Universidad de las Ciencias Informáticas

Facultad 15

Título: Implementación de una herramienta de recuperación de información dentro de la Informática Jurídica.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autor: Alberto Mercader Guevara

Tutor: Lic. Reyder Cruz de la Osa

Ciudad de La Habana, 2009-2010

DECLARACIÓN DE AUTORÍA

Declaramos ser los autores del presente trabajo de diploma y autorizamos a la Universidad de las Ciencias Informáticas para que hagan uso pertinente de este trabajo.

Para que así conste firmamos el presente a los – días del mes de

_____ _____

Firma de Autor Firma de Autor

_____ _____

Firma Tutor Firma Tutor

_____

Firma Tutor

AGRADECIMIENTOS

Ante todo, agradecer infinitamente el esfuerzo que día a día mis padres hacen por mí, que nada de esto fuera posible sin ellos, a ellos que me han sabido criar con tanto cariño y sabiduría, a ellos que le debo mi existir y mi pensar.

A toda mi familia, tías, tíos, primos que quiero mucho y todas las personas que de una forma u otra forman parte de mi vida.

A Arlena mi novia que jamás dejo ni un segundo de dudar de mi, por darme un bebe tan lindo como el que tengo y compartir sin esperar nada a cambio todo estos días de mi vida.

A mis amistades de la UCI que sin ellos nada hubiese sido igual, al piquete de viciosos que han sabido compartir uno de los mejores momentos de mi vida, que no me olvidare de ellos porque son amigos para siempre.

A Walter por compartir buenos momentos conmigo.

A todos los profesores que formaron parte de mi carrera.

Y por ultimo y sin quietarle importancia a esta escuela maravillosa.

Albert

DEDICATORIA

A mis padres por estar siempre presentes, por ayudarme cuando lo necesitaba y demostrar tanto amor.

A toda mi familia hermano, primos, tíos y tías que siempre estuvieron presentes.

A mis amigos del barrio Javier, Pedro, Yadiel y los de la escuela que siempre estuvieron para ayudarme y compartir conmigo.

A mi novia por estar siempre en los malos y buenos momentos.

A mi tutor por ayudarme, preocuparse y ocupar horas de su trabaja a mi tesis.

A todos los maestros y profesores que han contribuido a mi formación.

RESUMEN

El presente trabajo se refiere a la propuesta de una herramienta que permita ayudar a gestionar la información dentro de un gran cúmulo de la misma en el proyecto Tribunales Populares Cubanos (TPC). Esta herramienta de asistencia está basada en técnicas de recuperación de información. Está desarrollada utilizando C++ como lenguaje de programación. La implementación está hecha utilizando la plataforma de desarrollo .Net 2008 y se utiliza Lucene como API de desarrollo para indexación y búsquedas. Se utilizará la versión implementada para C++, en este caso Clucene, la cual presenta funcionalidades para lograr un mejor manejo de la información en cuanto a la indexación y búsqueda de esta. Su objetivo fundamental es lograr un mejor desempeño en el proyecto de Tribunales Populares Cubanos (TPC).

Palabras clave: sistemas de información, modelo vectorial, Lucene, relevancia, peso asociado, recobrado, precisión, lematización, indización.

INDICE

INTRODUCCION

El tesoro más valioso de la raza humana es el conocimiento. Gran parte de este conocimiento existe en forma de lenguaje natural: libros, periódicos, artículos, etcétera. La posesión real de todo este conocimiento depende de la habilidad para realizar ciertas operaciones con la información, por ejemplo: buscarla, compararla y resumirla.

La lingüística computacional se enfoca principalmente en el diseño de los mecanismos que permitan a las computadoras entender el lenguaje natural, aunque también considera varias tareas relacionadas con el procesamiento de información textual. Algunos ejemplos de estas tareas son la búsqueda de información, la extracción de información, entre otros.

Los sistemas de búsqueda han adquirido una gran importancia en el uso cotidiano de los ordenadores hasta el punto de que realizar una consulta en un buscador es la acción más frecuente, tras el envío de un correo electrónico. Sin embargo, la búsqueda y recuperación de información textual tienen asociadas una serie de problemas todavía no resueltos satisfactoriamente. Algunos de estos problemas provienen de la ambigüedad y la falta de estructura propias del lenguaje natural. En la actualidad es inconcebible poder manejar grandes cantidades de información sin ayuda de sistemas automatizados que permitan un mejor manejo de la misma, además de que ahorran tiempo y por consiguiente trabajo, esfuerzos y recursos. Estos sistemas de recuperación de información se basan de dos principales procesos: uno es la indización de los documentos o información, y la otra parte o el segundo proceso es la búsqueda automatizada y eficiente.

Debido a la falta de experiencia en el tema y la necesidad que representa el manejo eficaz de la información, hoy se hace necesario desarrollar técnicas que permitan un mejor manejo de la información, lo cual representa un problema a resolver dentro de la informática jurídica.

Por todo lo antes mencionado se presenta una situación problemática a la cual el presente trabajo de diploma propone una solución que plantea como **problema a resolver:** ¿Cómo lograr que las personas del ámbito jurídico tenga la posibilidad de manejar más fácil la información dentro de una gran cantidad de la misma? (La detección de asociaciones entre disímiles casos se realiza de forma manual, lo que resulta grandemente tedioso y bastante lento).

Siendo el **objeto de estudio** principal los sistemas de recuperación de información.

Determinándose como **objetivo general:** Realizar la implementación y validación de una herramienta de recuperación de información.

Presentando como **campo de acción** en esta investigación**:** Los modelos algorítmicos en los sistemas de recuperación de información.

A partir del problema referenciado anteriormente se puede enunciar la siguiente **hipótesis**: si se logra implementar una herramienta de recuperación de información para el tratado con grandes cúmulos de datos de textos entonces se facilitará el manejo de los documentos jurídicos.

Como **objetivos específicos** y tareas de la investigación**:**

- Realizar una investigación sobre los aspectos teóricos y técnicos que se necesitan conocer para dar cumplimiento al objetivo trazado.
- Implementación del sistema propuesto.
- Validar el sistema propuesto mediante pruebas de Caja Negra y pruebas para Sistemas de Recuperación de Información (SRI).

Para darle cumplimiento a los objetivos trazados se ha decidido desarrollar las siguientes **Tareas de la investigación:**

- Valoración de la implementación de sistemas y herramientas similares.
- Caracterización de los diferentes modelos de recuperación de información.
- Diseño de una interfaz.
- Determinación de los estándares de codificación.
- Realización de los diagramas correspondientes a la implementación.
- Implementación de algoritmos de procesamiento de textos
- Validar la implementación.

Métodos científicos de investigación

Para el desarrollo de esta investigación se utilizarán métodos teóricos y empíricos.

Métodos teóricos

- **Análisis histórico-lógico:** Este método permitirá realizar un análisis de los elementos que se utilizan en la implementación de sistemas similares, dígase origen y evolución de los diferentes lenguajes de programación, Entornos de Desarrollo Integrado (en lo adelante IDE) y Sistemas Gestores de Base de Datos para determinar los más idóneos para desarrollar la aplicación.
- **Analítico - Sintético:** Este método permitirá después de realizar un análisis de las tendencias actuales en torno a la investigación para la gestión de actividades, determinar los principales métodos y algoritmos que son usados a nivel mundial y que nos pueden servir en la implementación del módulo.

Métodos empíricos

- **Entrevista:** Se realizaron múltiples entrevistas no formales a los clientes, líderes de proyecto, analistas en aras de obtener información referente al funcionamiento del posible sistema.

Posibles resultados:

Un prototipo de herramienta de recuperación de información usando modelos o herramientas que usen estos tipos de modelos para el desenvolvimiento dentro del ámbito jurídico.

Estructuración del contenido

El presente trabajo está conformado por 2 capítulos.

Capítulo 1. Fundamentación teórica:

Se realiza una comparación entre los lenguajes de programación para el desarrollo de aplicaciones similares, para la implementación del sistema, al igual que la argumentación de por qué el uso de una aplicación como esta sería de vital importancia. También se exponen conceptos y definiciones relacionados con el tema.

Capítulo 2. Descripción y análisis de la solución propuesta:

Descripción y análisis de la solución propuesta: Se describen algunos algoritmos a implementar y se analiza la complejidad de los mismos, así como las clases que modelan la solución y las principales funcionalidades de cada una de ellas. Finalmente se establecen los estándares de codificación a utilizar.

Capítulo 3. Validación de la solución propuesta:

Se refiere al diseño de las pruebas de unidad que permitan validar la solución propuesta, detallando de las mismas su objetivo, alcance y tipo, al igual que los valores utilizados para la ejecución de dichas pruebas.

CAPITULO 1 Fundamentación Teórica

En este capítulo es preciso una comparación de los distintos lenguajes de programación posibles a emplearse en la implementación de la herramienta de recuperación de información, tales como Java, C# y C++; al igual que los IDE de desarrollo de los mismos, además de las tecnologías utilizadas en la actualidad, describiendo ventajas y desventajas, así como sus características. Se explicaran brevemente conceptos y definiciones relacionados con los modelos de recuperación de información y sus aplicaciones; concluyendo con una explicación del por qué de la selección realizada para el desarrollo definitivo de la aplicación.

1.1. Aplicación de Escritorio vs Aplicación Web

Se denomina aplicación web a aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación de software codificado en un lenguaje soportado por los navegadores web (HTML, JavaScript, Java, ASP, PHP, etc.), en la que se confía la ejecución al navegador (Definicion.org).

Las aplicaciones Web, por ejemplo un programa de contabilidad, generan en forma dinámica páginas en formato HTML o XHTML. Generalmente cada página Web individual es vista por el cliente como un formulario o documento estático, pero la secuencia de páginas provee de una experiencia interactiva.

Entre algunas características se pueden ver (Masternewmedia):

- Desarrollo barato, sencillo y rápido.
- Acceso ubicuo, sin necesidad de distribución e, idealmente, con pocos requerimientos técnicos.
- Datos centralizados y fácil integración de datos de múltiples fuentes.
- Permiten el desarrollo de comunidades que dan valor a las aplicaciones (software social).
- Consumo de recursos para terceros: la mayor parte de consumo de ciclos de procesador, memoria, etcétera, ocurren en el servidor.

- Múltiples usuarios concurrentes**:** Las aplicaciones web pueden ser utilizada por múltiples usuarios al mismo tiempo.

Entre las desventajas podemos citar:

- Acceso limitado, la necesidad de conexión permanente y rápida a Internet hacen que el acceso a estas aplicaciones no esté al alcance de todos.
- La interactividad no se produce en tiempo real, en las aplicaciones web cada acción del usuario conlleva un tiempo de espera excesivo hasta que se obtiene la reacción del sistema.
- Diferencias de presentación entre plataformas y navegadores. La falta de estándares ampliamente soportados dificulta el desarrollo de las aplicaciones.

Por aplicaciones de escritorio se entiende toda aplicación que ha sido desarrollada para ser ejecutada en una plataforma específica, ya sea Windows, Linux ó Mac. El desarrollo sobre una plataforma, normalmente, implica que la aplicación no pueda ser ejecutada en otras, pero no significa una desventaja en la actualidad ya que el código puede ser recompilado en otro sistema operativo y mostrar el mismo resultado como por ejemplo: una aplicación hecha en C++ para Windows, se recompila el código en plataformas existentes para Linux sin ninguna necesidad de hacer grandes cambios ya que Linux también está hecho sobre C/C++ y se obtiene el mismo resultado, así como aplicaciones de .Net pueden ser corridas en Mono para Linux por solo mencionar algunos ejemplos.

