**Instituto Tecnológico de Costa Rica**

Escuela de Ingeniería en Computación



**Primer Proyecto Programado**

**SpiderSearch Engine: Stage 1**

Elaborado por:

Manuel Arturo Chinchilla Sánchez

Curso:

Algoritmos y Estructuras de Datos 1

Grupo 2

Profesor:

Kevin Moraga García

Cartago Abril I Semestre 2015

# Descripción de las Bibliotecas y Funciones.

* **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.BufferedReader:

Lee texto de una corriente de caracteres de entrada, almacenando temporalmente caracteres con el fin de prever la lectura eficiente de los caracteres, matrices y líneas.

* + java.io.IOException:

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

* **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.

1. 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.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.

* **org.w3c.dom:**

Proporciona las interfaces para el Document Object Model (DOM), el cuál 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.

* **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 producen árboles de objetos DOM desde documentos XML.

* **java.awt:**

Contiene todas las clases para crear interfaces de usuario y para el pintado de gráficos e imágenes. Un objeto de interfaz de usuario, como un botón o una barra de desplazamiento se llama, en la terminología de AWT, un componente.

* + java.awt.BorderLayout
  + java.awt.EventQueue
  + java.awt.GridLayout
* **java.swing:**

Proporciona un conjunto (todo-lenguaje Java) de componentes "ligeros" que, en la mayor medida posible, funcionan de la misma manera en todas las plataformas.

* + javax.swing.JPanel:

JPanel es un contenedor ligero genérico.

* + javax.swing.JButton:

Una implementación de un botón "push".

* + javax.swing.JTextField:

JTextField es un componente ligero que permite la edición de una sola línea de texto.

* + javax.swing.JLabel

Un área de visualización de una cadena corta de texto o una imagen, o ambos.

# 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.

* listaHilos()

Crea una listaEnlazada de Hilos, la cantidad de Hilos (Spiders) dependerá de los parámetros que se reciban desde el XML que contiene las configuraciones.

# Descripción de las estructuras de datos desarrolladas.

* Clase Nodo:

Su constructor recibe como único parámetro un dato de tipo Object, el cuál será el valor a almacenar dentro del nodo.

Además la clase Nodo tiene los métodos:

* + obtenerValor(): Object

Retorna el valor almacenado en el Nodo.

* + enlazarSiguiente(Nodo): void

Recibe y enlaza el siguiente nodo de la lista.

* + enlazarAnterior(Nodo): void

Recibe y enlaza el Nodo anterior de la lista.

* + enlazarProfundidad(Nodo): void

Recibe y enlaza un Nodo que contiene el valor de la profundidad. El valor de la profundidad es un valor único para las URL que se metan en la cola.

* + obtenerSiguiente(): Nodo

Retorna el Nodo siguiente de la lista.

* + obtenerAnterior(): Nodo

Retorna el Nodo anterior de la lista.

* + obtenerProfundidad

Retorna el Nodo que tiene asignado el valor de la profundidad.

* Clase Cola:

Crea un tipo de lista enlazada de nodos donde solo se pueden agregar nodos al final y eliminarlos al inicio de dicha lista. Su constructor no recibe ningún tipo de parámetro.

Métodos:

* + estaVacia(): boolean

Revisa si la cola está vacía, si lo está retorna el valor booleano true, si no lo está retorna false.

* + addFinal(Object): void

Crea un nodo, al cual se le pasa como valor el objeto que recibe este método y se agrega el nodo al final de la cola.

* + eliminaInicio(): void

Cambia de posición la cabeza de la cola, para que de esta forma eliminar el primer nodo de la cola.

* + obtenerInicio(): Nodo

Retorna el nodo que está a la cabeza de la cola.

* + imprimir(): void

Imprime en consola los valores contenidos en los nodos de la cola.

* + getSize(): int

Retorna el tamaño de la cola.

* Clase listaEnlazada:

Crea una lista de Nodos, la que se puede manejar por medio de sus propios métodos. Su constructor no recibe ningún parámetro.

Métodos:

* + estaVacia(): boolean

Revisa si la lista está vacía, si lo está retorna el valor booleano true, si no lo está retorna false.

* + addInicio(Object): void

Crea un nodo, al cual se le pasa como valor el objeto que recibe este método y se agrega el nodo al inicio de la lista.

* + addFinal(Object): void

Crea un nodo, al cual se le pasa como valor el objeto que recibe este método y se agrega el nodo al final de la lista.

* + eliminaInicio(): void

Cambia de posición la cabeza de la lista al nodo siguiente, para que de esta forma no quede referencia al primer nodo de la lista.

* + eliminaFinal(): void

Cambia de posición la cola de la lista al nodo anterior, para que de esta forma no quede referencia al último nodo de la lista.

* + eliminarenPos(int): void

Recibe como parámetro un índice, se recorre la lista y elimina la referencia al nodo ubicado en esta posición.

* + obtenerenPos(int): Object

Recibe como parámetro un índice, se recorre la lista y obtiene el objeto almacenado en el nodo ubicado en esta posición de la lista

* + imprimir(): void

Imprime en consola los valores contenidos en los nodos de la lista.

* + getSize(): int
  + Retorna el tamaño de la lista.
* Clase listaCircularDoble:

Crea una lista circular de Nodos, donde el último nodo apunta al primero, y el primero tiene una referencia al último. Su constructor no recibe ningún parámetro.

Métodos:

* + estaVacia(): boolean

Revisa si la lista está vacía, si lo está retorna el valor booleano true, si no lo está retorna false.

* + addInicio(Object): void

Crea un nodo, al cual se le pasa como valor el objeto que recibe este método y se agrega el nodo al inicio de la lista.

* + addFinal(Object): void

Crea un nodo, al cual se le pasa como valor el objeto que recibe este método y se agrega el nodo al final de la lista.

* + eliminaInicio(): void

Cambia de posición la cabeza de la lista al nodo siguiente, para que de esta forma no quede referencia al primer nodo de la lista.

* + eliminaFinal(): void

Cambia de posición la cola de la lista al nodo anterior, para que de esta forma no quede referencia al último nodo de la lista.

* + eliminarenPos(int): void

Recibe como parámetro un índice, se recorre la lista y elimina la referencia al nodo ubicado en esta posición.

* + obtenerenPos(int): Object

Recibe como parámetro un índice, se recorre la lista y obtiene el objeto almacenado en el nodo ubicado en esta posición de la lista

* + imprimir(): void

Imprime en consola los valores contenidos en los nodos de la lista.

* + getSize(): int
  + Retorna el tamaño de la lista.

# Descripción detallada de los algoritmos desarrollados.

* 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. Ademas de ésto se crean instancias del tipo Node y NodeList para Mediante un buble for se recorre la estructura DOM del