UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

“PROPUESTA DE OPTIMIZACIÓN DE LA ARQUITECTURA DE CORPUS ELECTRÓNICOS DEL GRUPO DE INGENIERÍA LINGÜÍSTICA”

TESIS

QUE PARA OBTENER EL TÍTULO DE

INGENIERO “**EN COMPUTACIÓN**”

PRESENTA:

“**ALEJANDRO ROSALES MARTÍNEZ**”

ASESOR:

DR. CARLOS FRANCISCO MÉNDEZ CRUZ

CIUDAD UNIVERSITARIA 09 /junio/ 2014.

Contenido

Índice de figuras

Índice de tablas

Introducción

Dentro de la carrera de Ingeniería en Computación se encuentra el módulo de Sistemas Inteligentes al cual pertenezco. Como sub-módulo de éste se encuentra el área de tecnologías del lenguaje, misma que el Grupo de Ingeniería Lingüística (GIL) explota como puerta de entrada hacia la Facultad de Ingeniería. Y es así como se puede conocer desde la Facultad de Ingeniería al GIL.

El GIL es un grupo dedicado a la investigación en ingeniería lingüística. Un área bastante amplia y relativamente desconocida, es por esto último que las materias del sub-módulo no sean muy concurridas; sin embargo, esto en nada desalienta la labor del GIL. Sus 15 años de existencia, en los que se han obtenido un gran número de desarrollos y proyectos, le han dado gran prestigio y reconocimiento dentro de su área. Este grupo multidisciplinario incluye a estudiantes de lingüística, computación, inclusive de odontología y es que el área es tan amplia como el lenguaje mismo.

Dentro de los desarrollos del GIL se pueden encontrar:

- Un generador de palabras clave.
- Un generador de agrupamientos semánticos.
- Un banco terminológico.
- El DESCRIBE (un sistema de extracción y clasificación de términos y definiciones).
- El Corpus Histórico del Español en México.
- El Corpus Básico Científico del Español de México.
- El Corpus del Español Mexicano Contemporáneo.

Los anteriores son sólo algunos de los desarrollos que han surgido a partir de diversos proyectos que el GIL ha encabezado. Estos y más desarrollos, así como la explicación más a fondo de cada uno se pueden encontrar en el sitio web del grupo (http://www.iling.unam.mx).

Como ya he mencionado, pertenezco al módulo de Sistemas Inteligentes y fue así como llegué al GIL. En mi tiempo dentro del grupo he colaborado en el Corpus Histórico del Español en México (CHEM). Mi colaboración al corpus consistió en desarrollos para el indexador del mismo, tales como: aportar las bases para un indexado multi-hilos, un indexado incremental y la adaptación para una interfaz que ejecutara el indexador vía terminal.

Gracias a esta colaboración, fui asignado al proyecto del Corpus del Español Mexicano Contemporáneo (CEMC). Este proyecto, ya desarrollado y puesto en línea, presentó distintos nichos de oportunidad una vez que se comenzó a trabajar con él. Estos me dieron la oportunidad de obtener una beca por parte de El Colegio de México, A.C. para el desarrollo de soluciones y mejoras al corpus.

Y es a partir de este punto que surge el interés por el desarrollo de una propuesta de optimización para este corpus y otros del GIL. Luego, a partir de esta idea surge el interés por esta tesis y se deriva su objetivo central, el cual definiré más adelante. Y es que la comparación entre los corpus del GIL y otros corpus lingüísticos, como el Corpus del Español o el Corpus de Referencia del Español Actual, muestra un balance que no resulta positivo para el grupo. De hecho, con esta comparación se puede apreciar, de alguna manera, la relevancia y el impacto de los desarrollos del GIL.

Ya desde el siglo XIX eran empleadas las colecciones de textos (corpus textuales) para distintos análisis (Procházková, 2006); sin embargo, no resultaba nada práctico obtener

datos cuantitativos de estas colecciones, dado que una búsqueda o investigación manual resultaba bastante lenta y muy poco eficaz. Un método automatizado y más confiable era necesario para que se le pudiera dar un mayor uso a los corpus textuales. Con la llegada de la computación, la automatización se vio como una opción; mas primero había que tener los textos en un formato digital y fue que en los años 60 aparecieron los primeros corpus electrónicos. Éstos estaban en forma de tarjetas perforadas, marcando el inicio de esta nueva herramienta lingüística.

Como lo había mencionado, actualmente el GIL cuenta con distintos corpus lingüísticos electrónicos en línea abiertos a consulta. Un corpus lingüístico electrónico es "un conjunto de textos elegidos y anotados con ciertas normas y criterios para el análisis lingüístico, de forma que se sirve de la tecnología y de las herramientas computacionales para generar resultados más exactos." (Sierra, 2008, p. 455).

En la actualidad la cantidad de información digital disponible es abrumadora y la capacidad de almacenamiento ha dejado de ser un problema. Es por esto que ahora el problema radica en manejar toda esta información, pues al tener acceso a tal cantidad de documentos, el revisarlos o saber su contenido se vuelve una tarea de mucho tiempo, incluso utilizando herramientas informáticas. Con el paso del tiempo, la tecnología actual supera en mucho a la de hace diez años, logrando así disminuir el tiempo de procesamiento y de respuesta de los sistemas, para darle así un nuevo significado a la frase "mucho tiempo", siendo éste cuestión de minutos.