Entre las ventajas que posee las aplicaciones de escritorios se encuentran las siguientes:

- Mayor capacidad gráfica visual.
- Menor tiempo de respuesta (aplicación más rápida) y mayor capacidad de corrección.
- Mayor personalización.
- Mayor rendimiento en cuanto a memoria que utilizan.

A pesar de esto las aplicaciones de escritorio poseen las siguientes desventajas:

- Diseminación de la información y lógica en muchas partes (cada computadora que la use).
- Traumas a la hora de realizar actualizaciones o correcciones al programa ya que las instalaciones están diseminadas.
- La desviación del uso, usándose cada vez más las aplicaciones Web.

1.2. Paradigmas de Programación

Un paradigma de programación es un modelo básico de diseño y desarrollo de programas, que permite producir programas con unas directrices específicas, tales como: estructura modular, fuerte cohesión, alta rentabilidad, etc. (JOYANES AGUILAR, 2003).

1.2.1 Paradigma Declarativo o Paradigma de Programación Lógica

Se basa en el hecho que un programa implementa una relación antes que una correspondencia. Debido a que las relaciones son más generales que las correspondencias (identificador - dirección de memoria), la programación lógica es potencialmente de más alto nivel que la programación funcional o la imperativa. El lenguaje más popular enmarcado dentro de este paradigma es el lenguaje PROLOG. El auge del paradigma declarativo se debe a que el área de la lógica formal de las matemáticas ofrece un sencillo algoritmo de resolución de problemas adecuado para usarse en un sistema de programación declarativo de propósito general (Zárate Rea, Noviembre de 2008).

1.2.2 Programación Orientada a Objetos

El paradigma orientado a objetos, se basa en los conceptos de objetos y clases de objetos. Un objeto es una variable equipada con un conjunto de operaciones que le pertenecen o están definidas para ellos. El paradigma orientado a objetos actualmente es el más popular y día a día los programadores, estudiantes y profesionales tratan de tomar algún curso que tenga que ver con este paradigma (Zárate Rea, Noviembre de 2008).

Alrededor de 1970 David Parnas planteó el ocultamiento de la información como una solución al problema de administrar grandes proyectos software. Su idea fue encapsular cada variable global en un módulo con un grupo de operaciones (al igual que los procedimientos y las funciones) que permitan tener un acceso directo a la variable. Otros módulos pueden acceder a la variable sólo indirectamente, llamando a estas

operaciones. Hoy se usa el término objeto para tales módulos o variables encapsuladas a sí mismas. Lenguajes imperativos como Pascal y C han sido modificados (o añadidos) para que soporten el paradigma orientado a objetos para dar Delphi en el caso de Pascal y C++ en el caso de C.

Una de las bondades importantes de los lenguajes orientados a objetos es que las definiciones de los objetos pueden usarse una y otra vez para construir múltiples objetos con las mismas propiedades o modificarse para construir nuevos objetos con propiedades similares pero no exactamente iguales. El lenguaje orientado a objetos por excelencia es Smaltalk[1] desarrollado en Palo Alto Research Center durante los 1970's.

¿Pero qué es exactamente un lenguaje orientado a objetos? Los siguientes conceptos señalan las características generalmente aceptadas acerca de los lenguajes orientados a objetos.

- Objetos y clases son obviamente los conceptos fundamentales. Una clase es un conjunto de objetos que comparten las mismas operaciones.
- Objetos (o al menos referencia a objetos) deben ser valores de la clase base. Así, cualquier operación puede tomar un objeto como un argumento y puede devolver un objeto como resultado. De esta manera el concepto de clase de objetos está relacionado con el concepto de tipo de dato.
- Herencia es también vista como un concepto clave dentro del mundo de los objetos. En este contexto, la herencia es la habilidad para organizar las clases de objetos en una jerarquía de subclases y superclases y las operaciones dadas para una clase se pueden aplicar a los objetos de la subclase.

Una vez analizado los distintos paradigmas de programación y ver el más apropiado para dar solución al problema planteado, se propone utilizar el paradigma de programación Orientado a Objetos y se hará un estudio del lenguaje de programación a utilizar, el cual depende en gran medida de la selección anterior.

[1] Primer sistema puro de objetos. Todo en Smalltalk es un objeto y toda la computación es desarrollada mediante mensajes que son enviados entre los objetos.

1.3 Lenguajes de Programación

Un lenguaje de programación es aquel elemento dentro de la informática que nos permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos de hardware y software existentes (Def10).

1.3.1 C++

El C++ es un derivado del mítico lenguaje C. Este lenguaje apareció en la década de los 70 de la mano de Dennis Ritchie para la programación en sistemas operativos Unix (El mejor ejemplo actual de un sistema operativo Unix es Linux), el cual surgió como un lenguaje generalista recomendado sobre todo para programadores ya expertos, ya que no llevaba implementadas muchas funciones que hacen a un lenguaje más comprensible. Sin embargo, aunque esto en un principio puede convertirse en un problema, en la práctica es su mayor virtud, ya que permite al programador un mayor control sobre lo que está haciendo. Años más tarde, un programador llamado Bjarne Stroustrup, creó lo que se conoce como C++. Necesitaba ciertas facilidades de programación, incluidas en otros lenguajes pero que C no soportaba, al menos directamente, como son las llamadas clases y objetos, conceptos muy en boga en la programación actual (Tucker, Allen B). Para ello rediseñó el C, ampliando sus posibilidades pero manteniendo su mayor cualidad, la de permitir al programador en todo momento tener controlado lo que está haciendo, consiguiendo así una mayor rapidez que no se conseguiría en otros lenguajes. De clases y objetos baste saber por ahora que consisten en un sistema que pretende acercar los lenguajes de programación a una comprensión más humana basándose en la construcción de objetos, con características propias solo de ellos, agrupados en clases.

¿Por qué programar en C++?

Algunas ventajas:

- Su increíble versatilidad.

- Con él pueden programarse desde los programas más simples, a los programas más complicados como incluso sistemas operativos.
- Es portable, es decir, un programa con el código escrito en C++, se podrá compilar en cualquier sistema operativo o sistemas informáticos sin necesidad de cambiar casi el código fuente.
- Es un lenguaje multinivel, es decir, se puede usar tanto para programar directamente el hardware (dependiendo del sistema operativo), como para crear aplicaciones tipo Windows definidas todas por poseer una misma interfaz.

Ventajas frente a Java:

- Al compilarlo, se genera código objeto, nativo de cada máquina. A la hora de elegir un lenguaje, la necesidad de velocidad de la aplicación inclinaría la balanza hacia C++.
- Es una extensión de C, por eso, muchos programadores encontrarán muy sencilla la transición, ya que podrán seguir haciendo cosas a la antigua usanza.
- Permite un control de la memoria y una capacidad de programación de bajo nivel impensable en Java.

1.4 Entornos de Desarrollo

Un entorno de desarrollo integrado o IDE (acrónimo en inglés de Integrated Development Environment), es un programa informático compuesto por un conjunto de herramientas de programación.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI) (apm10).

1.4.1 Borland C++Builder 6

Borland C++Builder 6 Personal constituye una vía muy rápida para desarrollar aplicaciones C++. Es un entorno de desarrollo probado, ofrece el entorno de desarrollo visual que miles de desarrolladores C++ buscan a la hora de crear sus aplicaciones. Se puede desarrollar rápidamente prototipos y aplicaciones completas, mediante una amplia paleta de componentes con más de 85 componentes reutilizables. Los conocimientos de programación quedan protegidos gracias al uso de código 100% estándar ANSI/ISO, sin extensiones propietarias que limiten la elección de futuras herramientas de desarrollo. Si dispone de código C++ existente, puede importarlo directamente en C++Builder. Tiene incluido Advanced Project Manager, una herramienta que le ayuda a controlar las fuentes y archivos utilizados en su proyecto. Puede visualizar sus aplicaciones y las distribuciones realizadas. Muchas de las herramientas que forman parte del entorno de desarrollo integrado han sido mejoradas para incrementar su productividad, incluyendo muchos Asistentes para la mayoría de tareas habituales. El compilador C++ incluido, Borland C++ Compiler 5.5, es un compilador y optimizador del código, de alto rendimiento y multihebrado, que actúa en segundo plano. Las aplicaciones se compilan y se ejecutan más rápidamente (Reisdorph, Kent).

- Usa el concepto de RAD: Rapid Application Develpment, Desarrollo Rápido de Aplicaciones.
- Gran cantidad de componentes incluidos en la distribución básica (más de 200 vs menos de 20 en un entorno Visual Basic 6).
- La librería VCL y el código generado usan código nativo lo que le permite ejecutarse con gran velocidad.
- El compilador de C++ es de reciente actualización, incluye soporte al estándar ISO C++98, C++0x y TR1.
- Soporta las reconocidas librerías Boost, que permiten reducir el costo de mantenimiento y son muy utilizadas por las diversas funcionalidades que brindan (estadísticas, concurrencia, procesamiento de imágenes, procesamiento de texto, etc.)

1.4.2 Visual Studio 2008

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE) de Microsoft. Puede ser utilizado para desarrollar aplicaciones de consola y gráfica de interfaz de usuario junto con aplicaciones de Windows Forms, sitios web, aplicaciones web y servicios web, tanto en código nativo con código administrado para todas las plataformas compatibles con Microsoft Windows, Windows Mobile, Windows CE, NET Framework,. NET Compact Framework y Microsoft Silverlight (Sons, 2008).

Visual Studio incluye un editor de código de IntelliSense (definida para el completamiento de código) de apoyo, así como la refactorización de código. El depurador integrado funciona tanto como un depurador a nivel de fuente y un depurador a nivel de máquina. Otras herramientas integradas incluyen un diseñador de formularios para crear aplicaciones GUI, diseñador web, diseñador de la clase, y el diseñador del esquema de base de datos. Acepta plugins que mejoran la funcionalidad en casi todos los niveles, incluyendo la adición de soporte para sistemas de control de código fuente (como Subversion y Visual SourceSafe) a la adición de conjuntos de herramientas nuevas, como los editores y los diseñadores visuales de lenguajes específicos de dominio o de conjuntos de herramientas para otros aspectos del ciclo de desarrollo de software (como el cliente de Team Foundation Server: Team Explorer).

Visual Studio es compatible con idiomas por medio de servicios de lenguaje, que permiten que el editor de código y un depurador para apoyar (en diversos grados) casi cualquier lenguaje de programación, con la condición de que un servicio específico del lenguaje exista. Aquí se incluyen C / C + + (a través de Visual C + +), VB.NET (a través de Visual Basic. NET), y C # (a través de Visual C #). El soporte para idiomas como F #, H, Python y Ruby, entre otros está disponible a través de los servicios de idiomas instalado por separado. También es compatible con XML / XSLT, HTML / XHTML, Javascript y CSS. Versiones lingüísticas específicas de Visual Studio también existen más servicios que ofrecen en algunos idiomas para el usuario. Estos paquetes individuales se llaman Microsoft Visual Basic, Visual J #, Visual C # y Visual C + +.

