# Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación


# Segundo Proyecto Programado

# SpiderSearch Engine: Stage 2

Elaborado por:
Arturo Chinchilla Sánchez
Fabricio Miranda Leitón

Curso:
Algoritmos y Estructuras de Datos 1
Grupo 2

Profesor:
Kevin Moraga García

Cartago Mayo I Semestre 2015

Tabla de contenido

## Descripción detallada de los algoritmos desarrollados.

➔ java.net:

Proporciona las clases para la implementación de aplicaciones de red. El paquete java.net puede dividirse en dos secciones:

1- Un API de bajo nivel, que se ocupa de los siguientes abstracciones:

Direcciones, los cuales son identificadores de redes, como direcciones IP.

Sockets, que son mecanismos básicos de comunicación bidireccional de datos.

Interfaces, que describen las interfaces de red.

2- Un API de alto nivel, que se ocupa de los siguientes abstracciones:

URI, que representan identificadores de recursos universales.

URLs, que representan los localizadores de recursos universales.

Conexiones, lo que representa conexiones al recurso apuntado por las direcciones URL.

- ❖ java.net.ContentHandler:
  Dada una URL conecta una transmisión posicionada en el comienzo de la representación de un objeto, este método lee esa transmisión y crea un objeto que coincide con uno de los tipos especificados. La implementación por defecto de este método debe llamar getContent () y detectar el tipo de retorno para una coincidencia de los tipos sugeridas.
- ❖ java.net.URL:
  La clase URL representa un Localizador Uniforme de Recursos, un puntero a un "recurso" en la World Wide Web.
- ❖ java.net.MalformedURLException:
  Lanzado para indicar que se ha producido una URL con formato incorrecto.

➔ java.util:

Contiene un marco de colecciones, clases de colección legadas, modelo de eventos, facilidades de fecha y hora, internacionalización, y clases de servicios públicos diversos (Tokenizer de strings, un generador de números aleatorios y un array de bits).

- ❖ java.util.StringTokenizer:
  Construye un tokenizer de tipo string para el string especificado. Los caracteres en el argumento delim son los delimitadores para separar tokens. Los caracteres delimitadores no serán tratados como tokens.
- ❖ java.util.random:

Una instancia de esta clase se utiliza para generar una corriente de números pseudoaleatorios. La clase utiliza una semilla de 48 bits, que está modificado usando una fórmula de congruencia lineal.

➔ java.io:

Proporciona un sistema de entrada y salida a través de flujos de datos, serialización y el sistema de archivos. A menos que se indique lo contrario, pasar un argumento nulo a un constructor o método en cualquier clase o interface en este paquete provocará una NullPointerException al ser lanzado.

- ❖ java.io.File:

  Una representación abstracta de las rutas de acceso de archivos y directorios.

- ❖ java.io.InputStream:

  Esta clase abstracta es la superclase de todas las clases que representan un flujo de entrada de bytes.

- ❖ java.io.IOException:

  Indica que se ha producido una excepción de E / S de algún tipo.

- ❖ java.io.BufferedInputStream:

  Un BufferedInputStream añade funcionalidad a otra secuencia de entrada, es decir, la capacidad de almacenar la entrada y apoyar la marca y re-establecer métodos. Cuando se crea el BufferedInputStream, se crea una matriz de amortiguación interna. A medida que se leen o saltan bytes de la corriente, el buffer interno se vuelve a llenar como sea necesario desde el flujo de entrada contenida, muchos bytes a la vez. La operación marca recuerda un punto en el flujo de entrada y la operación de reposición hace que todos los bytes leídos desde la última operación de la marca que se releen antes nuevos bytes se toman de la secuencia de entrada contenida.

- ❖ FileInputStream:

  Un FileInputStream obtiene bytes de entrada desde un archivo en un sistema de archivos. ¿Qué archivos están disponibles depende del entorno de host.

  FileInputStream está destinado para la lectura de flujos de bytes como datos de imagen. Para la lectura de corrientes de caracteres, considere el uso FileReader.

➔ javax.xml.parsers:

Proporciona clases que permiten el procesamiento de documentos XML. Se admiten dos tipos de analizadores:

- SAX (Simple API for XML)
- DOM (Document Object Model)

- ❖ javax.xml.parsers.DocumentBuilder:

Define el API para obtener las instancias de documento DOM de un documento XML.

- ❖ javax.xml.parsers.DocumentBuilderFactory:
  Define un API de fábrica que permite a las aplicaciones obtener un analizador que produce árboles de objetos DOM desde documentos XML.

➔ org.w3c.dom:

Proporciona las interfaces para el Document Object Model (DOM), el cual es un componente API de la API Java para Procesar XML. El Document Object Model Nivel 2 Core API permite a los programas acceder y actualizar dinámicamente el contenido y estructura de los documentos.