No en todos los corpus de hoy en día hay que esperar minutos para recibir una respuesta, existen distintos ejemplos de corpus en los que el tiempo de respuesta está medido en segundos, tales como:

- Collins corpus. Contiene 2.5 mil millones de palabras en inglés, obtenidas a partir de sitios web, periódicos, revistas y libros publicados alrededor del mundo y material hablado de la radio, televisión y conversaciones del día a día. Este corpus es actualizado cada mes para ayudar a los editores del diccionario Collins a identificar la aparición de neologismos y sus significados a partir del momento de su primer uso (*About the Collins and the Bank of English*, 2012).
- Bank of English corpus. Forma parte del Collins Corpus. Contiene 650 millones de palabras elegidas de una cuidadosa selección de fuentes para dar una reflexión balanceada y precisa del inglés como es usado todos los días (About the Collins and the Bank of English, 2012).
- Corpus of Contemporary American English (COCA). Este corpus es el corpus gratuito más grande disponible en inglés, además de ser el único de gran tamaño y balanceado del inglés americano. Este corpus es una obra de Mark Davies y tiene más de 425 millones de palabras obtenidas en iguales cantidades de textos de ficción, revistas populares, periódicos y documentos académicos, además de fuentes habladas. Este corpus, es actualizado de una a dos veces al año (Davies, 2008).
- British National Corpus (BNC). Este corpus es una colección de las palabras escritas y habladas del inglés británico de la última parte del siglo XX, cuenta con un tamaño de 100 millones de palabras. Su última edición fue publicada en el año 2007 (*What is the BNC?*, 2009).

La lista anterior es una muestra de los corpus para inglés más conocidos y grandes. Existen también corpus en español, entre los que destacan por su tamaño los siguientes.

- Corpus de referencia del español actual (CREA). Este corpus está constituido con ejemplos de los últimos 25 años del español. El último dato registrado acerca del tamaño de este corpus es de 125 millones de formas. Este corpus es desarrollado y sustentado por la Real Academia Española (REAL ACADEMIA ESPAÑOLA, "Corpus de referencia del español actual", s.f.).
- Corpus del Español. Este corpus cuenta con 100 millones de palabras obtenidas de más de 20,000 textos del español, que pertenecen al periodo comprendido entre los siglos XIII al XX. Este corpus permite hacer búsquedas por lema, categoría gramatical y hasta por sinónimos, además de otras características que permiten un estudio más detallado de la lengua. (Davies, 2002).

Así como éstos, existen muchos otros corpus de distintos tamaños y lenguajes, como el Corpus do Português de 45 millones de palabras (Davies, 2006) o el Corpus del Español Mexicano Contemporáneo de casi 1, 900, 000 ocurrencias, notablemente más chico que los demás ejemplos. Sin embargo, el tamaño de un corpus no está ligado directamente con su grado de utilidad. Incluso dentro de la lingüística este es un tema que genera bastante controversia y es que desde el surgimiento de los corpus electrónicos, a principios de los 60 con el Brown Corpus, han existido dos líneas de estudio o grupos dentro de los lingüistas. Por un lado, aquellos que apoyan el uso de los corpus lingüísticos para el estudio de distintos fenómenos de la lengua y, por otro, aquellos que apoyan la idea de Chomsky, quien en algún momento se mostró en contra de tomar a los corpus como una fuente consistente para el estudio de la lengua (Rojo, 2002). Mas el objetivo de esta tesis no es el de tomar parte o apoyar alguna de estas posturas, es en cambio el de proponer una arquitectura que permita un trabajo ágil y eficiente con corpus de gran tamaño.

La arquitectura que esta tesis pretende proponer será puesta a prueba inicialmente con el CEMC. Sin embargo, de resultar exitosa esta prueba, podría dar servicio a los siguientes corpus del GIL:

- Corpus Histórico del Español en México (CHEM)
- Corpus de las Sexualidades en México (CSMX)
- Corpus Básico Científico del Español de México (COCIEM)
- Corpus Lingüístico en Ingeniería (CLI).

Esta tesis se divide en varios capítulos. Dentro de la introducción expongo los objetivos, los problemas, la metodología y el alcance de la tesis. En el capítulo uno, se definen los conceptos que he considerado básicos para poder comprender el resto del trabajo. En el capítulo dos, entraré en detalle sobre la arquitectura de los corpus electrónicos del GIL y haré un análisis a detalle sobre la arquitectura base de esta tesis, la arquitectura del CEMC.

En el capítulo tres presentaré mi propuesta, haciendo un análisis preliminar sobre las opciones disponibles y la justificación de mi selección. Así también presento en este capítulo un análisis del software utilizado, del diseño de la propuesta y del desarrollo de la misma. En el capítulo cuatro presento los resultados y la evaluación hecha a la nueva propuesta, preparando así al usuario para las conclusiones obtenidas al finalizar esta tesis. Mismas que muestro en el quinto capítulo de este trabajo. Por último se encuentran las referencias de todas las citas escritas en esta tesis.

Problema

Describir la arquitectura del CHEM resulta trascendente para este trabajo de tesis, puesto que es ésta la base de la arquitectura del CEMC. El CHEM es un corpus diacrónico[1] del español de México. Está formado por 183 documentos, etiquetados en XML (Extensible Markup Language), obtenidos de distintas fuentes, mismos que dan un total de 290, 245 palabras. El etiquetado permite un manejo más sencillo de la información, evitando así el procesamiento del lenguaje natural para la obtención y clasificación de las ocurrencias de palabras.