Ventajas:

- Código administrado: El Common Language Runtime (CLR) realiza un control automático del código para que este sea seguro, es decir, controla los recursos del sistema para que la aplicación se ejecute correctamente.
- Interoperabilidad multilenguaje: El código puede ser escrito en cualquier lenguaje compatible con .Net ya que siempre se compila en código intermedio (MSIL).
- Compilación just-in-time: El compilador JIT incluido en el Framework compila el código intermedio (MSIL) generando el código máquina propio de la plataforma. Se aumenta así el rendimiento de la aplicación al ser específico para cada plataforma.
- Garbage collector: El CLR proporciona un sistema automático de administración de memoria denominado recolector de basura (garbage collector). El CLR detecta cuándo el programa deja de utilizar la memoria y la libera automáticamente. De esta forma el programador no tiene por que liberar la memoria de forma explícita aunque también sea posible hacerlo manualmente (mediante el método disponse () se libera el objeto para que el recolector de basura lo elimine de memoria).
- Seguridad de acceso al código: Se puede especificar que una pieza de código tenga permisos de lectura de archivos pero no de escritura. Es posible aplicar distintos niveles de seguridad al código, de forma que se puede ejecutar código procedente del Web sin tener que preocuparse si esto va a estropear el sistema.
- Despliegue: Por medio de los ensamblados resulta mucho más fácil el desarrollo de aplicaciones distribuidas y el mantenimiento de las mismas. El Framework realiza esta tarea de forma automática mejorando el rendimiento y asegurando el funcionamiento correcto de todas las aplicaciones.

Pero no todo son ventajas:

- Procesos como la recolección de basura de .Net o la administración de código introducen factores de sobrecarga que repercuten en la demanda de más requisitos del sistema.
- El código administrado proporciona una mayor velocidad de desarrollo y mayor seguridad de que el código sea bueno. En contrapartida el consumo de recursos durante la ejecución es mucho mayor, aunque con los procesadores actuales esto cada vez es menos inconveniente.

- El nivel de administración del código dependerá en gran medida del lenguaje que se utilice para programar. Por ejemplo, mientras que Visual Basic .Net es un lenguaje totalmente administrado, pues C# permite la administración de código de forma manual, siendo por defecto también un lenguaje administrado. Mientras que C++ es un lenguaje no administrado en el que se tiene un control mucho mayor del uso de la memoria que utiliza la aplicación.

1.5 Recuperación de Información

Recuperación de Información (IR, Information Retrieval) es el término utilizado para referirse a la tarea de proporcionar información sobre la existencia de documentos relevantes para una petición de información. El conocimiento del área está suficientemente estructurado en libros como (Apmarin), (Witten, 1999) por lo que aquí se van a presentar de forma breve los conceptos más relevantes. La Recuperación de Información, en su acepción tradicional, se dirige idealmente a obtener todos y solo aquellos documentos relevantes para una consulta en una colección dada de documentos. La mayoría de los sistemas devuelven un ranking de los documentos de acuerdo a su grado de relevancia respecto a la consulta.

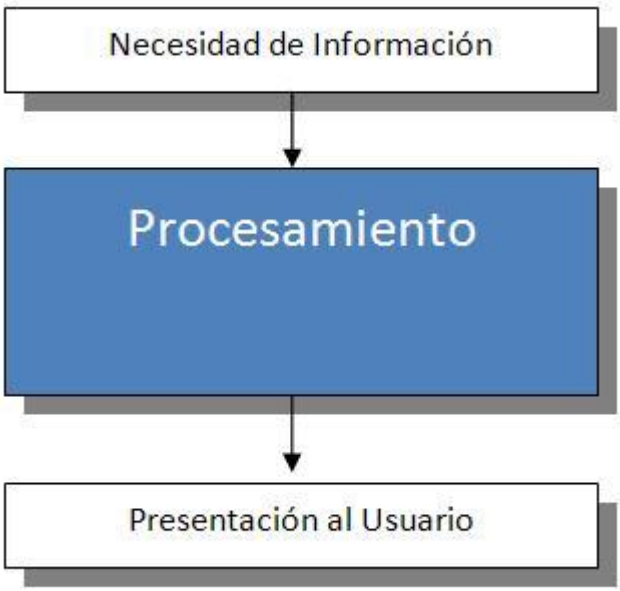


Imagen 1: Proceso de Recuperación de Información.

La tarea de Recuperación de Información se divide en tres subtareas:

1. Indexación, cuya finalidad es obtener una representación de los documentos que permita, de forma eficiente, su almacenamiento y comparación con las consultas.

2. Procesamiento de la consulta, para adecuarla a la representación interna de los documentos. En la mayoría de los motores de búsqueda actuales existe un lenguaje intermedio con operadores que permiten pesar y relacionar entre sí las palabras de la consulta.

3. Recuperación y ranking de documentos a partir de la comparación entre las representaciones internas de consulta y documentos.

También se puede entender como subtarea IR, no siempre presente en todos los buscadores, la posibilidad de retroalimentación (feedback) por parte de los usuarios al sistema. La selección de uno o varios documentos permite al sistema utilizar esta información para refinar la consulta (relevance feedback) (Introducción a los Sistemas de recuperación de Información, 2008-2009).

El objeto de la indexación es caracterizar los documentos para la tarea de recuperación. Idealmente, la indexación de un documento debería ser una representación eficiente del contenido semántico del documento original. Al igual que los índices que indican en un libro en qué pagina aparece una palabra, los índices en Recuperación de Información permiten localizar un término en un documento de una colección. Los términos de indexación pueden ser de dos tipos:

1. Términos de un vocabulario controlado (tesauros). En este caso se asignan al documento una serie de términos pertenecientes a un vocabulario que no tienen por qué estar contenidos en el documento. La asignación puede realizarse de forma manual, semiautomática o automática mediante sistemas entrenados.

2. Texto libre. La indexación de un documento se realiza con los términos más significativos extraídos automáticamente del documento. Estos términos usualmente son palabras del texto, pero nada impide que un procesamiento apropiado sea capaz de extraer elementos más complejos como nombres propios,

sintagmas, expresiones normalizadas para fechas y números, incluso la categoría o el sentido de las palabras.

Los tesauros llevan utilizándose mucho tiempo como vocabularios controlados respecto a los cuales indexar y recuperar documentos. Cuando se hizo posible trabajar con bases documentales extensas y considerar los textos completos, se comenzaron a desarrollar métodos de indexación automáticos basados en procesamiento lingüístico y estadístico sobre texto libre. Estas técnicas de búsqueda libre proporcionaron unos resultados que cuestionaron el uso de los tesauros para recuperación de información por el coste que supone construir un tesauro, mantenerlo y asignar sus términos a los documentos de la colección (Peña, 2002).

1.5.1 Información

La información se describe como un mensaje normalmente bajo la forma de un documento o algún tipo de comunicación audible o visible.

A diferencia de los datos, la información tiene significado (relevancia y propósito). Los datos se convierten en información cuando su creador les añade significado.

1.5.2 Conocimiento

El conocimiento es una mezcla de experiencia, valores, información y “saber hacer” que sirve como marco para la incorporación de nuevas experiencias e información, y es útil para la acción (Santos, 2008-2009).

1.5.3 Sistema

Sistema es un conjunto de objetos, cuya interacción produce la aparición de nuevas calidades interactivas, no inherentes a los componentes aislados que constituyen el sistema (Santos, 2008-2009).

1.5.4 Sistema de Información

Un conjunto de componentes interrelacionados que permiten capturar, procesar almacenar y distribuir la información para apoyar la toma de decisiones y el control en una institución (Santos, 2008-2009).

1.5.4.1 Tipos de Sistemas de Información

- Sistemas de Procesamiento de Transacciones.
- Sistemas de Trabajo con el Conocimiento.
- Sistemas de Oficina.
- Sistemas de Información Gerencial.
- Sistemas de Soporte a Decisiones.
- Sistemas de Soporte a Decisiones en Grupos.

En la siguiente imagen se representa de forma resumida la ocurrencia de sucesos de un Sistema de Información.

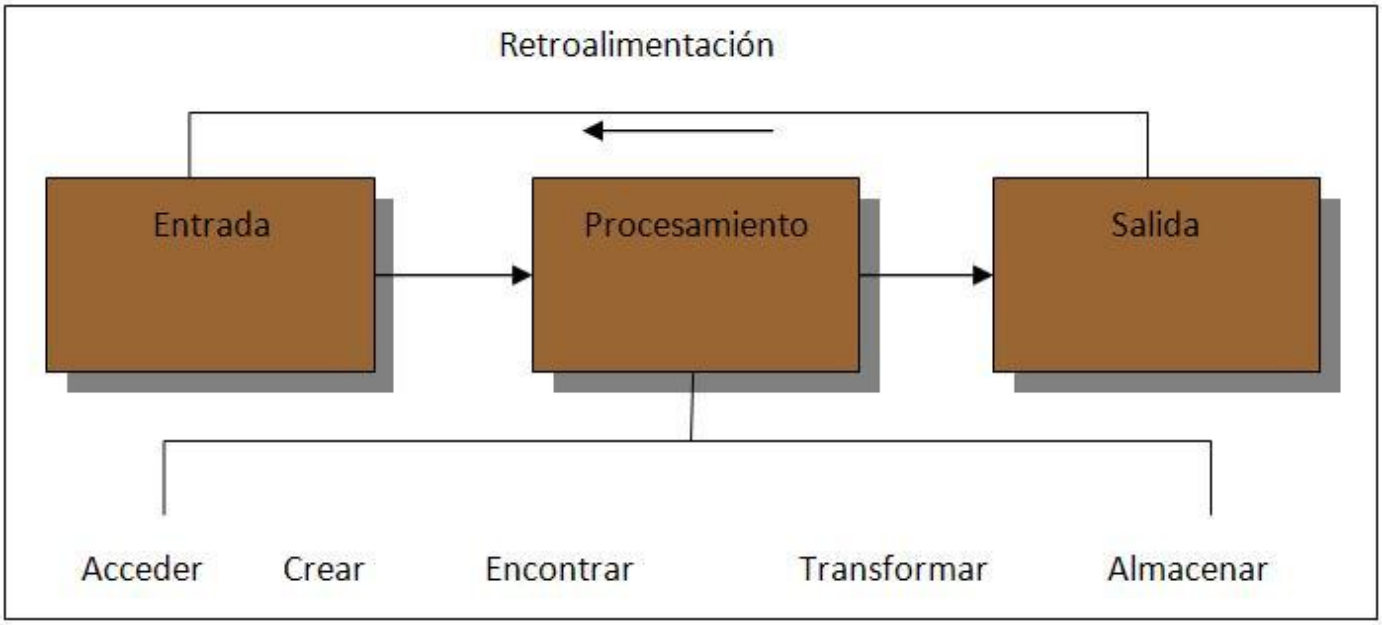


Imagen 2: Sistema de Información.

1.5.5 Operaciones Textuales

Los sistemas que en la actualidad se implementan con fines similares a este trabajo o sea basados en la implementación de Sistemas de Recuperación de Información, algunas de sus funcionalidades principales son las que se describen a continuación:

- Análisis léxico.
- Eliminación de las Stopwords.
- Proceso de stemming/lematización (reducción de las palabras a su raíz gramatical).
- Identificación de grupos de sustantivos (eliminación de adjetivos, adverbios y verbos).
- Construcción de diccionarios.