- ❖ org.w3c.dom.Document:
  La interfaz Document representa todo el documento HTML o XML.
- ❖ org.w3c.dom.NodeList:
  La interfaz NodeList proporciona la abstracción de un conjunto ordenado de nodos, sin definir o restringir cómo se implementa esta colección.
- ❖ org.w3c.dom.Node:
  La interfaz Node es el tipo de datos primaria para todo el Document Object Model.

➔ org.apache:

Kit de herramientas que detecta y extrae los metadatos y texto de más de mil tipos de archivos diferentes (tales como PPT, XLS y PDF). Todos estos tipos de archivos se pueden analizar a través de una única interfaz, haciéndola útil para la indexación del motor de búsqueda, análisis de contenido, traducción, y mucho más.

## Descripción de los métodos implementados.

- XMLParser( ):
  Encargado de recuperar información directamente desde archivos XML. Recibe como parámetros el nombre del archivo y el tag que se desea encontrar en el XML, ambos de tipo String.
  - obtenerCola( ): Cola
    Encargado de parsear el archivo XML y obtener una cola con los datos que se desean.

- Separar(String name):
  Se utiliza para poder separar todas las palabras una vez que se ha realizadoel parseo del documento previamente seleccionado. Recibe como parámetro un texto

  - getWords(String text, String separador):
    Éste método es utilizado para poder eliminar todos las excepciones previamente establecidas en un arreglo llamado separador, retornando así una palabra sin ningún tipo de signos.

  - cicloToken():
    Encargado de ejecutar un ciclo que verifique si aún hay más palabras por verificar en un texto dado, retornando cada palabra hasta que el mismo éste completo.

- TikaParser(String archivo):
  Su función es realizar un parseo sobre un documento dado, sin importar su formato. Esta biblioteca recibe un archivo y devuelve un String con todo el texto contenido en el mismo.

  - GetHandler():
    Se utiliza sólo para retornar el handler, en este caso el String con el texto.

- Hilo( ):
  - run( ): void
    Ejecuta las tareas del hilo.

## Descripción de las estructuras de datos desarrolladas.

➔ Clase Nodo:
Su constructor recibe como único parámetro un dato de tipo Object, el cual será el valor a almacenar dentro del nodo.

Métodos:

- setnodoIzq(Nodo nodo): Establece un nodo izquierdo.
- setnodoDer(Nodo nodo): Establece un nodo derecho.
- getnodoIzq(): Retorna el nodo izquierdo.
- getnodoDer(): Retorna el nodo derecho.
- getDato(): Retorna el atributo dato.
- isComplete(): Booleano que verifica si un nodo está completo o no.

➔ Clase NodoArbolAVL:
Esta clase se encarga crear un Árbol de tipo AVL(Árbol Binario de Búsqueda) que tiene la capacidad de auto-balancearse.
Métodos:

- insertar(String palabra, int cantidad): Este método recibe como parámetros un entero y un entero, crea una instancia de la clase Palabra y dicha instancia es guardada dentro de un NodoArbolAVL que será agregado al Arbol AVL.
- insertarAVL(NodoArbolAVL nuevo, NodoArbolAVL subAr): Recibe dos parámetros de tipo NodoArbolAVL, los cuale el primero es el Nodo a ser insertado y el segundo es la raíz del árbol al cual se le quiere hacer una insercion, en caso de que no esté vacío.
- obtenerRaiz(): Retorna un NodoArbolAVL con la raíz del árbol en cuestión.
- buscar(String d, NodoArbolAVL r): Busca en cada NodoArbolAVL la palabra asignada al atributo d, comenzando por la raiz del Arbol AVL asignada al atributo r, en caso de haber coincidencia retorna el nodo.
- eliminar(String d): Busca el dato almacenado en el atributo d dentro del arbol, si hay alguna coincidencia el NodoArbolAVL que lo contenga será eliminado.
- inOrden(NodoArbolAVL): Hace un recorrido de todo el árbol, recorriendo primero: Busca el string asignado al atributo d y los busca en todo el Árbol AVL, en caso de haber alguna coincidencia el Nodo que contiene este string será eliminado.
- obtenerFE(NodoArbolAVL x): Recibe como parámetro el NodoArbolAVL al cual se le desea calcular el Factor de Equilibrio.
- rotacionDerecha(NodoArbolAVL c): Como parámetro recibe un nodo con problemas en el Factor de Equilibrio, esta función hace una rotación simple a la derecha para corregir este problema.