Además, permite estandarizar la información contenida dentro del corpus ya que al ser éste diacrónico, presenta problemas como "la falta de normas ortográficas que impiden identificar rasgos lingüísticos consistentes y el manejo de caracteres que hoy en día ya no se utilizan" (Medina y Méndez, 2006, p. 248). Siendo así, el etiquetado permite realizar búsquedas por características lingüísticas, dándole una mayor utilidad al corpus. Es por esto que cada texto en el corpus esta etiquetado en los siguientes niveles (Medina y Méndez, 2006):

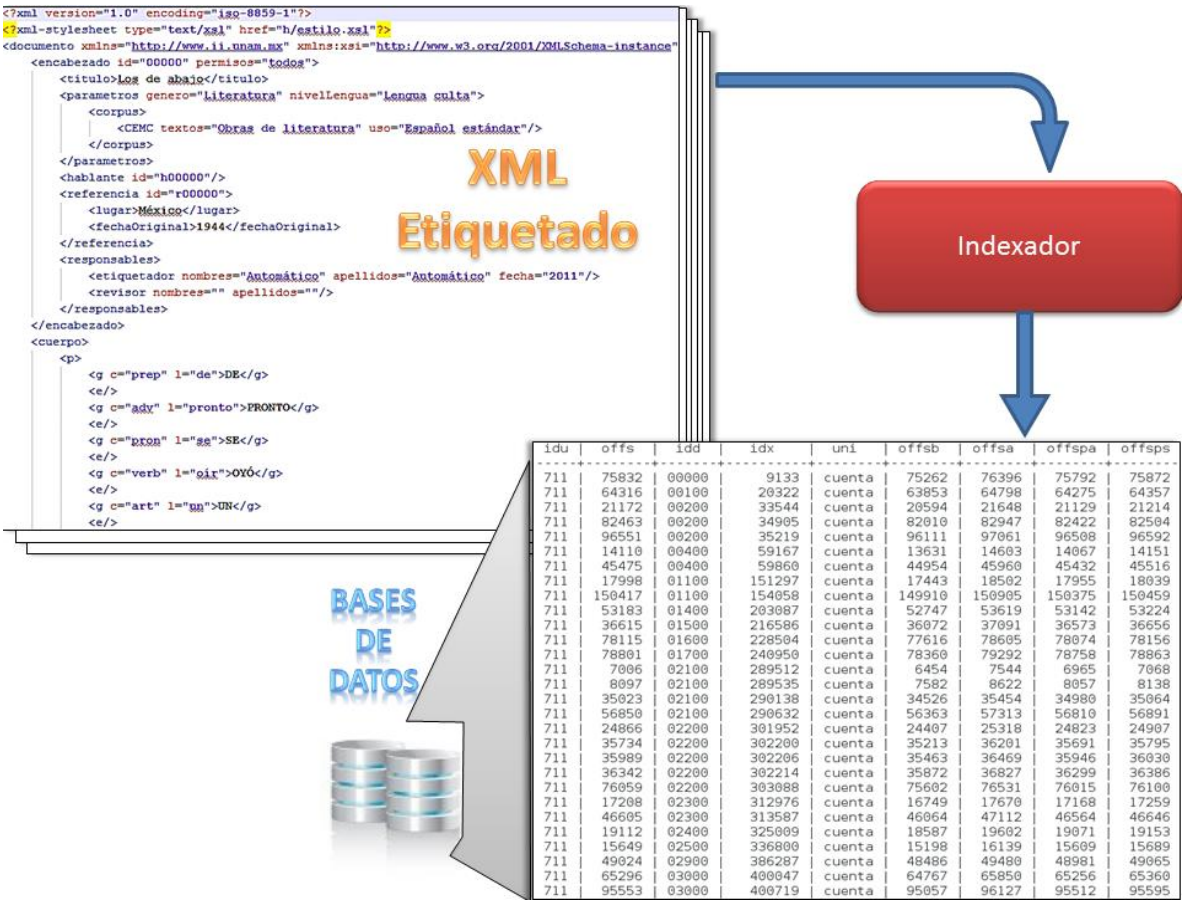
- Fonológico. Permitiendo realizar búsquedas sin importar la representación gráfica de la palabra deseada.
- De lema. El lematizar cada palabra de cada documento le da la capacidad al sistema de realizar búsquedas de distintas palabras con la misma forma canónica (por ejemplo, el lema vivir permitiría buscar las palabras vivimos, vivieron, vivirán).

[1] Diacrónico se refiere a que contiene textos de distintos periodos de tiempo.

- POS o etiquetado gramatical. Con este nivel, se alcanzan búsquedas por partes de la oración. Tal es el caso de poder buscar 'verbo + preposición + DECIR' y obtener frases como 'voy a decir' o 'enviaba a decir'.

Del mismo modo, el CEMC cuenta con los niveles de etiquetado de lema y POS, obteniendo así las mismas características de búsqueda.

En cuanto al método de indexado de palabras, el CHEM utiliza el enfoque de bases de datos relacionales inspirado en el trabajo de Mark Davies (s.f.). Sobre la arquitectura de este enfoque, se puede decir que todo comienza a partir de los textos etiquetados en XML que forman el corpus. De ellos se obtienen una base bibliográfica y una tabla de n-gramas. La base bibliográfica se construye a partir de los encabezados de los textos en XML. Éstos contienen los datos bibliográficos de cada documento, como se pueden ver en la Figura 1.



idu	offs	idd	idx	uni	offsb	offsa	offspa	offsps
711	75832	00000	9133	cuenta	75262	76396	75792	75872
711	64316	00100	20322	cuenta	63853	64798	64275	64357
711	21172	00200	33544	cuenta	20594	21648	21129	21214
711	82463	00200	34905	cuenta	82010	82947	82422	82504
711	96551	00200	35219	cuenta	96111	97061	96508	96592
711	14110	00400	59167	cuenta	13631	14603	14067	14151
711	45475	00400	59860	cuenta	44954	45960	45432	45516
711	17998	01100	151297	cuenta	17443	18502	17955	18039
711	150417	01100	154058	cuenta	149910	150905	150375	150459
711	53183	01400	203087	cuenta	52747	53619	53142	53224
711	36615	01500	216586	cuenta	36072	37091	36573	36656
711	78115	01600	228504	cuenta	77616	78605	78074	78156
711	78801	01700	240950	cuenta	78360	79292	78758	78863
711	7006	02100	289512	cuenta	6454	7544	6965	7068
711	8097	02100	289535	cuenta	7582	8622	8057	8138
711	35023	02100	290138	cuenta	34526	35454	34980	35064
711	56850	02100	290632	cuenta	56363	57313	56810	56891
711	24866	02200	301952	cuenta	24407	25318	24823	24907
711	35734	02200	302200	cuenta	35213	36201	35691	35795
711	35989	02200	302206	cuenta	35463	36469	35946	36030
711	36342	02200	302214	cuenta	35872	36827	36299	36386
711	76059	02200	303088	cuenta	75602	76531	76015	76100
711	17208	02300	312976	cuenta	16749	17670	17168	17259
711	46605	02300	313587	cuenta	46064	47112	46564	46646
711	19112	02400	325009	cuenta	18587	19602	19071	19153
711	15649	02500	336800	cuenta	15198	16139	15609	15689
711	49024	02900	386287	cuenta	48486	49480	48981	49065
711	65296	03000	400047	cuenta	64767	65850	65256	65360
711	95553	03000	400719	cuenta	95057	96127	95512	95595