A continuación se muestra un esquema simple de los procesos antes descritos:

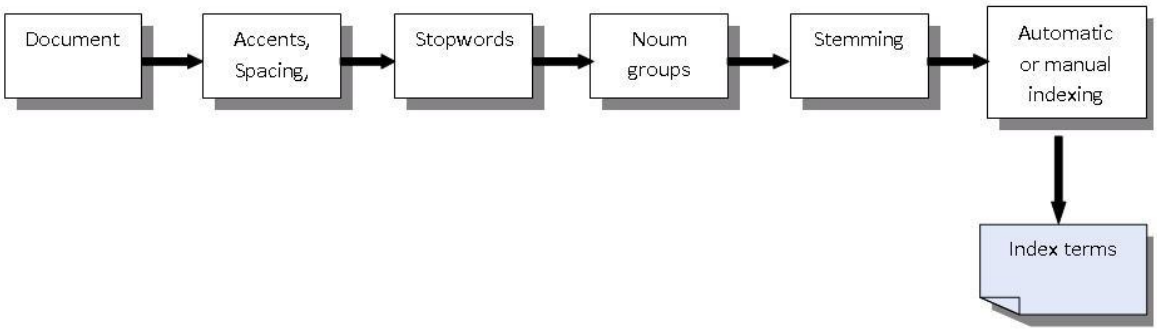


Imagen 3: Construcción de la Representación lógica de Documentos

1.5.6 Modelos de Recuperación de Información

1.5.6.1 ¿Qué es un modelo?

- Es la primera etapa para abordar el tema de la RI.
- Representación matemática para resolver el problema de la RI.
- Un buen modelo debe de “adivinar” lo que el usuario quiso preguntar.

- Los modelos son utilizados para el cálculo de la relevancia.
- Para construir un modelo:

    - Analizar las representaciones de documentos y consultas.
    - Concebir el marco en el que pueden ser representadas.
    - Construcción de función de ranking.

1.5.6.2 Definición formal de Modelo de RI

Un modelo de recuperación de información es un cuádruplo [D, Q, F, R (qj, dj)] donde (Santos, 2008-2009):

D es un conjunto de vistas lógicas o representación de los documentos de la colección.

Q es un conjunto compuesto por vistas lógicas o representación de las necesidades del usuario. Estas representaciones son denominadas consultas.

F es un framework para modelar las representaciones de los documentos, consultas y sus relaciones

R es una función de ranking que asocia un número real con una consulta qj que pertenece a Q y una representación de documento dj que pertenece a D. Este ranking define un orden entre los documentos de acuerdo a la consulta.

1.5.6.3 Modelos

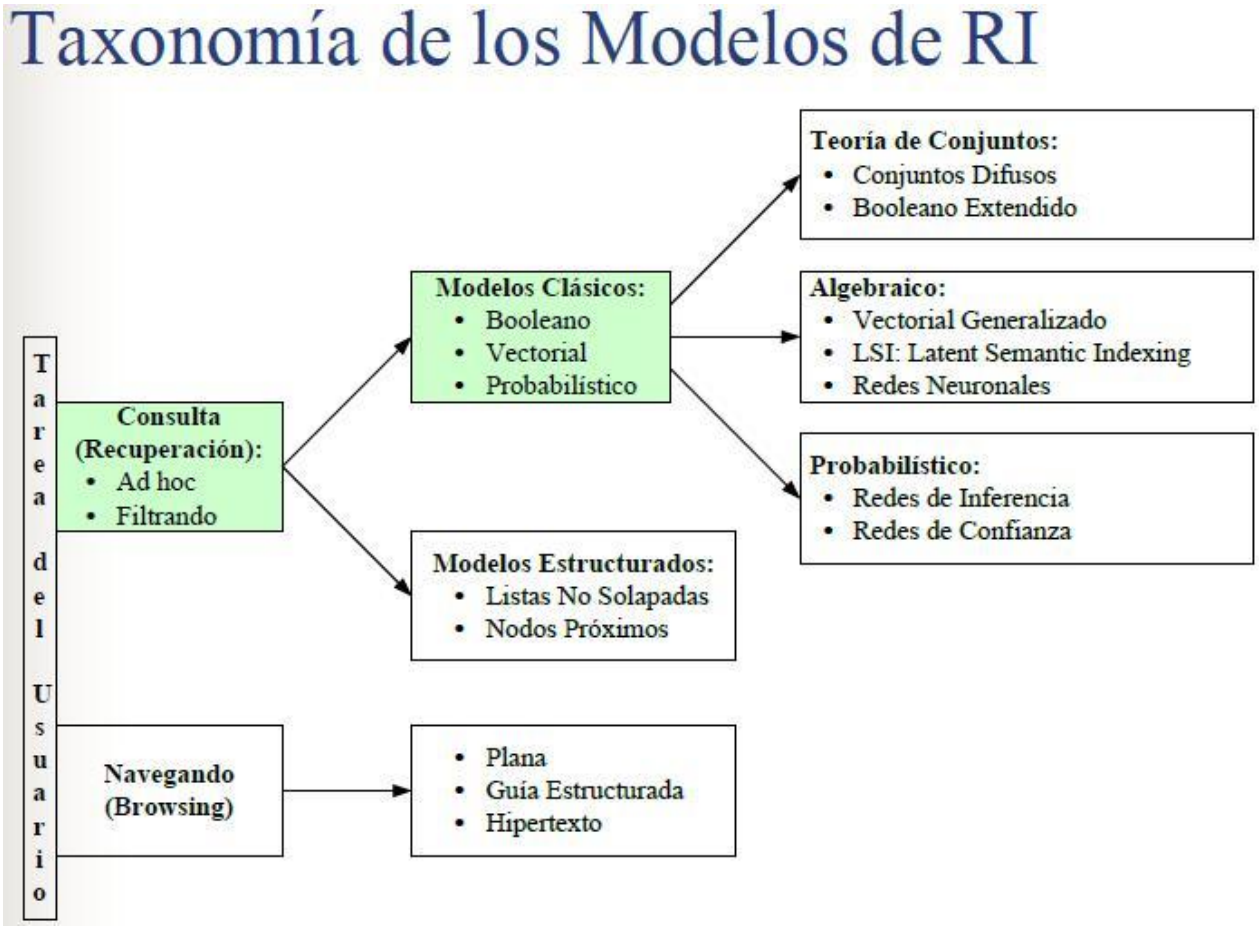


Imagen 4: Taxonomía de los Modelos de RI.

1.5.6.4 Modelos Clásicos

Los modelos clásicos de recuperación de información consideran que cada documento está descrito por un conjunto de términos indexados, en la cual la relevancia de un término de cada documento está dado por (Santos, 2008-2009): **Peso** *wij.*

1.5.6.4.1 Peso

Definición:

Sea *n* la cantidad total de términos indexados en el SRI y *ki* un término indexado cualquiera. *K = {k1,k2, k3, ..., kn}* es el conjunto de todos los términos indexados. Un peso *wij* es el valor real positivo asociado a cada término en el documento (Santos, 2008-2009).

- Si el término *ki* no aparece en el documento *j* entonces *wij* = 0.
- dj = (*w1j, w2j, w3j, ..., wnj*).

- Los documentos se describen a través de un conjunto de términos representativos llamados índices o términos índice.
- Se pueden considerar todos los términos como importantes en una aproximación llamada full text.
- No todos los términos son igualmente importantes. El proceso de decidir la importancia de un término se puede realizar a través de la asignación de pesos.
- Todos los modelos clásicos tienen ciertas desventajas comunes, la más notoria es la incapacidad para capturar las relaciones entre términos, ya que los términos son independientes.

1.5.6.5 Modelos Alternativos

- Se extiende de los modelos clásicos.
- Son en general bastante costosos de implementar, y no siempre dan grandes resultados.
- Por ello, son en general poco populares, con la excepción del LSI y, las Redes Neuronales y Bayesianas.

1.5.6.6 Modelos Estructurados

- Tratan de combinar la información del contenido del texto con la estructura del texto.
- Se pierde la noción de relevancia, y estamos ante una recuperación de datos.

- Ejemplo:

- Un usuario tiene mucha memoria visual. Recuerda un documento donde aparece 'holocausto atómico' en cursiva, cerca de una imagen que tiene en la etiqueta la palabra 'tierra'. **same-page(near('holocausto atómico', Figure(etiqueta('tierra')))**
- Se recuperarán aquellos documentos que satisfagan exactamente la consulta, por tanto no hay orden de relevancia en los resultados (Modelos de RI).

1.5.6.7 Modelos de Navegación

- No se basan en consultas del usuario, sino que el usuario navega a través de una jerarquía hasta encontrar los documentos relacionados a lo que anda buscando.

- Son una guía jerárquica de directorios que va de los temas más generales a los más particulares. Listan lugares (URLs) y los clasifican en categorías, además de añadir comentarios identificativos sobre ellos.

- Su objetivo es encontrar los documentos que pertenezcan al área temática seleccionada.

- Están compuestos por dos partes:

- La BD que es construida por los URLs remitidos.

- Una estructura jerárquica que facilita la consulta a la base.

- Al conectar con algún buscador nos encontraremos con una página que contiene una estructura jerárquica de temas, es decir, hay un grupo de temas generales, al seleccionar uno nos sale otro grupo de temas dependiente (cada vez más específico) del que nos llevó allí, y podemos seguir así hasta que localicemos el tema de nuestro interés o se acaben las categorías creadas por el autor del buscador (Modelos de RI).

1.5.6.8 Especificación de Modelos

1.5.6.8.1 Modelo Booleano

El Modelo Booleano es un modelo basado en la teoría de conjuntos y el algebra Booleana. Para este modelo se definen varios aspectos fundamentales entre los que se encuentran (Tucker, Allen B):

- Los documentos: Conjunto de términos.
- La consulta: Expresiones Booleanas sobre los términos.
- **D**: Conjuntos de palabras (términos indexados).
- **Q**: Expresión Booleana de los términos indexados usando los operadores AND, OR, NOT.
- **F**: Algebra Booleana sobre conjuntos de términos y conjuntos de documentos.
- **R(qi, dj)**: Función booleana que indica si el documento *dj* satisface la consulta *qi*.

Este modelo entre sus principales características presenta:

- Modelo muy simple basado en conjuntos.
- Fácil de entender e implementar.
- Solo recupera los documentos donde la coincidencia sea exacta.
- No se realiza ranking de los documentos dado que R es una función booleana.
- Recupera mucho o muy poco.
- Todos los términos son igual de importante.

1.5.6.8.2 Modelo Vectorial

El modelo vectorial, en lugar de predecir si un documento es pertinente o no, da un *ranking* de los documentos de acuerdo con su grado de similitud a la consulta.

Para la recuperación de los documentos, puede establecerse un umbral de similitud y recuperar los documentos cuyo grado de similitud sea mayor que este umbral.

Consideraciones para ese modelo:

"Un término que aparece en muchos documentos no debe considerarse como más importante que uno que aparece en pocos documentos."

"Un documento con muchas ocurrencias de un término no debe considerarse como menos importante que un documento con pocas ocurrencias de ese término."

TF (Term Frequency):

Frecuencia de ocurrencia de un término *ki* dentro de un documento *dj*.

Dado que:

freqij: cantidad de ocurrencias del término *ki* en el documento *dj*.

$$tf_{i,j} = \frac{freq_{i,j}}{max_l freq_{l,j}}$$

$$tfij = K + \frac{(1-K)freqiq}{\max freqlq}$$

El termino K se incluye en la formula con el objetivo de optimizar el resultado, teniendo en cuenta que:

- se ha comprobado que para K = 0.5 se obtienen buenos resultados.

IDF (Inverse Document Frequency):

Frecuencia de ocurrencia de un término *ki* dentro todos de los documentos de la colección.

