una impresión de toda la lista cuando el programador la solicite.

Clase QueueList:

```
15      public QueueList (Node head, Node tail){
16          this._head=head;
17          this._tail=tail;
18      }
19      /**
20       * Metodo para encolar.
21       * @param pData. Dato para encolar.
22       */
24      public void enqueue (G pData){
25          if (_head == null)
26              _head=_tail=new Node (pData, null, null);
27          else{
28              _tail.setNextNode(new Node (pData, null, _tail));
29              _tail=_tail.getNextNode();
30          }
31      }
32      /**
33       * Metodo para desencolar
34       * @return Nodo desencolado.
35       */
36      public Node dequeue (){
37          if (_head == null)
38              return null;
39          else{
40              Node tmp=_head;
41              _head=_head.getNextNode();
42              _head.setPrevNode(null);
43              tmp.setNextNode(null);
44              return tmp;
45          }
46      }
    }
```

Este algoritmo realiza lo siguiente:

El metodo **enqueue**: Toma un URL obtenido de la lectura del archivo XML y lo encola en la cola de URL's,. **dequeue** retorna la primer URL de esta cola.

Clase extraerLinks:

```
31      public void extraerTexto(String string_url)throws MalformedURLException, IOException{
32          URL url = new URL(string_url);
33          URLConnection conexion=url.openConnection();
34          InputStream entrada =conexion.getInputStream();
35          BufferedReader br= new BufferedReader(new InputStreamReader(entrada));
36          String contenido="";
37          String linea=br.readLine();
38          while (linea!=null){
39                  contenido+=linea;
40                  linea=br.readLine();
41          }
42          extraerURL(contenido);
43      }
```

```
    public void procesar(){
        String nombre="";
        //keywords_referencia objeto_keyword;
        while(!dato.equals("")){
            Character chardato =dato.charAt(0);
            if(Character.toString(chardato).equals(" ")){
                if (nombre!="")
                    //lista.insertHead(nombre);
                    //lista.insertHead(new keywords_referencia(nombre, cir_lis.getHead(
                dato=dato.substring(1,dato.length());
                nombre="";
            }
            else{
                nombre=nombre+chardato;
                dato=dato.substring(1,dato.length());
            }
        }
        lista.print();

    }
}
```

```
    public void extraerURL(String contenido){
        Pattern patron=Pattern.compile("(?i)HREF\\s*=\\s*\"(.*?)\"");
        Matcher matcher=patron.matcher(contenido);
        while(matcher.find())
            verificar(matcher.group(1), _url);
        }
    /**
     * Metodo para dar el formato a los links
     * @param dato. link a analizar
     * @param url. Pagina solicitada
     */
    private void verificar(String dato, String url){
        Character chardato =dato.charAt(0);
        System.out.println(url+dato);
        if(Character.toString(chardato).equals("/"))
            System.out.println(url+dato);
        else if (Character.toString(chardato).equals("#"))
            System.out.println("nada");
        else if (Character.toString(chardato).equals("h")){

            String tmp = "";
            int i=0;
            while((tmp!="https:" || tmp!="http:") && i<6){
                tmp+=dato.charAt(i);
                i++;
            }
            if (tmp!="https:" || tmp!="http:" )
                System.out.println(dato);
            else
                System.out.println(url+"/"+dato);
        }
        else
            System.out.println(url+"/"+dato);
    }
```

El algoritmo anterior funciona de la siguiente manera:

El metodo **extraerTexto** se conecta al servidor de una pagina web segun el URL recibido, se toma todo el texto escrito en lenguaje HTML .Con los metodos **extraerURL** se pasea el texto para sacar todos los links que se encuentran en el. Y con el método **verificar** se analiza cada link obtenido para ver si esta escrito correctamente o si hay que corregirlo y completarlo.