Figura 1. Diagrama básico del proceso de indexado

En la Figura 1 se muestran: un fragmento de uno de los documentos contenidos en el corpus y la captura de pantalla de una consulta hecha a la base de datos del CEMC. En la parte superior del documento XML se pueden identificar las etiquetas de *título, género, nivel de la lengua*, entre otras. Mismas que darán forma a la creación de la base bibliográfica. Paralelamente a ésta base se construye la tabla de n-gramas, realizando primero la tokenización[2] de los textos para así conformar los unigramas[3].

[2] La tokenización consiste en separar el corpus en elementos mínimos llamados palabras gráficas.
[3] Un unigrama corresponde a una palabra gráfica.

El siguiente paso dentro de la construcción de la tabla de n-gramas es la indexación de archivos. En éste se calcula el número de bytes necesarios para acceder al n-grama en el documento, junto a este proceso se recupera la descripción lingüística del n-grama. Estos datos son guardados en la base de datos. El último paso dentro de la construcción de la tabla es, la generación de frecuencias, en donde son obtenidas las frecuencias de cada n-grama y almacenadas como una característica más del mismo. Una vez que se cuenta con la base bibliográfica y la tabla de n-gramas se establecen las relaciones entre n-gramas y el documento que los contiene. Este proceso se ve resumido en el siguiente diagrama (Figura 2).

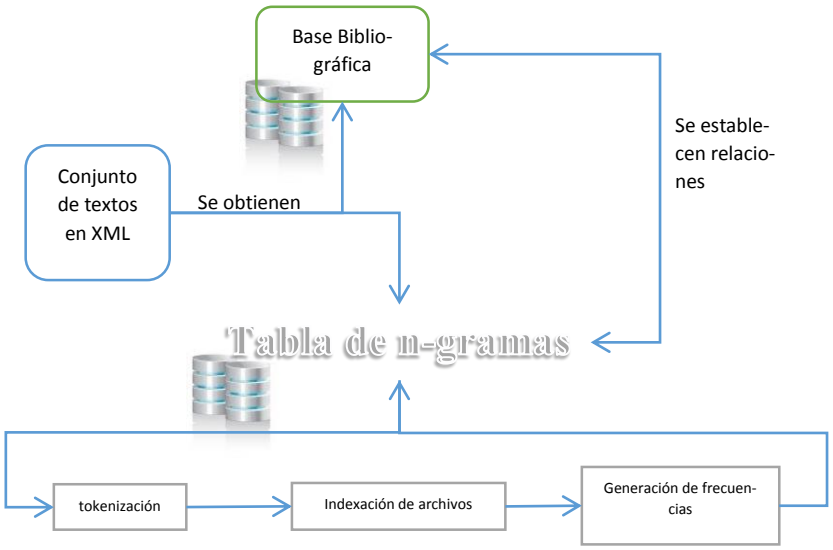


Figura 2**.** Proceso de indexado de corpus

La última parte de la arquitectura del CHEM, más en contacto con el usuario final, es el generador de concordancias. Este consta, como lo explican Medina y Méndez (2006), de una serie de clases Java que toman la petición del usuario, con las variantes de lema, fonológicas y POS, y regresan una ventana de caracteres hacia la izquierda y derecha de las palabras buscadas (concordancia).

El proceso del generador de concordancias comienza identificando la descripción lingüística de la petición del usuario y el n-grama deseados. Ya con estos datos, se realiza la consulta sobre la tabla de n-gramas. Cuando se obtienen las coincidencias, se obtienen también los documentos y las posiciones en el archivo de cada resultado. Con estos datos se accede al texto y se recupera la ventana de caracteres o concordancia, siendo esto y la información bibliográfica lo que se muestra en la pantalla de resultados al usuario.

Figura 3. Pantalla inicial de consulta del CHEM

En las figuras: Figura 3 y Figura 4, se muestra una consulta hecha al CHEM. En la Figura 3, se ve la página inicial sin consulta y tal cual como el usuario final la va a encontrar. Por otra parte, en la Figura 4, se ve el conjunto de concordancias como resultado de la consulta para la palabra *vida* en el rango de años de 1500 a 1600, con una ventana de 10 palabras

a la derecha y 10 a la izquierda. Dentro de los resultados se ven, en el extremo derecho, los datos bibliográficos del texto de donde fue obtenida cada concordancia.