Dado que:

N: número de documentos en el sistema.

ni: número de documentos en los cuales aparece el término *ki.*

$$idf_i = \log \frac{N}{n_i}$$

Peso:

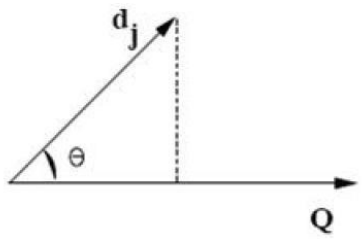
Definición de la ecuación para el peso:

*Wij = tfij * idfij*

$$w_{i,q} = (0.5 + \frac{0.5 freq_{i,q}}{max_l freq_{l,q}}) * \log \frac{N}{n_i}$$

Similitud o R(qi, dj):

La similitud se calcula utilizando el coseno del ángulo formado entre los vectores de la consulta y el documento.



$$sim(\vec{d_j}, \vec{q}) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|}$$

$$sim(\vec{d_j}, \vec{q}) = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sum_{i=1}^{t} w_{i,j}^2 \times \sum_{i=1}^{t} w_{i,q}^2}$$

Imagen 5: ángulo de similitud entre dj y Q.

Definición de términos:

D: Vectores de pesos de palabras (términos indexados).

Q: Vectores de pesos.

F: Espacio t-dimensional y operaciones entre vectores del álgebra lineal.

R:

$$sim(\vec{d_j}, \vec{q}) = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sum_{i=1}^{t} w_{i,j}^2 \times \sum_{i=1}^{t} w_{i,q}^2}$$

Conclusiones:

- Se hace ranking de los documentos.
- Recupera documentos con coincidencia parcial.
- Hay diferenciación entre los pesos de los términos de un documento dado por su relevancia.
- Asume la independencia entre términos.
- Cálculo de pesos de forma intuitiva.

M.Booleano vs M.Vectorial

Criterios	Booleano	Vectorial
Framework	Teoría de conjuntos y álgebra booleana	Espacio t-dimensional y operaciones entre vectores del álgebra lineal.
Pesos	Binarios	dados por tf*idf
Consultas	Expresiones booleanas	vectores de pesos no binarios
Documentos	vectores binarios	vectores de pesos no binarios
Similitud	Sim = {0,1}	Sim varía de 0 a 1 dada por el coseno del ángulo
Dependencia entre términos	No	No
Macheo parcial	No	Sí
Ranking	No	Sí
Simplicidad/costo implementación	Mayor / menor	Menor / mayor

Imagen 6: Comparación del MV y el MB.

Acorde con las ventajas y definiciones antes expuestas se acordó apoyar la implementación del sistema utilizando el Modelo Vectorial antes mencionado por su gran robustez, capacidad y usabilidad.

1.5.6.8.3 Modelo Booleano Extendido

Fue presentado por Salton, Fox y Wu en 1983. Es una extensión del modelo booleano tradicional, que agrega la funcionalidad de agregar pesos a cada término en los documentos, con la capacidad de formular consultas con operadores booleanos (Santos, 2008-2009).

Idea fundamental:

- Toma la idea de la representación de los documentos que se planteó en el modelo vectorial clásico.
- Toma la idea de las consultas como expresiones booleanas utilizada en el modelo booleano.

Peso:

Los pesos serán calculados de manera equivalente al modelo vectorial, normalizando el factor idf para acotarlo en el intervalo [0,1]:

$$w_{i,j} = f_{x,j} \times \frac{idf_x}{max_i idf_i}$$

Ejemplo:

- Toma la idea de la representación de los documentos que se planteó en el modelo vectorial clásico.
- Toma la idea de las consultas como expresiones booleanas utilizada en el modelo booleano.

Conclusiones sobre el Modelo Booleano Extendido (BE):

- Modelo poderoso híbrido entre el Vectorial y el Booleano.
- Algo complejo de implementar.
- Es uno de los más utilizados por sistemas RI.
- Permite obtener ranking aunque la no distributividad de las operaciones afecta o hace complejo el cálculo del ranking(Santos, 2008-2009):

$q1 = (k1 \vee k2) \wedge k3$
$q2 = (k1 \wedge k3) \vee (k2 \wedge k3)$
$sim(q1,dj) \neq sim(q2,dj)$

Definiendo que entre los modelos de RI es uno de los más complejos, difícil de implementar y a la vez difícil de entender.

1.5.7 Lucene

Lucene es un API de alto rendimiento, que presenta una escalable Recuperación de Información (IR) de la biblioteca. Permite agregar capacidades de indexación y búsqueda en sus aplicaciones. Lucene es una herramienta muy utilizada, libre, proyecto de código abierto implementado en Java, es un miembro de la popular familia de proyectos Apache Jakarta, bajo la licencia Apache liberales. Como tal, Lucene es actualmente la librería de IR de Java más popular [5].

Lucene proporciona un sencillo pero potente núcleo API que requiere una comprensión mínima de la indexación de texto completo y búsqueda. Se necesita aprender acerca de sólo un puñado de sus clases con el fin de comenzar a integrar en Lucene una aplicación.

Uno de los factores clave detrás de la popularidad y el éxito de Lucene es su simplicidad. La exposición cuidadosa de su indexación y búsqueda de la API es una muestra del software bien diseñado. En consecuencia, no es necesario un conocimiento profundo acerca de cómo Lucene realiza el indexado y el trabajo de recuperación con el fin de comenzar a usarlo (Hatcher).

Lucene se encuentra actualmente implementado en más de un lenguaje de programación, lo que proporciona y ratifica su reusabilidad, así como son C++, Java y C# entre otros.

1.5.7.1 Comparación de Lucene con Lémur, Terrier y Xapian

- Lucene es multiplataforma, al igual que Lemur, las demás tecnologías no.
- Terrier y Lucene son implementados en Java, Lemur y Xapian en C++, aunque todas tienen soporte para otros lenguajes de programación.
- Todas indizan diferentes formatos de texto como: PDF, WORD, HTML, HTM, TXT, XML, RTF, entre otras.
- Lucene permite Stemming para varios idiomas, las demás tecnologías también.
- Lucene permite búsqueda mientras se actualiza el índice, lo que otras tecnologías no hacen.
- Lemur y Lucene permiten la indización incremental, Xapian y Terrier no.

- Todas trabajan con modelos probabilísticos, excepto Lucene que trabaja con el modelo de espacio vectorial.
- Todas las tecnologías son OpenSource (Software Libre).
- Lucene permite búsqueda por cualquier campo, las demás tecnologías no.

Cabe mencionar que es de suma importancia conocer que existen otras tecnologías aparte de Lucene por eso en este apartado se muestran las ventajas que una u otra pueden tener. También es importante conocer qué ambientes de desarrollo abarcan a Lucene. Además de Java existen otros lenguajes donde se implementa los cuales se explican en la siguiente sección.

Después del estudio realizado al respecto de las diferentes variantes actuales en el desarrollo de aplicaciones de recuperación de información, dígase lenguajes de programación, frameworks, entornos de desarrollo, se concluye que para llevar a cabo la implementación del sistema, se empleará el lenguaje de programación C++ en su versión para Visual C++, utilizando el entorno de desarrollo Visual Studio 2008 para VC++, por las facilidades que proveen. Todas estas herramientas tienen en común que son altamente calificadas y de alto rendimiento, además de que el componente principal utilizado es multiplataforma y de código abierto.

CAPITULO 2 Descripción y Análisis de la Solución Propuesta.

El presente capítulo se centra en la implementación de la aplicación que se propone como solución con este trabajo de diploma, donde se estudiarán los cambios necesarios para la transición del diseño a la implementación, teniendo en cuenta los requerimientos de la aplicación. Se abordarán cuestiones referentes a componentes posibles utilizados, dígase clases o librerías previamente implementadas, al igual que las estrategias de integración. Posteriormente se describirán los algoritmos no triviales a implementar, realizando un análisis de la complejidad de los mismos, para finalmente describir las nuevas clases y operaciones necesarias para darle solución a la situación problemática.

2.0 Análisis de posibles implementaciones, componentes o módulos ya existentes y que puedan ser rehusados.

Visual Studio.Net 2008

En el capítulo anterior se expuso que por sus características y potencialidades, se decidió emplear, para la implementación del sistema, el ID o Plataforma de Desarrollo de C# Visual Studio.Net 2008. Entre las características que lo describen también es fácil de utilizar y da la posibilidad de implementar cualquier tipo de herramienta, desde las más sencillas hasta las más complejas, desde aplicaciones de consola hasta aplicaciones graficas y de escritorio.

Framework Class Library (FCL):

Librería de clases que proporciona una gran cantidad de servicios:

Entrada/Salida, XML, ADO.NET (acceso a Bases de datos), Windows.Forms (aplicaciones gráficas), sockets, colecciones, threads....

2.0.1 Descripción de Librerías y/o Componentes Usados.

Otras librerías que se investigaron para su reutilización son PDFBox-0.7.2, IKVM.Runtime, Lucene, que brindan un conjunto de funcionalidades para la gestión de documentos en Texto plano, DOC, PDF

(Portable Document File) entre otros, para el manejo de textos en cuanto al proceso de indexación y otras funcionalidades.

2.0.2 TextMining.org

Esta es una de las librerías que permite extraer el texto de los documentos Word component (.doc).
Sus funciones son altamente confiables y a la vez presentan mejoras con respecto a otras librerías existentes, como por ejemplo con respecto a POI (JOYANES AGUILAR, 2003).
Ventajas frente POI:

- Optimización de extracción de textos mientras que POI no.
- Soporta la extracción de textos de Word 6/95 mientras que POI no lo hace.
- No toma en cuenta los textos que se encuentra invisibles o no son de importancia en el documento para el proceso de revisión, POI no incorpora esta funcionalidad.

2.0.3 PDFBox-0.7.2:

Es una librería sencilla, programada en Java, que permite trabajar con PDF de manera rápida y simple. Entre otras funciones, permite la creación de nuevos documentos, manipularlos o extraer distintos contenidos de ellos. Muy práctica, es gratuita y está disponible en distintos idiomas (SourceForge).

Entre las funcionalidades que nos ofrece esta librería nos encontramos:

- Elección de la unidad de medida, formato de página y márgenes.
- Gestión de cabeceras y pies de página.
- Salto de página automático.
- Salto de línea y justificación del texto automáticos.
- Omisión de imágenes (JPEG, PNG y GIF (versión 1.6)).
- Enlaces.

2.0.4 IKVM.Runtime

IKVM.NET es una implementación de Java para Mono y Microsoft. NET Framework. Incluye los siguientes componentes (SourceForge):
Una máquina virtual de Java implementada en. NET.

Herramientas que permiten a Java y. NET la interoperabilidad.

2.0.5 Lucene

Además de lo ya antes mencionado decir que Lucene es un API de código abierto para recuperación de información, originalmente implementada en Java por Doug Cutting (Hatcher). Está apoyado por el Apache Software Foundation y se distribuye bajo la Apache Software License. Lucene tiene versiones para otros lenguajes incluyendo Delphi, Perl, C#, C++, Python, Ruby y PHP.
Es útil para cualquier aplicación que requiera indexado y búsqueda a texto completo. Lucene ha sido ampliamente usado por su utilidad en la implementación de motores de búsquedas. Por ello, a veces se confunde Lucene con un motor de búsquedas con funciones de "crawling" y análisis de documentos en HTML incorporadas.
El centro de la arquitectura lógica de Lucene se encuentra el concepto de Documento (Document) que contiene Campos (Fields) de texto. Esta flexibilidad permite a Lucene ser independiente del formato del fichero. Textos que se encuentran en PDFs, páginas HTML, documentos de Microsoft Word, así como muchos otros pueden ser indexados mientras que se pueda extraer información de ellos.
Más resumidamente se puede decir que Lucene presenta características como:

- Lucene es un API de desarrollo para indización y búsquedas, escrita en Java.
- Está disponible en C++, Perl, C# y Ruby.
- Multiplataforma.
- Permite indización incremental.