Clase keyBykey:

```
31      public void procesar(){
32          String nombre="";
33          //keywords_referencia objeto_keyword;
34          while(!dato.equals("")){
35              Character chardato =dato.charAt(0);
36              if(Character.toString(chardato).equals(" ")){
                    if (nombre!="")
                        //lista.insertHead(nombre);
                        //lista.insertHead(new keywords_referencia(nombre, cir_lis.getHead(
40                  dato=dato.substring(1,dato.length());
                    nombre="";
42              }
43              else{
44                  nombre=nombre+chardato;
45                  dato=dato.substring(1,dato.length());
46              }
47          }
48          lista.print();
49
50      }
51  }
```

Este algoritmo toma el texto de una pagina HTML y separa cada palabra. Su funcion es similar a la de un Tokenizer, Asimismo es capaz de agregar palabra por palabra a la lista de Keywords.

Clase leerxml:

```
40      public String leer (String direccion, int indice) throws ParserConfigurationException, FileNotFoundException, SAXExce
41          try {
42              DocumentBuilderFactory DBF = DocumentBuilderFactory.newInstance();
43              DocumentBuilder DB = DBF.newDocumentBuilder();
44              Document documento = DB.parse(direccion);
45
46              //Obtener el elemento raiz del documento
47              Element raiz = documento.getDocumentElement();
48
49
50              //Obtener los elementos de la raiz
51              NodeList elemRaiz = raiz.getChildNodes();
52
53              Node dato = elemRaiz.item(indice);
54              if(dato.getNodeType()==Node.ELEMENT_NODE) {
55
56                      //El valor está contenido en un hijo del nodo Element
57                      Node datoContenido = dato.getFirstChild();
58
59                      //Mostrar el valor contenido en el nodo que debe ser de tipo Text
60                      if(datoContenido!=null && datoContenido.getNodeType()==Node.TEXT_NODE)
61                          //System.out.println(datoContenido.getNodeValue());
62                          return datoContenido.getNodeValue();
63                      }
64
65          } catch (SAXException ex) {
66              System.out.println("ERROR: El formato XML del fichero no es correcto\n"+ex.getMessage());
67              ex.printStackTrace();
68          } catch (IOException ex) {
69              System.out.println("ERROR: Se ha producido un error al leer el fichero\n"+ex.getMessage());
```

Este algoritmo realiza lo siguiente:
Realiza la lectura de un archivo XML, crea un documento en blanco al cual le va agregando los datos que se encuentran adentro de las etiquetas. Este algoritmo funciona igual a la hora de extraer los URL's del archivo XML que los contiene como a la hora de extraer la configuración del SpiderBot.

Problemas Conocidos

- Funcionamiento del spiderEngine.
- Funcionamiento de ciertos links obtenidos, de diferentes paginas; por ejemplo http://www.amazon.com.
- Presentar los resultados finales de la búsqueda.
- Realizar el sistema de calificación de los resultados de las diferentes páginas web.
- Aun se cuenta con el problema de que a la hora de extraer las palabras de texto, se extraen también las palabras de código JavaScript que están escritas en la parte del código de HTML, como por ejemplo palabras como heigth, weith.,Ya se conoce que en ningún momento a la hora de parsear se esta tomando en cuenta el código javascript dentro del HTML.
- El conteo de las veces que sale la palabra por página, ya que no se ha logrado establecer la relación con el url.
- La manipulación de los threads.

Actividades realizadas (timesheet).