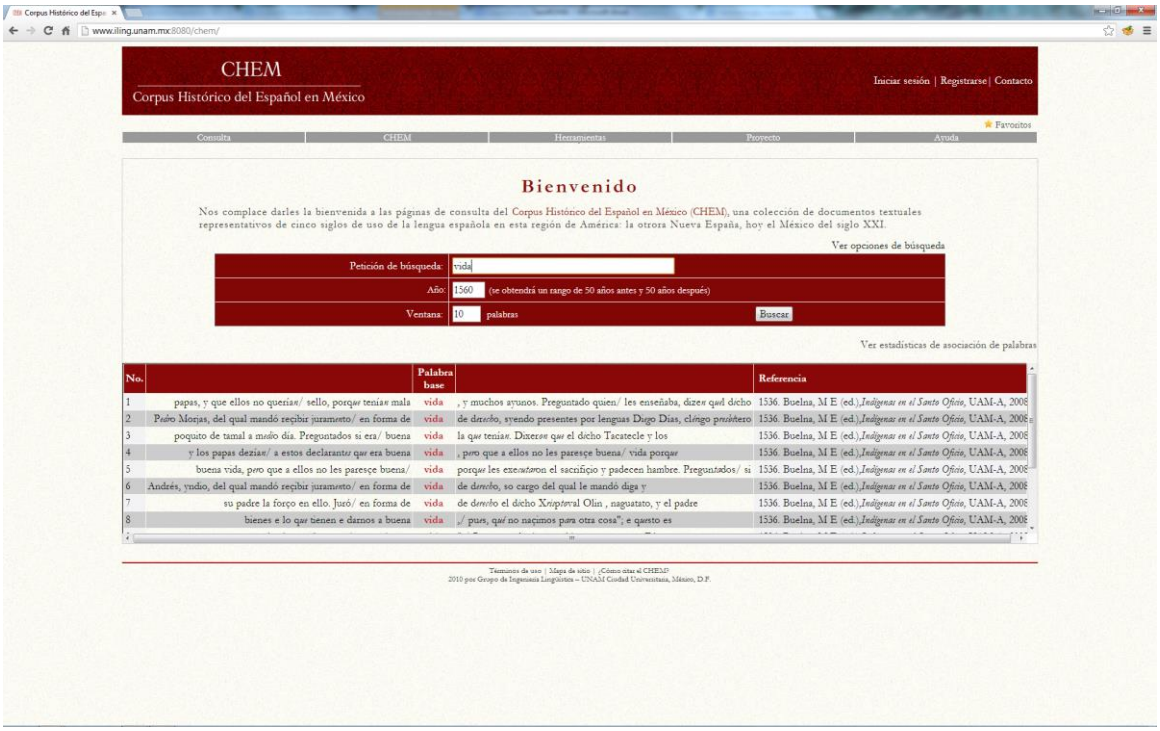


Figura 4. Ejemplo de consulta al CHEM

Aunque en mucho tiempo no se habían requerido grandes modificaciones al sistema de consulta, el Corpus del Español Mexicano Contemporáneo (CEMC) ha presentado un reto distinto al sistema. Reto que hasta hoy no tiene los resultados deseados y es que si bien existen consultas que se pueden realizar en segundos, existen otras para las que hay que esperar doce minutos y medio para poder visualizar los resultados. Tiempo que no resulta aceptable y que da pie a las preguntas: ¿Qué se está haciendo mal? Y ¿Cómo se puede mejorar? Es en la búsqueda de responder estas interrogantes que se origina este trabajo de investigación.

El solucionar este problema ha resultado de gran interés para el grupo, ya que éste es uno de los últimos corpus que el GIL ha puesto en línea[4].

Un segundo problema que llamó mi atención y que complementa la razón de ser de ésta tesis, es el indexado. Parte base del motor de búsqueda actual. Es aquí, en este proceso donde se han de generar todas las tablas que permiten acceder a la información contenida dentro del corpus. De modo que se han de procesar durante el indexado, en el caso del CEMC, alrededor de 1,000 documentos que van a dar casi 2 millones de palabras a la base de datos.

Esta cantidad de documentos, no debe representar reto alguno para un sistema de manejo de grandes cantidades de información. Es por esto que el tiempo de indexado actual, no deja por completo satisfecho al GIL. Ya que como se desea tener corpus de gran tamaño, tener un indexador que tarda 13.05 horas en 100 documentos, no ayuda en la idea de incrementar los corpus. Es por esto que se desea que este tiempo sea menor ya que a medida que crece el tamaño del corpus, este tiempo crece.

Con la idea de vislumbrar cuanto tardaría un corpus grande. Ya que en la arquitectura actual se están haciendo pruebas sobre un corpus artificial de 30 millones de palabras y el tiempo de indexado está siendo de 16.5679 horas por archivo, donde cada documento cuenta con un promedio de 30, 347 palabras.

En resumen los problemas de la arquitectura actual de los corpus del GIL son:

[4] Durante el desarrollo de esta tesis, el GIL puso en línea el COCIEM. Aunque éste es un corpus más grande que el CEMC, ya se habían realizado experimentos con éste, por lo que se decidió terminar este trabajo de investigación con el CEMC. De cualquier forma, en la medida de lo posible traté de involucrar al COCIEM en mi trabajo.

1. Concordancias. El generador de concordancias tiene tiempos de respuesta inaceptables. Tiempos de hasta 12 minutos, en consultas de grandes cantidades de resultados.
2. Indexado. El indexador puede llegar a tardar años de procesamiento continuo para lograr procesar un corpus de 30 millones de palabras. Así también puede tardar días de procesamiento continuo para indexar alguno de los corpus actuales del GIL. Tiempo que resulta poco óptimo, en la búsqueda de incrementar los mismos.

El impacto de estas modificaciones permitirá al GIL trabajar con corpus de gran tamaño.

Objetivos

Los retrasos en las consultas han cuestionado el cómo saber si se cuenta con la mejor arquitectura para soportar corpus de gran tamaño. Estas ideas y situaciones han dado origen al objetivo central de esta tesis, siendo éste: proponer una arquitectura distinta a la actual para los corpus lingüísticos electrónicos del GIL con el fin de solucionar los retos que enfrenta el sistema de consulta ante grandes volúmenes de datos.

Dentro de estos retos, se encuentran los siguientes puntos, que pueden verse como los objetivos específicos de esta tesis:

- Tener una arquitectura capaz de indexar y trabajar sobre corpus de gran tamaño.
- Tener un indexador incremental. Con esto se busca tener la capacidad de que una vez indexado un corpus, éste pueda crecer en tamaño y sea indexado sin la necesidad de volver a indexar los archivos previamente procesados. Esto es, indexando únicamente la parte nueva del corpus.