- Algoritmos de búsquedas fiables y confiables.
- Permite ordenar resultados por relevancia.
- Lenguaje de consulta.
- Stemming.
- Búsqueda por campos, rangos de fecha, entre otras.
- Ordenación por cualquier campo.
- Permite búsqueda mientras se actualiza el índice.
- Lucene soporta la indización de documentos con formato: TXT, PDF, DOC, RTF, XML, PPT y HTML.

2.1 Descripción de las funcionalidades básicas. Análisis y complejidad de las mismas. Descripción de clases del sistema.

2.1.1 Funcionalidad básica de Lucene (Indización y Búsqueda)

A continuación de se explicara más detalladamente los dos procesos por si solos, ya que indización y búsqueda son dos objetivos muy generales (Hatcher).

2.1.1.1 Concepto de Indización

Cuando se requiere hacer uso de búsquedas dentro de una aplicación, rápido viene a la mente crear un programa que haga esto, es decir, que busque en todos los archivos palabras o frases relacionadas, esto tendría fallas en el caso de archivos muy grandes. Por eso es importante crear los índices, transformar el texto en un formato donde la búsqueda sea más rápida, eliminando el proceso de exploración lento. Este proceso de conversión es llamado indización y al archivo resultante se le llama índice. Un índice separa las palabras el documento en campos y permite el acceso rápido a los datos que fueron almacenados en el proceso de indizado (Hatcher).

2.1.1.2 Concepto de búsqueda

La búsqueda es el proceso de entrar al índice y buscar palabras relacionadas, para encontrar documentos donde aparezca. Es importante para la búsqueda tomar en cuenta dos factores: la destitución y la precisión. La destitución se encarga de indicar que documentos son relevantes a la búsqueda mientras que la precisión se encarga del filtrado de los datos (Hatcher).

2.1.1.3 Clases básicas en la indización

Las clases que se muestran a continuación son las principales durante el proceso de indización, para ello se definen cada una de ellas y el uso que tienen en Lucene.

- IndexWriter
- Directory
- Analyzer
- Document
- Field

2.1.1.3.1 IndexWriter

Es el componente central del proceso de indización. Esta clase crea índices y agrega documentos a uno ya existente. IndexWriter es un objeto que permite acceder al índice pero no leer o buscar en el.

2.1.1.3.2 Directory

La clase Directory representa la ubicación de un índice en Lucene. Esta a su vez utiliza subclases FSDirectory para guardar los índices en el sistema de archivos. Esta es la clase que más se usa para el almacenamiento de índices.

La clase IndexWriter hace uso de FSDirectory cuando necesita recibir como parámetro el directorio donde se almacenarán los índices.

Otra subclase llamada RAMDirectory, a diferencia de FSDirectory, esta se usa para almacenar los índices en memoria es recomendable cuando se crean índices pequeños o si se realizan pruebas de indización o búsqueda (Hatcher).

2.1.1.3.3 Analyzer

Antes de indizar un documento este pasa por la clase Analyzer. Esta clase elimina del documento palabras que no ayudan o distinguen un documento de otro como él, la, en, una, entre otras. También convierte las palabras a minúsculas para que las búsquedas sean más exactas.

2.1.1.3.4 Document

Una clase Document representa una colección de campos. El documento a indizar es separado en campos o en metadatos como son el título del documento, fecha de modificación, autor, entre otras. Estos se guardan en archivos diferentes cuando se indizan. Cuando se hace referencia a un documento se refiere a todo archivo que contenga texto como HTML, PDF, XML, entre otros.

2.1.1.3.5 Field

Cada documento contiene uno o más campos, en Lucene existen 4 métodos Field diferentes:

1. Keyword: Se almacena y se indiza tal cual, no se analiza, se utiliza para los campos que necesitan guardarse en el índice sin modificaciones como el número de seguridad social, los sitios de internet, directorio donde se encuentra el documento, entre otras.

2. UnIndexed: Se almacena pero nunca se usa en las búsquedas, como las llaves primarias en una base de datos.

3. Text: el valor se analiza e indiza, el valor original también se almacena.

4. UnStored: Es la opuesta a UnIndexed. Se analiza e indiza, se utiliza para todos los documentos de texto o sitios web donde solo se requiera guardar titulo y contenido.

2.1.1.4 Clases básicas para la Búsqueda

Para llevar a cabo una búsqueda con Lucene es importante familiarizarse con las siguientes clases (Hatcher):

- IndexSearcher
- Term
- Query
- TermQuery
- Hits

2.1.1.4.1 IndexSearcher

IndexSearcher es en la búsqueda lo que IndexWriter es en la indización. Es la clase principal que abre el índice para buscar en él, ofrece varios métodos de búsqueda, lo que hace esta clase es pasar como parámetro la query y regresar un objeto hits.

2.1.1.4.2 Term

Un término es la unidad básica para la búsqueda. Similar al objeto Field, consiste de un par de elementos: el nombre del campo y su valor. Por ejemplo en la siguiente figura 2 se busca la palabra Lucene en el contenido del documento. Un ejemplo sería:

Query q = new TermQuery(new Term("contents", "lucene"));

Hits hits = is.search(q);

2.1.1.4.3 Query

Lucene tiene diferentes subclases de Query, la más utilizada es TermQuery por los métodos que ella contiene.

2.1.1.4.4 TermQuery

Es el tipo de Query mas básico soportada por Lucene, se utiliza para hacer coincidir documentos que tienen valores específicos.

2.1.1.4.5 Hits

La clase Hits almacena los puntos de referencia a los resultados de la búsqueda, es decir todos los documentos encontrados que se relacionan con la Query. Vistas las clases principales, es importante ver realmente la funcionalidad de esta tecnología por lo cual a continuación se mostrará un ejemplo de cómo utilizar Lucene.

2.1.1.5 Creación de un Índice con CLucene

La creación de un índice constituye el punto de partida para el trabajo con Lucene, puesto que una vez que es creado, se irán añadiendo todos aquellos documentos que serán indizados.

A continuación se muestra un pequeño código de cómo crear el índice con CLucene (Hatcher)(Vale aclarar que las especificaciones del libro de Lucene que se está referenciando, los ejemplos de código descrito en el mismo están en java, pero la idea de las clases y métodos es lo importante):

```
File *indexDir = new File(("c:\\indexdir");

IndexWriter  *writer;

writer = new IndexWriter(indexDir, null, true);

writer.close();
```

Bien se podría haber creado una cadena (string) que guardara la dirección para el índice en sustitución de (File *indexDir = new File(("c:\\indexdir");), o se podía haber capturado como dato de entrada.

La clase IndexWriter se utiliza tanto para la creación de índices como para su mantenimiento. Cuando se crea un objeto de esta clase, como se observa en el ejemplo, al constructor se le pasan tres parámetros. De los cuales, solamente son relevantes el primero y el tercero: el primero representa el path (directorio) dónde se almacena el índice y con el valor del tercero establecido a true, indicamos que lo que estamos es creando el índice y no abriéndolo para su mantenimiento. El segundo de los parámetros se establece a un valor null. El método close () se llama para liberar todos los recursos asociados a la creación del índice.

2.1.2 Algoritmos no triviales

Los algoritmos que se analizan a continuación tienen un nivel de complejidad superior a la media, por lo que se hace necesario su estudio debido a que brindan la solución a problemáticas claves en la implementación del sistema.

Indexar Documentos:

Consiste en pasar por parámetros la dirección donde se encuentran los documentos (.txt), el sistema deberá ser capaz de recoger documento por documento, hacer el proceso de indexación, así como crear los índices de los documentos.

Realizar Consulta:

Consiste en que un usuario realice una consulta sobre la aplicación y la aplicación deberá ser capaz de especificar el tipo de consulta, tratarla y prepararla para el proceso de RI.

Respuesta de la Consulta:

En este caso el sistema deberá de ser capaz de mostrar los documentos según la consulta por su importancia, relevancia entre otras cualidades que deberá tomar en cuenta.

2.1.3 Estándares de Codificación.

Las convenciones o estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. A continuación se presenta la propuesta de estándar de codificación a utilizar para el desarrollo de este trabajo.

Comentarios:
Cada archivo cabecera comenzará con un comentario que incluya:

- Nombre.
Especifica el nombre del archivo.

- Descripción.
Se hace una pequeña descripción de las clases o funcionalidades que contiene el archivo.

- Autor.
Especifica el nombre del autor.

- Referencia.
La referencia es opcional, aquí se especifica la fuente de la idea original.

ej:
/* ----------
*

```
 * Nombre: ArchivoCabecera.h
* ----------------------------------------------------------------------
* Desc: Descripción.
 *
* Autor: Juan Pérez.
*
* Referencia: Thinking in C++.
*
* ----------------------------------------------------------------------
*/
```

Cada variable debe contener una pequeña descripción del objetivo de la variable.

```
 ej:
 // Almacena un entero.
 int entero;
```

Cada función debe contener una pequeña descripción del objetivo de la función.

```
ej:
 // Retorna un entero.
 int GetInt(){return entero;}
```

Nombre de identificadores:

Se considera como identificador a los nombres de variables (arreglos, matrices, apuntadores), funciones, así como cualquier tipo de dato definido por el usuario (estructura, clase). Dichos identificadores deberán seguir las siguientes normas.

Identificadores de variables:

Los nombres de las variables comienzan con un identificador del tipo de dato al que correspondan, en caso de que sean variables miembros de una clase, se le antepone en identificador m en minúscula seguido del tipo de dato al que corresponde, (“mi_” par una variable miembro de tipo integer). En caso de ser argumento de funciones se le antepone el prefijo “arg_” seguido del identificador en minúscula.

ej:

int mi_Variable; //variable miembro de una clase

float mf_Variable; //variable miembro de una clase

Instancias y tipos creados:

A las instancias y tipos creado se le antepone, como identificador, la letra k minúscula.
En caso de los tipos enumerativos se le antepone la letra e como identificador.

ej: MyEnumerated **e**Name; //tipo enumerativo

ClassName **k**ObjectName; // objeto de una clase

ClassName* **pk**Name; // puntero a objeto

ClassName* **mk_**Name; // variable miembro de una clase

Métodos:

En el caso de los métodos, se les antepone el identificador del tipo de dato de devolución, y en caso de no tenerlo (void), no se les antepone nada. Los constructores y destructores, como lo exigen los compiladores, llevan el nombre de la clase.

Métodos de acceso a miembros:

Los métodos de acceso a los miembros de las clases se nombrarán “Get” seguido del identificador, para el caso de retorno de valores y “Set” en caso de la actualización de valores.

En este trabajo se decidió colocar los comentarios encima de la línea a la que se le quiera aplicar y encima de la línea cabecera de los bloques. La Indentación se hará al nivel de la línea en cuestión. Se utilizarán en funciones y clases. Se puede utilizar en algoritmos no triviales y secciones de diferentes contextos dentro de los métodos.

Con todo lo antes mencionado y propuesto se ha podido arribar a un diseño acorde con las restricciones del sistema, que deben tenerse en consideración a la hora de implementarlo.