Fecha	Gerald	Jairo
11 Marzo	**9:30-11:30**. Reunión para crear una idea general del proyecto y una primera división del trabajo.	
14 Marzo	**10:30-11:30**. Realización de una primera versión del diagrama de clases.	
20 Marzo	**15:00-17:00.** Pequeña investigación sobre generics y Telnet.	
21 Marzo		**14:00- 17:00**. Creación de la lista doble, circular y cola de manera genérica.
23 Marzo	**13:00-15:00.** Investigacion y creacion de la clase para leer archivos XML.	
24 Marzo	**18:00-20:00.** Se desarrolló la clase para conectarse a una página WEB y obtener su texto HTML.	
28 Marzo	**13:30-15:30**. Descripción más detallada de la forma en que funciona la interacción entre las clases, así como el inicio de la unión de las mismas.	
29 Marzo	**16:00-21:00.** Desarrollo de clase para extracción de palabras e inserción de estas en la lista Keywords.	
30 Marzo	**14:00-20:00 22:00-23:00 01:00-03:00**. Correcciones a la clase lectora de archivos XML. Correcciones a los problemas del dequeue.	
31 Marzo	**14:30-20:00 22:00-06:00.** Investigacion y creacion de los Threads. Correcciones a la clase extractora de URL. Mejora en el spiderBot.	
1 Abril	**9:30-13:00.** Comienzo de la parte escrita.	
2 Abril	**15:30-23:00.** Culminación de la parte escrita.	
Total Horas invertidas	48.5 horas	46.5 horas

Problemas encontrados.

- Realizar un casteo de mala forma. Esto se daba basicamente a una mala ubicacion en los parentesis para reailzar el mismo; para este problema se se encontro solucion de la fuente: https://sites.google.com/site/pro012iessanandres/java/conversion-hacia-abajo-casting-de-objetos.
- Manejo de los links obtenidos. En este problema sucede que de la página web, se obtiene links que se refieren a la misma dirección web, por ejemplo, cuando se procesa http://www.google.com, algunos de los links que se obtienen son “/search” o “search”; para corregir este problema se creó un método que evalúa lo obtenido y a partir de ahí se reparan las direcciones para poder ingresar a ellas.
- Selección de el patrón para obtener los links de las paginas.Para solucionar este problema se consultó en diversas fuentes y se utilizó: ("(?i)HREF\\s*=\\s*\"(.*?)\"")
- Obtención de diferentes links de las páginas web. Para ello debe verificarse la clase que se encarga de extraer los links.
- Manejo del dequeue: esto sucedia cuando se quería transmitir datos, se hacia un doble dequeue “sin intención” esto cuando se realizaban diferentes funciones. Para solucionar este inconveniente se decide guardar en una variable lo obtenido y utilizar solamente esa variable para la transmisión del dato.
- La manipulación de los threads: Este problema aun esta en proceso de solución se han consultado diferentes fuentes entre ellas: http://javabasico.osmosislatina.com/curso/progfinal2/threads2.htm, http://proton.ucting.udg.mx/tutorial/java/Cap7/thread.html, entre otras.

Conclusiones y recomendaciones.

Conclusiones:

El spider Search Engine: Stage 1, es un programa para realizar búsquedas en internet, la característica principal es que esta constituido a partir de estructuras de datos lineales como es el caso de listas doblemente enlazadas; si bien es cierto que estas estructuras son poco eficientes comparadas con otras, se demuestra que estas pueden ser de utilidad para diferentes problemas.

Por otro lado la implementación de los tipos de datos genéricos volvió innecesaria la comprobación de los tipos de datos ingresados durante el tiempo de ejecución, lo cual evidentemente reduce la utilización de recursos del computador, además de hacer que el código sea poco ambiguo y que sea reutilizable para la implementación de otra función, como por ejemplo la utilización de las listas para almacenado de datos totalmente diferentes.

Recomendaciones:

Debido a que el Spider Seach Engine: Stage 1, no ha sido probado en computadores con menor capacidad a la siguiente:
Procesador:

- Procesador Intel® Atom™ N455.
    - 1.66 GHz, 512KB L2 Cache

Memoria:

- Configurada con 2GB DDR2 (trabaja a 667MHz, max 2GB).

Además se probó en los siguientes sistema operativos:

- Windows 8 y Windows 7
- Ubuntu 14.04 y sus distribuciones Xubuntu 14.04 y Lubuntu 14.04.

Por ende para asegurarse una experiencia gratificante es recomendable correr este programa con un mínimo de las características anteriormente mencionadas.

Bibliografía.