- Mejorar la velocidad de respuesta del generador de concordancias. Esto es, para que no se lleguen a dar tiempos de hasta doce minutos en una consulta.
- Conseguir un mejor uso del hardware disponible. Buscando tener una arquitectura que aproveche al máximo los recursos físicos disponibles, con la finalidad de obtener un mejor desempeño del sistema.

Metodología

La metodología que se pretende seguir para el logro de los objetivos planteados se plasma a continuación.

1. Analizar la arquitectura actual del CEMC. El primer paso para comenzar la investigación y antes de hacer cualquier modificación será analizar la arquitectura del motor de búsqueda para observar detenidamente el funcionamiento y las fallas que se presenten. Y es que para poder proponer una arquitectura que pueda competir con la actual, hay que conocer lo que se está haciendo bien y dónde se puede mejorar. Dentro de los puntos a considerar durante el análisis se encuentran:
    a) El proceso de indexado. Éste es una parte crucial dentro del motor de búsqueda, puesto que va a organizar los datos que se van a presentar al generador de concordancias para que éste realice las consultas. Así también es un punto a considerar puesto que la velocidad actual de indexado no es aceptable si se quiere trabajar con corpus de gran tamaño.
    b) El uso de las bases de datos relacionales. Revisar que otras opciones existen para administrar la información almacenada. Y tomar una decisión entre continuar o no con el uso de las bases de datos relacionales.

c) El generador de concordancias. Es importante analizar este módulo dentro de la arquitectura pues de aquí se obtienen los resultados tal cual se presentan al usuario. Es donde más accesos al disco duro se realizan. Lo que significa más tiempo de procesamiento. Además de que se procesa más estilo y detalle de la presentación, que en cualquier otra parte de la arquitectura, dándole una complejidad y tiempo de procesamiento altos para este módulo.

2. Investigación. Una vez comprendida la arquitectura y definidos los problemas de la misma, el siguiente paso será investigar: distintas opciones, conocer lo que utilizan otros corpus, lo que proponen distintos autores y las tendencias de la tecnología. Lo anterior para poder tomar una decisión y seleccionar la arquitectura que se va a desarrollar e implementar para trabajar con los corpus del GIL, según los recursos disponibles, los requerimientos, su estimado de vida y qué tan bien satisfaga todos estos puntos.
3. Diseño de la arquitectura del motor de búsqueda. El tercer paso de la metodología comprende desde el diseño del método de indexado hasta la presentación de los resultados, pues es preciso que cada parte funcione correctamente y se relacione de igual forma con las otras partes; para obtener así, un sistema útil y eficaz. Diseñar antes de desarrollar permitirá conocer qué se puede esperar de la arquitectura.
4. Desarrollar el motor de búsqueda. Desde el indexador hasta la presentación de los resultados, pasando por el generador de concordancias.
5. Probar la nueva arquitectura. Una vez terminado el motor de búsqueda, sólo queda evaluarlo para revisar el cumplimiento de los objetivos planteados.

Alcance

Según los objetivos anteriormente marcados, se podría pensar que como resultado final de ésta tesis se va a obtener un reemplazo completo del sistema actual. Es por esto que, me permito aclarar en el alcance de esta tesis que, no se pretende obtener todo un sistema de consulta que reemplace al actual.

Lo anterior se explica al considerar que la estrategia que se va a entregar al finalizar esta tesis, no abarca una interfaz web que permita una interacción sencilla entre el usuario final y el corpus. En cambio, sí se busca dejar una opción distinta lista para ser analizada por el GIL, dejando así la decisión y acción de implementar al grupo.

Dentro del documento hablaré del etiquetado, del formato que se utiliza (XML) y de las características de cada palabra tomadas en cuenta dentro del corpus. Mismas que se encuentran en etiquetadas dentro de cada documento. Pero aunque es algo que he de definir y usar, no es un punto que vaya a incluir dentro de la propuesta de optimización. Y es que a mi parecer, sugerir una modificación al estándar del etiquetado, merece una investigación propia para poder dar una justificación válida que permita siquiera considerar el etiquetado hecho a todos los textos del corpus.

La organización de los documentos del corpus, el tamaño, clasificación y características del corpus tampoco entran dentro del alcance de esta tesis. Pues cambios a estos valores sugieren un cambio el objetivo final del corpus mismo. Cosa que no es de interés dentro de la optimización del motor de búsqueda.

Los puntos que sí entran dentro del alcance son: el indexado de los documentos, el uso de bases de datos relacionales, la generación de concordancias y la presentación de los

resultados. Ésta última no desde un punto de vista estético, pero sí desde un punto de vista funcional.

I. Marco conceptual

En este capítulo se revisarán los aspectos relacionados con los conceptos necesarios para el desarrollo de la investigación. En primer lugar se expondrá lo relacionado al procesamiento del lenguaje natural, dentro de este punto ahondaré en algunos valores y características del procesamiento de textos que me van a servir para el desarrollo de todo el trabajo. Estos son la lematización, los tipos y ocurrencias y el etiquetado por partes de la oración.

En un segundo punto dentro del marco conceptual, se verá un poco de lo que es la ingeniería lingüística, técnicas y recursos. Además de su papel en México. Ya un poco más dedicado al tema, el punto tres tratará los corpus electrónicos. Definición, corpus existentes, su uso y las concordancias son tópicos a tratar dentro de la sección referente a corpus. Por último y ya con un conocimiento más amplio del tema, es que mostraré distintos métodos de indexado y arquitecturas de algunos corpus electrónicos.