También se realizó un análisis a un conjunto de librerías y clases previamente implementadas a reutilizar, que repercuten en la dinámica de implementación porque brindan facilidades que es solamente cuestión de utilizar, entre ellas la estructura del framework de .NET a emplear, Lucene.

De igual manera se estudiaron los algoritmos no triviales presentes en la solución, realizándole a cada uno un análisis de su complejidad y se describieron funciones necesarias para solucionar el problema.

Capítulo 3 Validación de la solución propuesta.

La prueba del software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación. El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Es una actividad en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos específicos, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente.
Se especificarán técnicas de prueba de software que permitan comprobar la validez de la solución propuesta. Se detallan los casos de prueba con sus objetivos y alcance. Se abordará el tema relacionado con las herramientas empleadas en el proceso de desarrollo y se analizan los resultados obtenidos

3.0 Diseño de las pruebas unitarias y funcionales.

Las pruebas unitarias permiten probar cada unidad independiente del software. Actúan esencialmente sobre el código fuente y sobre los elementos básicos de la interfaz de cada módulo. Los casos de prueba que se generan durante las pruebas de unidad deben estar encaminados a verificar los siguientes elementos (Angelfire):

- Interfaz: Se prueba la interfaz de la herramienta para asegurar que la información fluye de forma adecuada hacia y desde la unidad de programa que está siendo probada.
- Estructuras de datos locales: Se examinan las estructuras de datos locales para asegurar que los datos que se mantienen temporalmente, conservan su integridad durante todos los pasos de ejecución del algoritmo.
- Condiciones límites: Se prueban las condiciones límites para asegurar que la herramienta funciona correctamente en los límites establecidos como restricciones de procesamiento.
- Caminos independientes: Se ejercitan todos los caminos independientes de la estructura de control para asegurar que todas las sentencias de la herramienta se ejecutan por lo menos una vez.
- Caminos de manejo de errores: Se prueban todos los caminos de manejo de errores‖ .

Con las pruebas unitarias es posible aislar una parte del código de manera que pueda ser analizado. Un ejemplo de esto es evaluar las funciones o métodos, a los cuales se les realiza una entrada de datos para

obtener los datos de salida correctos. Este tipo de pruebas validan la forma en la que las funciones y métodos trabajan en cada caso particular. Las pruebas funcionales no solo validan la transformación de una entrada en una salida, sino que validan una característica completa. De modo que las pruebas funcionales validan procesos y requieren de un escenario.

Estas pruebas simulan la navegación del usuario, realizan peticiones y comprueban los elementos de la respuesta tal y como lo haría manualmente un usuario para validar que una determinada acción hace lo que se supone que debe hacer. En las pruebas funcionales se ejecuta un escenario correspondiente a lo que se denomina un "caso de uso".

3.1 Tipos de prueba de software

La calidad de un sistema está determinada, entre otras cosas, por la coincidencia entre lo que se programó y los requisitos establecidos en la primera fase. Para comprobar el grado de cumplimiento de estos requisitos se usan las pruebas del sistema. La prueba de unidad es la prueba enfocada a los elementos probables más pequeños del software, consiste en una prueba estructural (o caja blanca), lo cual requiere conocer el diseño interno de la unidad puesto que verifica la lógica interna y el flujo de datos; y una prueba de especificación (de caja negra), basada sólo en la especificación del comportamiento externamente visible de la unidad (Andrews, 2003).

Estas definen un conjunto amplio de acciones de comprobación que abarcan todas las características que determinan la calidad de un software. Se comprueban las funcionalidades diseñando casos de prueba que definen cómo proceder. Estos casos de prueba incluyen los juegos de datos a usar que son los válidos o esperados y los no válidos o no esperados por el programa. Las pruebas se deben aplicar durante todo el ciclo de vida del software e invariablemente se le debe dedicar una gran parte del esfuerzo total del desarrollo. Se deben planificar correctamente desde el inicio y establecer qué hacer, cómo hacer, quién va a hacer y en qué condiciones hacer las comprobaciones.

El objetivo de los casos de prueba es forzar al máximo el sistema en los puntos críticos para encontrar fallos y detectar defectos.

Se debe escoger los tipos de prueba que se adapten mejor al sistema que se va a probar. Para esto se debe tener en cuenta el lenguaje de programación, el proceso de desarrollo, las características de los desarrolladores, el tipo de funcionalidad que se implementa, la plataforma en que se ejecutan los

procesos, los errores más importantes, si la aplicación es de escritorio o web, si realiza conexiones a bases de datos entre otras observaciones.

3.1.1 Colecciones de pruebas para SRI

A continuación se mostrarán breves descripciones de pruebas para evaluar la calidad y rendimiento de los Sistemas de Recuperación de Información (Santos, 2008-2009).

Colecciones de pruebas basadas en:

- Conjuntos de documentos.
- Conjuntos de consultas.
- Juicios realizados por expertos sobre la relevancia o pertinencia de cada documento a cada consulta.

Evaluación:

Niveles en los que puede ser evaluado el proceso de RI:

- Procesamiento: Eficiencia en cuanto a tiempo de procesamiento y uso de la memoria.
- Búsqueda: Efectividad de los resultados.
- Sistema: Satisfacción del cliente.

Importante: En respuesta a una consulta un Sistema de Recuperación de Información obtiene de una colección de documentos un conjunto o lista ordenada de los mismos.

Una medida de calidad del funcionamiento de los SRI se tiene a partir del análisis de la lista ordenada, lo que conllevaría a una mayor satisfacción de las necesidades del cliente.

Para la evaluación se tiene en cuenta los siguientes términos:

	RELEVANTES	IRRELEVANTES
Recuperados	RR	RI
No Recuperados	NR	NI

RR
RI = REC – RR
NR = (REL - RR)
NI = NN U (REC - RR)

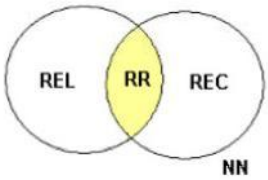


Imagen 6: Evaluación de la Precisión.

Precisión: Fracción de los documentos recuperados que son relevantes.

$$Precisión = \frac{|RR|}{|RR \cup RI|}$$

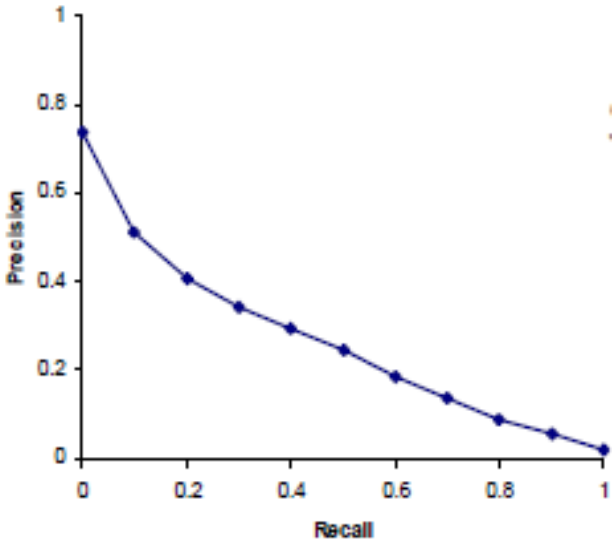
Recobrado: Fracción de los documentos relevantes que fueron recuperados.

$$Recobrado = \frac{|RR|}{|RR \cup NR|}$$

Análisis del Recobrado vs Precisión:

- La **Precisión** garantiza que todos los documentos que se obtienen son relevantes, pero podrían faltar documentos relevantes.
- El **Recobrado** garantiza que todos los documentos que se obtienen son relevantes, pero podría incluir otros que no lo son.
  - La medida de recobrado es una función no decreciente para el número de documentos no recuperados.
  - Se puede conseguir un recobrado alto (pero la precisión baja), recuperando todos los documentos para todas las consultas.

En la mayoría de los SRI a medida que el Recobrado o la cantidad de documentos recuperados aumentan la Precisión disminuye.



Gráfica 1: Precisión vs Recobrado

Cabe mencionar que también existen otras medidas de evaluación.

Ejemplos de ellas son:

- R-precisión
- Medida F
- Medida E
- Proporción de Fallo

R-Precisión:

Precisión para la posición R del ranking de documentos relevantes a una consulta dada para la cual existen R documentos relevantes.
Ejemplo:

n	doc #	relevant
1	588	x
2	589	x
3	576	
4	590	x
5	986	
6	592	x
7	984	
8	988	
9	578	
10	985	
11	103	
12	591	

Total de documentos relevantes: 6

R-Precisión = 4/6 = 0.67

Gráfica 2 R-Precisión

Esta medida es un buen parámetro para medir el comportamiento del modelo utilizado para cada consulta individual en un experimento. Sin embargo, usar un único número para evaluar todo el comportamiento de un algoritmo de recuperación a partir de varias consultas puede resultar impreciso, por ello se utilizan otras medidas alternativas.

Medida F:

Es una medida que armoniza Precisión y Recobrado teniendo en cuenta a ambos.
Por ejemplo:

$$F = \frac{2PR}{P+R} = \frac{2}{\frac{1}{R}+\frac{1}{P}}$$

- La medida F toma un alto valor, la Precisión y el Recobrado son altos, por la que la determinación del valor máximo para F puede interpretarse como un esfuerzo por encontrar el mejor compromiso entre Precisión y Recobrado.

Medida E:

Es una variante de la medida F que permite enfatizar la Precisión sobre el Recobrado, y está dada por la siguiente ecuación.

$$E = \frac{(1+\beta^2)PR}{\beta^2 P+R} = \frac{(1+\beta^2)}{\frac{\beta^2}{R}+\frac{1}{P}}$$

- ß= 1: Igual peso o énfasis para la Precisión y el Recobrado (E=F).

- ß>1: Mayor peso a la Precisión.
- ß<1: Mayor peso para el Recobrado.

Proposición de Fallo:

Problemas con la Precisión y el Recobrado.

- Números de documentos irrelevantes que no son tenidos en cuenta.
- El Recobrado es indefinido cuando no hay documentos relevantes en la colección.
- La Precisión es indefinida cuando no se recupera ningún documento.

$$Fallout = \frac{no.of\ \ nonrelevant\ items\ retrieved}{total\ no.of\ \ nonrelevant\ items\ in\ the\ collection}$$

Fallout (fallo): es directamente proporcional a la división entre el *no.of nonrelevant ítems retrieved* (número de términos no relevantes recobrados) y el *total no.of nonrelevant items in the collection* (total de términos no relevantes pertenecientes a la colección).

Ahora bien existen otras medidas de calidad con respecto a los SRI y de acuerdo con su evaluación, entre las que cabe mencionar como por ejemplo **Medidas Subjetivas** que se definen en dos principales aspectos.

- Proporción de Novedad: La proporción de artículos recuperados y juzgados como relevantes por el usuario y que no habían sido previamente considerados relevantes.
- Proporción de Cobertura: La proporción de artículos recuperados relevantes además de los documentos relevantes conocidos por el usuario en búsquedas previas.

3.1.2 Especificación de Pruebas del sistema

Como en este caso se utilizó un API como Lucene (Clucene para C++), que está altamente validado y comprobado su rendimiento de calidad, teniendo en cuenta los estándares internacionales por los que se rigen estos tipos de SRI y teniendo en cuenta las pruebas antes mencionadas. Mencionar también que ha sido comparado con más de un software de este tipo, como los que se muestran en el capitulo anterior en el apéndice dedicado a Lucene, y su rendimiento es altamente confiable y ventajoso con respecto a los demás sistemas dedicados para estos tipos de procesos de RI, concluyendo entonces que es un API altamente confiable y de alto rendimiento.