- rotacionIzquierda(NodoArbolAVL c): Como parámetro recibe un nodo con problemas en el Factor de Equilibrio, esta función hace una rotación simple a la izquierda para corregir este problema.
- rotacionDobleDerecha(NodoArbolAVL c): Como parámetro recibe un nodo con problemas en el Factor de Equilibrio, esta función hace una rotación doble a la derecha para corregir este problema.
- rotacionDobleIzquierda(NodoArbolAVL c): Como parámetro recibe un nodo con problemas en el Factor de Equilibrio, esta función hace una rotación doble a la izquierda para corregir este problema.
- obtenerNodoReemplazo(NodoArbolAVL nodoReemp): Cuando un nodo debe ser eliminado, esta función obtiene el nodo que reemplazará el nodo a eliminar, recibe como parámetro el nodo a eliminar.
- inOrden(NodoArbolAVL r): Como argumento recibe la raíz del árbol que se desea recorrer, en orden de izquierda a derecha(izquierdo-raíz-derecho).
- postOrden(NodoArbolAVL r): Como argumento recibe la raíz del árbol que se desea recorrer, en orden izquierdo, derecho, raíz.
- preOrden(NodoArbolAVL r): Como argumento recibe la raíz del árbol que se desea recorrer, primero visita la raíz y luego ambos nodos hijos.

## Descripción detallada de los algoritmos desarrollados.

Módulo de Búsquedas:

- Lector, analizador y recuperador de datos de los archivos XML (Clase XMLParser):
  Mediante una instancia de la clase java.io.File se carga la ruta de acceso al archivo .XML, después creamos una instancia de la clase java.io.DocumentBuilder y java.io.DocumentBuilderFactory para obtener las instancias de documento DOM desde los archivos XML. Además de esto se crean instancias del tipo Node para obtener la cantidad de Nodos hijos de la estructura DOM y por un bucle for se recorre esta estructura para obtener los valores que se desean encontrar mediante un" tag".

- Creador de hilos (Clase Hilos):
  Al ejecutar el programa se crea una clase donde se creará un árbol AVL, además una clase Hilo que extiende de la clase Thread anidada en la clase, mediante un bucle *for* se van creando los hilos indicados en el documento XML que contiene las direcciones de los documentos a parsear, estos hilos tienen las funciones de parsear los documentos,(llamando la clase TikaParser), obtener las palabras (llamando la clase Separar) y estas meterlas en un árbol AVL.

- Clase Separador:
  Se utiliza un arreglo contenido de símbolos para poder separar las palabras e insertarlas en el árbol AVL sin ningún tipo de símbolos que puedan alterar el contenido de la misma, así como la utilización de un método para poder verificar las veces que una palabra está contenida en un documento.

## Problemas conocidos.

- Problemas externos al código:
  - Falta de horas programación y por ende falta de conocimiento de lenguaje java.
  - Falta de coordinación y compromiso para con el proyecto.
  - Carencia de tiempo, por lo cual no se pudo terminar a tiempo todas las especificaciones del proyecto.
  - El proyecto coincidió con varias evaluaciones de distintos cursos, limitando el tiempo designado para el proyecto.

- Problemas con respecto al código:
  - Los hilos trabajadores pueden agarrar una misma URL antes de que ésta sea quitada de la cola.
  - La implementación de sockets no se logró de una manera adecuada, ya que no se logró hacer comunicación cliente/servidor.
  - Falta de conocimientos en el tema de árboles, y complejidad en las soluciones de los problemas descubiertos.

## Actividades realizadas por el estudiante.

Resumen Bitácora

Arturo Chinchilla:
Investigación y creación del árbol AVL, además de su correcto funcionamiento con la implementación de métodos de búsqueda, inserción y eliminar. Además de la creación de estructuras de datos.Tiempo invertido: 10 Horas.

Fauricio Miranda: Parseo de documentos y su inserción correcta en un árbol, además de investigación sobre sockets, uso correcto del XML, creación de estructuras de datos. Tiempo invertido 10 horas.

## Conclusiones

Al ser Java un lenguaje de programación orientada a objetos hace que el manejo del código sea mejorado, esto quiere decir que facilita a tener un programa más estructurado, simple, y limpio. Es por esto que el manejo de bibliotecas externas, la creación de nuestras propias clases con sus respectivos métodos hace que sea más fácil manejar nuestros proyectos.

Respectivamente con este proyecto se observa como la organización y el trabajo en equipo es fundamental para lograr el éxito, sin embargo no se pudo alcanzar todas las metas propuestas para éste proyecto SpiderSearch Engine: Stage 2, siendo así la falta de comunicación y organización problemas que se dieron en el transcurso del mismo, a su vez se recomienda una mayor investigación para próximos proyectos.

## Bibliografías.


Apache Tika - Apache Tika Documentation (s. f.). Recuperado de https://tika.apache.org/0.5/documentation.html

Apache Tika 1.9-SNAPSHOT API (s. f.). Recuperado de https://tika.apache.org/1.8/api/

Cs.wellesley.edu/~cs231/fall01/red-black.pdf (s. f.). Recuperado de http://cs.wellesley.edu/~cs231/fall01/red-black.pdf

Heap Sort - Sorting Algorithm Animations (s. f.). Recuperado de http://www.sorting-algorithms.com/heap-sort

Java Platform SE 7 (s. f.). Recuperado de http://docs.oracle.com/javase/7/docs/api/

Lesson: All About Sockets (The Java? Tutorials > Custom Networking) (s. f.). Recuperado de https://docs.oracle.com/javase/tutorial/networking/sockets/