- Universidad de Guadalajara. (agosto 25, 2004). Tutorial Java: Threads y Multithreading. marzo 30, 2015, de Universidad de Guadalajara Sitio web: http://proton.ucting.udg.mx/tutorial/java/Cap7/thread.html
- Jenkov, J. Creating and Starting Java Threads. Marzo 31, 2015, de http://tutorials.jenkov.com/ Sitio web: http://tutorials.jenkov.com/java-concurrency/creating-and-starting-threads.html
- stackoverflow.com. (octubre 17, 2012). Java- creating a new thread. marzo 31, 2015, de stackoverflow.com Sitio web: http://stackoverflow.com/questions/17758411/java-creating-a-new-thread
- www.chuidiang.com. (enero 10, 2004). Sincronización de hilos. marzo 31, 2015, de www.chuidiang.com Sitio web: http://www.chuidiang.com/java/hilos/sincronizar_hilos_java.php
- OsmosisLatina. (Noviembre 17, 2012). Uso de "Threads" con sincronización. marzo 31, 2015, de OsmosisLatina Sitio web: http://javabasico.osmosislatina.com/curso/progfinal2/threads2.htm
- Jenkov, J. Locks in Java. Marzo 31, 2015, de http://tutorials.jenkov.com/ Sitio web: http://tutorials.jenkov.com/java-concurrency/locks.html
- Jenkov, J. Lock. Marzo 31, 2015, de http://tutorials.jenkov.com/ Sitio web: http://tutorials.jenkov.com/java-util-concurrent/lock.html
- tutorialspoint. (Mayo 19, 2003). Java Multithreading. marzo 31, 2015, de tutorialspoint Sitio web: http://www.tutorialspoint.com/java/java_multithreading.htm
- puntocomnoesunlenguaje.blogspot.com. (agosto 07, 2013).Ejemplos de Expresiones Regulares en Java . marzo 31, 2015, de puntocomnoesunlenguaje.blogspot.com Sitio web: http://puntocomnoesunlenguaje.blogspot.com/2013/07/ejemplos-expresiones-regulares-java-split.html
- Garcia, L.. (octubre 8, 2010). Java Concurrente: Uso de Semáforos. 31 de marzo del 2015, de WordPress.com Sitio web: puntocomnoesunlenguaje.blogspot.com.
- Santamaría, R. Java Threads . marzo 31, 2015, de vis.usal.es Sitio web: http://vis.usal.es/rodrigo/documentos/aso/seminarios/javaThread.pdf
- Goi Mailako Escola. (1999). Aprenda Servlest en Java. 12 Marzo 2015, de tecnum Sitio web: http://www4.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/javaservlets/servlets.pdf
- Google. (2015). Crear un sitemap. 14 de Marzo de 2015, de Google Sitio web: https://support.google.com/webmasters/answer/183668?hl=es

- Fernando Campaña. (2006). ¿Qué es y para qué sirve un sitemap?. 14 de Marzo de 2015, de Maestros del Web Sitio web: http://www.maestrosdelweb.com/sitemap/
- Oracle and/or its affiliates. (2014). Class StringTokenizer. 22 de Marzo de 2015, de Oracle Sitio web: https://docs.oracle.com/javase/7/docs/api/java/util/StringTokenizer.html
- Desconocido. (2001). HTML Parser. 23 de Marzo de 20156, de Anonima Sitio web: http://htmlparser.sourceforge.net/
- Alex Guerra (Xela) . (2008). Java y XML: DOM (II) . 28 de Marzo de 2015, de La Tasca de Xela Sitio web: www.latascadexela.es/2008/07/java-y-xml-dom-ii.html
- Anónimo . (2006). La clase StringTokenizer. 23 de Marzo de 2015, de EHU Sitio web: http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/colecciones/stringtokenizer.html
- Rodrigo Santamaria . (2010). Java Threads . 1 de Abril de 2015, de USAL Sitio web: http://vis.usal.es/rodrigo/documentos/aso/seminarios/javaThread.pdf