3.1.2.1 R-Precisión

A continuación se muestran pruebas de calidad en cuanto a precisión del sistema:

Aplicación de la **R-Precisión** sobre el sistema:

Para esta prueba de precisión se tomo como muestra una colección de 20 a 30 documentos, teniendo en cuenta que estuvieran relacionados sobre algún tema y otros que estuvieran pero sin ninguna relación.

R-Precisión para Asistente RI:

N	doc#	relev
1	20	
2	22	x
3	25	x
4	27	
5	29	x
6	30	x
8	30	

R-Precisión = 4/6 = 0.7 aproximado

Para esta primera prueba el sistema arrojó resultados satisfactorios por encima de la media (0.5 < 0.7) y similares a los planteados anteriormente, solo variando en la cantidad de documentos devueltos y relevante para cada consulta como se puede ver en la tabla anterior.

Aplicación de la prueba de ejecución de **indexado contra tiempo**:

Para una cantidad similar entre 20 y 30 documentos de tamaño similar al de una paginan de un libro:
El sistema arrojo un tiempo aproximado expresado en milisegundos de 280 para una cantidad a indexar de 20, y de sólo 290 milisegundos aproximados para una cantidad de 30 documentos, entre los que se encontraba los 20 primeros.
Concluyendo que es un buen tiempo para el indexado de documentos, además de estar acorde este tiempo con otros sistemas que realizan el mismo proceso de indexación.

3.1.2.2 Aplicación de Prueba de Caja Negra sobre el sistema.

El objetivo fundamental del diseño de casos de prueba es conseguir un conjunto de pruebas que tengan la mayor probabilidad de encontrar los defectos del software.
Un caso de prueba es un conjunto de entradas de pruebas, condiciones de ejecución y resultados esperados desarrollados para verificar una función esperada.

Diseño de Casos de prueba correspondiente al Caso de Uso: Indexar Documentos.

Descripción de la funcionalidad:

El usuario entra la dirección de los documentos para ser indexados y específica la dirección en la que se van a guardar los índices de documentos, en caso de que las direcciones no existan o estén mal el sistema deberá mostrar un error.

Condiciones de Ejecución:

1. Se deberá ejecutar la aplicación principal.
2. Se debe seleccionar la opción correspondiente al indexado de documentos.
3. Se debe seleccionar la opción indexar documentos.

Secciones a probar en el Caso de Uso:

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: Indexar Documentos.	EC 1.1: Indexar Documentos.	El Caso de Uso comienza cuando el usuario accede a la interfaz Indexar Documentos y oprime la opción Indexar.	1. El sistema muestra la interfaz que permite Indexar los Documentos, mostrando los campos que se deben llenar: • Dirección de los Documentos • Dirección del índice. 2. El Usuario introduce los datos de los Documentos y presiona la opción indexar. 3. El sistema verifica los datos introducidos por el Usuario. 4. Si los datos introducidos son correctos, el sistema Indexa los Documentos y termina el CU.

	EC 1.2: Datos Incorrectos.	Consiste en detectar que faltan datos por llenar o verificar error de dirección.	1. El sistema detecta que faltan datos por llenar, o que hay un error de dirección y muestra un mensaje de error.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Obs.
El usuario entra datos válidos para Indexar los Documentos.		El sistema muestra un mensaje de éxito de la operación.	El sistema muestra el mensaje: "La acción a ocurrido satisfactoriamente".	
	El usuario no especifica los datos para indexar los documentos.	El sistema muestra un mensaje de error.	El sistema muestra el mensaje: "Mensaje de error: campos no validos o faltan campos por llenar".	

Diseño de Casos de prueba correspondiente al Caso de Uso: Buscar Documentos.

Descripción de la funcionalidad:

El usuario deberá seleccionar la opción Buscar donde se deberá entrar la consulta a realizar en el campo correspondiente, en caso de mal redacción de la consulta o existencia de posibles fallas el sistema deberá mostrar un error.

Condiciones de Ejecución:

1. Se deberá ejecutar la aplicación principal.

2. Se debe seleccionar la opción correspondiente a la Búsqueda.
3. Se debe especificar la consulta.
4. Se debe seleccionar la opción Buscar.

Secciones a probar en el Caso de Uso:

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC 1: Buscar Documentos.	EC 1.1: Búsqueda de Documentos.	El Caso de Uso comienza cuando el usuario accede a la interfaz Buscar Documentos y oprime la opción Buscar.	5. El sistema muestra la interfaz que permite Buscar los Documentos, mostrando los campos que se deben llenar: • Consulta a realizar. 6. El Usuario introduce la consulta y presiona la opción Buscar. 7. El sistema verifica la consulta del Usuario. 8. Si la consulta es correcta, el sistema realiza la búsqueda y termina el CU.

	EC 1.2: Datos Incorrectos.	Consiste en detectar que la consulta sea coherente con el proceso de búsqueda.	2. El sistema detecta que no es correcta y muestra un mensaje de error.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Obs.
El usuario entra la consulta para Buscar los Documentos.		El sistema muestra un mensaje de éxito de la operación, además de mostrar los documentos.	El sistema muestra el mensaje: “La acción a ocurrido satisfactoriamente”.	
	El usuario no especifica los datos para la Búsqueda.	El sistema muestra un mensaje de error.	El sistema muestra el mensaje: “Mensaje de error: consulta no valida o ha surgido un fallo en la búsqueda”.	

Después de realizarle un conjunto de pruebas al sistema se concluye que las pruebas constituyen un paso contundente en el ciclo de desarrollo del software que permiten verificar y revelar la calidad de un producto. Para determinar el nivel de calidad se ha efectuado un conjunto de pruebas para comprobar el grado de cumplimiento respecto de las especificaciones iniciales del sistema, como las de R-Precisión y pruebas de Caja Negra.

Conclusiones

Al concluir el presente trabajo se arriba a las siguientes conclusiones:

Se hizo un estudio de los antecedentes de los SRI en el mundo, lo que ayudo realizar la aplicación propuesta.

Se estudiaron los contenidos teóricos referentes a las características de las posibles herramientas a usar, lo cual permitió realizar una selección que se ajustara a las características del sistema a desarrollar.

Se realizó una revisión bibliográfica de las tecnologías actuales para el desarrollo de SRI.

Se concluyó utilizar para la implementación del sistema, como lenguaje de programación C++ en su versión para VC++, con el framework .Net , sobre el entorno de desarrollo de Visual Studio 2008 para VC++ y utilizando Lucene (Clucene para C++) para los procesos de búsqueda e indexación.

Se comprobó la calidad del sistema mediante pruebas de R-Precisión destinadas para este tipo de herramienta y pruebas de Caja Negra.

Mediante el análisis de los resultados se llega a la conclusión de que se cumplió con el objetivo trazado en la investigación del presente trabajo de diploma.

Recomendaciones

Con vistas a dar continuidad al desarrollo de este tipo de herramienta se recomienda:

- Ponerlo en práctica en los proyectos de la facultad.
- Presentar este trabajo en futuros eventos de interés.
- Crear un grupo de desarrollo para darle continuidad al proyecto de manera que pueda utilizarse en otros entornos.

Bibliografía

[Online] Advanced Text Indexing with Lucene, http://www.onjava.com/pub/a/onjava/2003/03/05/lucene.html, 15/Julio/2007..

definición.org. [Online] [Cited: 7 5, 2010.] http://www.definicion.org/diccionario/186.

Andrews, A., R. France. 2003. *Test adequacy criteria for UML design models.Software Testing, Verification and Reliability.* 2003. 95-127.

Angelfire. Angelfire. *www.angelfire.com.* [Online] [Cited: 05 2, 2010.] http://www.angelfire.com/empire2/ivansanes/bywbox.htm..

Apmarin. Apmarin. *apmarin.com.* [Online] [Cited: 7 3, 2010.] http://www.apmarin.com/download/481_dpt2.pdf.

Benavides, Prof. M.Sc. KrysciaDaviana Ramírez. *Modelos de RI.*

Calinsoft. Calinsoft. *calinsoft.com.* [Online] [Cited: 7 5, 2010.] http://www.calinsoft.com/2008/08/aplicaciones-web-ventajas-y-desventajas.html.

Def. Definicion. *Definicion.org.* [Online] [Cited: 7 5, 2010.] http://www.definicion.org/diccionario/186.

Frakes, W. B. and Baeza-Yates R. 1992. *Information Retrieval. Data Structures and.* s.l. : Prentice Hall PTR, 1992.

Hatcher, Erik/ Gospodnetic, Otis. *Lucene In Action.* s.l. : Oreilly & Associates Inc. 1932394281.

http://www.apmarin.com/download/481_dpt2.pdf. *DESARROLLO DE PROYECTOS.*

JOYANES AGUILAR, LUIS. 2003. *FUNDAMENTOS DE PROGRAMACION, ALGORITMOS, ESTRUCTURAS DE DATOS Y O BJETOS.* 3. Madrid : S.A. MCGRAW-HILL, 2003. p. 996. 9788448136642 .

Masternewmedia. Masternewmedia. *masternewmedia.org.* [Online] [Cited: 7 5, 2010.] http://www.masternewmedia.org/es/aplicaciones_web/temas_de_aplicaciones_web/Beneficios_De_Las_Aplicacio.

masternewmedia.org. [Online] [Cited: 7 5, 2010.] http://www.masternewmedia.org/es/aplicaciones_web/temas_de_aplicaciones_web/Beneficios_De_Las_Aplicacio.

Peña, Catalina Naumis. 2002. *MODELO DE CONSTRUCCIÓN DE TESAUROS.* Madrid : s.n., 2002. 84-669-2214-8.

Reisdorph, Kent. *Borland C++ Builder 3.* 3. s.l. : Prentice Hall. p. 856. Un buen libro para empezar con Builder. Aunque trata la versión 3, todo es perfectamente válido para versiones posteriores. Aunque es un libro de iniciación, con un nivel más básico que los de Franciso Charte, es muy aconsejable saber C++ de antemano. . 970-17-0203-4.

Santos, Prof. MSc. Rafael Oliva. 2008-2009. *Introducción a los Sistemas de recuperación de Información.* Habana, Departamento de Ciencia de la Computación (UH) : s.n., 2008-2009. Departamento de Ciencia de la Computación UH.

Sons, John Wiley &. 2008. *Professional Visual Studio 2008.* 2008. p. 1032.

SourceForge. SourceForge. *sourceForge.net.* [Online] [Cited: junio 3, 2010.] http://sourceforge.net.

Tucker, Allen B. *Lenguajes de programación.* s.l. : Editorial McGraw Hill.

Usaola, Dr. Macario Polo. *Mantenimiento Avanzado de Sistemas de Información, Pruebas del Software.* Ciudad Real : s.n.

Witten, I. H. Moffat A. and Bell T. C. 1999. *Compressing and Indexing Documents and Images.* Second Edition ed. s.l. : Morgan Kaufmann, 1999.

Wordpress. Wordpress. *wordpress.com.* [Online] [Cited: 7 5, 2010.] http://conceptoscomputacionales.wordpress.com/2009/06/11/diccionario-informatico-a/.

Zárate Rea, Héctor. Noviembre de 2008. *Paradigmas de Programación.* Mexico : s.n., Noviembre de 2008.