I.1. Procesamiento del lenguaje natural

El lenguaje natural se puede entender como una forma de comunicación entre las personas, para expresar ideas, emociones y deseos a través de un sistema de signos que fueron creados en algún momento de la evolución del hombre (Lenguaje: ¿Qué es Lenguaje?, s.f.). Visto de esta manera, el uso del lenguaje parece dejar fuera a la tecnología. Sin embargo, también se puede tomar al lenguaje como la forma de comunicación entre dos agentes, un emisor y un receptor (Spyns, 1996).

En este segundo punto de vista, el emisor envía un conjunto de señales acústicas o gráficas a través de un medio de transmisión al receptor, mismo que comparte cierto conocimiento sobre las señales. Este conocimiento le permite interpretar el mensaje enviado. Por lo anterior, la tecnología puede entrar en la clasificación de medio de transmisión y en la de agente según la situación.

Específicamente, hoy en día las computadoras suelen ser un agente dentro de la comunicación usuario-máquina. Así, siendo el lenguaje una forma de comunicación, entre estos dos agentes se utiliza el lenguaje de programación. Esto permite a ambos agentes entenderse al compartir este mismo conocimiento, aunque esto resulte poco intuitivo y complicado para el usuario.

Estos lenguajes de programación permiten que el usuario le comunique al equipo sus intenciones y deseos. De éste modo el equipo tendrá conocimiento sobre alguna situación y podrá así reaccionar ejecutando las instrucciones correspondientes. Visto desde un punto de vista muy simplista, así es como funcionan los lenguajes de programación. Resulta fácil decir la función de un lenguaje de programación, pero no resulta igual de fácil establecer una comunicación hombre- máquina mediante un programa, dándole así la posibilidad de esta comunicación a un puñado de personas (los especialistas).

Mas éste no es el único uso que se le da a las computadoras. Como se mencionó previamente, también pueden fungir como medio de transmisión al permitir, gracias a esta gran red mundial conocida como Internet, establecer la comunicación entre personas sin importar la distancia entre las mismas. Así las computadoras dejan de ser un agente dentro de la comunicación, para ser ahora el medio para la misma. Un medio atemporal

incluso ya que permiten mediante textos o grabaciones, comunicar el conocimiento de tiempo atrás al momento en que se consulte, tal como lo haría un libro o un video guardados en el librero de la casa. Esto da pie a la inclusión de muchos más usuarios sin conocimientos profundos en los lenguajes de programación, pero sí con profundas ganas de mantenerse en contacto con el resto de las personas en este mundo.

Entonces, ¿cuál sería una manera más fácil de establecer esta comunicación? Una opción podría ser que todas las personas estuvieran capacitadas a un nivel medio en el uso de los equipos de cómputo y facilitar así el trabajo de los especialistas en la materia. Opción que no resulta sencilla en lo absoluto ni muy divertida para la mayor parte de la población. La otra opción es usar a éste medio, la tecnología, que no tiene voz ni voto, para que sea él quien trabaje para comprendernos y quien se acople a las necesidades de comunicación del usuario. De modo que la tecnología tendrá que procesar el lenguaje para simular que lo "comprende" el lenguaje natural del usuario dando paso al procesamiento del lenguaje natural.

I.1.1. Definición

El procesamiento del lenguaje natural (PLN) es un área de investigación dentro de la inteligencia artificial (IA). Según Liddy (2001) el PLN es un enfoque computacional para el manejo de textos que intenta ser similar a como lo hace un ser humano, esto como herramienta para realizar una variedad de tareas o aplicaciones.

Esta definición puede dejar a muchos inconformes; sin embargo, engloba en un sentido general la definición y objetivos del PLN. De cualquiera manera vale la pena ahondar un poco más en ciertos puntos de esta definición.

Desde un enfoque de cómputo, el PLN hace referencia al modelado matemático y computacional de varios aspectos del lenguaje, como lo menciona Joshi (1991). Busca modelar computacionalmente el lenguaje, ya sea oral o escrito. Esto genera un trabajo interdisciplinario en el cual se ven incluidas ciencias como la lingüística.

Con el uso del PLN, la lingüística ha de apoyarse más en la creación de reglas o ideas con la claridad necesaria para que puedan ser automatizadas o seguidas como instrucciones por máquinas. Así la lingüística será una ciencia más exacta, según lo expresa Gelbukh (2010).

En cuanto al manejo de textos se refiere, se podría entender como si la entrada de datos para un sistema de PLN fuera en todo momento datos escritos, dejando de lado la comunicación oral, lo cual no es cierto. Por esto conviene que se vea al texto, no como un escrito y sí como la materialización de la comunicación, incluyendo de este modo a la comunicación oral, escrita y gráfica. El campo de acción del PLN es tan amplio como lo es la comunicación y el lenguaje por lo que el PLN "incluye sistemas para el reconocimiento del habla, entendimiento del lenguaje y generación del lenguaje" (Joshi, 1991).

I.1.2. Tipos y ocurrencias

En específico, para esta tesis el caso es el manejo de corpus lingüísticos. Ya previamente se definió el concepto de corpus lingüístico electrónico, sin embargo, hay varios conceptos que aún faltan por definir para comprender más acerca de estos conjuntos de textos y su manejo. Manejo que sin duda se vuelve más sencillo al dividir los textos en unidades más pequeñas. Tales como, los tipos, las ocurrencias, los vocablos y la palabra gráfica misma.

Sin embargo, definir la palabra no es tarea sencilla. Pese a que todos manejamos un concepto propio de palabra, que nos permite comprender hasta cierto punto lo que es y su uso. Definir la palabra puede resultar en un problema lo suficientemente grande como para dedicarle una investigación específica, esto claro si queremos tomar en cuenta a la lingüística. Y es que lo que puede describir a una palabra para el español, puede no hacerlo para otro idioma.

Mas el objetivo en esta tesis no es ahondar en este problema. Pero es necesario definir una unidad básica para el estudio y análisis de estas grandes cantidades de datos. Para esto, Lara (2006) permite definir la palabra como la unidad de denominación que cumple con ciertas características demarcativas a nivel fonológico, semántico y morfológico. Sin embargo, para efectos del procesamiento computacional, una palabra gráfica o palabra generalmente será el conjunto de letras que se encuentran delimitados por espacios en blanco y/o signos de puntuación al principio y al final.

Esta definición de palabra gráfica, me permite tener una unidad mínima a partir de la cual comenzar el análisis de los resultados y el procesamiento del corpus. Y es que los corpus utilizados en este trabajo se traducen en miles de registros en una base de datos. Registros que se pueden ver como datos, cuyo resultado final exige ordenarlos y obtener estadísticas de ellos. Esto, con la finalidad de que el usuario final (quien hizo la consulta) sea capaz de obtener información a partir de los datos obtenidos.

Para lograr obtener dicha información, es necesario definir los conceptos de ocurrencia (*token*), tipo (*type*), lema y *stem*. Cada vez que una palabra aparece en un texto, será una ocurrencia (Lara, 2006). Visto de otra manera, la cantidad de ocurrencias en un

texto es igual a la cantidad de palabras en el mismo[5]. Esto implica que si hay palabras repetidas dentro del mismo texto, cada una representa una ocurrencia distinta. El número total de ocurrencias o *tokens* definirá el tamaño del corpus.

Por otra parte, un tipo será cada palabra distinta encontrada en el texto (Lara, 2006). Esto quiere decir que a diferencia de las ocurrencias, las repeticiones de cada palabra no generan un tipo distinto. De este modo, un texto puede tener 20 ocurrencias y 14 tipos, debido a las palabras repetidas dentro del mismo. Para el tratamiento e información que se va a obtener del corpus, se guarda la frecuencia absoluta de cada tipo. Ésta frecuencia absoluta es la cantidad de ocurrencias de la palabra dentro del corpus, la suma de las frecuencias absolutas de todos los tipos es igual al tamaño del corpus.

El lema puede entenderse como la forma normalizada de una palabra, algunas veces llamada forma de diccionario. De hecho, esta forma se vuelve un representante de un grupo de variantes morfológicas de una palabra. En el caso de los sustantivos se utiliza como lema el masculino singular, por ejemplo, para las palabras niño, niña, niños y niñas, el lema sería niño. Para los verbos suele emplearse el infinitivo, por ejemplo, para fui, iba y vamos, el lema asociado sería el infinitivo ir.

Finalmente, el *stem*, también llamado base de la palabra, es "una etiqueta […] para hacer referencia a la base sobre la cual las distintas formas flexivas [de una palabra] actúan" (Gómez, 2005). Para el caso de las palabras niño, niña, niños y niñas, el stem sería niñ-.

[5] Estoy consciente de que en un corpus también es posible encontrar elementos (tokens) distintos a las palabras, como las fechas, cifras, abreviaturas, etcétera.

Tanto el lema como el stem, como dice Gómez (2005), son una herramienta más dentro del procesamiento computacional, usada para reducir las grandes listas de índices que se pueden llegar a manejar con todos los tokens obtenidos de un corpus.

I.1.3. Lematización

Podemos entender la lematización como el procedimiento de asignar lemas a cada una de las palabras de un corpus.

Del interés por disminuir el gran tamaño que tenían las listas de palabras utilizadas dentro de los antiguos sistemas de recuperación de la información (RI), es que surge la lematización (Gómez, 2005). Hoy en día pese a que el almacenamiento no es un problema de consideración, la lematización es utilizada también para reducir las grandes listas de búsqueda, impactando así en los tiempos de respuesta de los sistemas. Al hacer una lista de lemas, se reduce la cantidad de datos llave y cada uno de ellos cuenta con sus distintas formas flexivas.

Hay que tener en cuenta que la lematización es una herramienta computacional que se basa en reglas estadísticas y métodos estadísticos cuyos fines son prácticos y no es exacta. Pues si se quiere analizar desde un punto de vista lingüístico, se verá que se cometen errores.

Si bien es cierto que en el CEMC el dato central sobre el que gira el indizado de la base de datos no es el lema, por lo cual se puede dudar del uso y la importancia de la lematización en el CEMC, sí se utiliza el lema para realizar búsquedas y como criterio dentro de otras consultas.

I.1.4. POST

El etiquetado, como ya lo he mencionado, no es más que una estrategia computacional para facilitar el procesamiento del texto. Una palabra etiquetada se ve acompañada, antes o después, por marcas descriptivas que la identifican y clasifican dentro de distintas categorías. En específico, el etiquetado por categorías gramaticales, en inglés *Part of Speech Tagging* (POST), busca asociar clases de palabra a cada *token* de un corpus.

En tiempos antes de Cristo, en la cultura griega ya se distinguía entre ocho clases de palabras (Mitkov, 2004). Estas clasificaciones eran: sustantivo, verbo, participio, artículo (incluyendo los pronombres relativos), pronombre, preposición, adverbio, conjunción, mismas que son tan solo un ejemplo del conjunto de clase posibles, ya que este conjunto puede cambiar de un corpus a otro. Por ejemplo, en el CEMC no se utilizó este conjunto. El corpus tiene su propia clasificación con trece campos distintos, como puede verse en la siguiente tabla de Méndez (2009) (Tabla 1).

Etiqueta	Categoría
0	Ambigua
1	Adverbio
2	Adjetivo
3	Conjunción
4	Preposición
5	Pronombre
6	Artículo
7	Contracción
8	Nominal
9	Verbo

A	Apoyos conversacionales
B	Nombres propios
C	Otros (cifras, errores y palabras que comenzaban con mayúscula)

Tabla 1. Conjunto de etiquetas del CEMC

Un ejemplo de etiquetado POS como se utilizaría en el CEMC se puede observar en la Tabla 2. Ésta ejemplifica el etiquetado de la frase:

demasiado grande y atrasada, a la escuela

DEMASIADO	Adverbio
GRANDE	Nominal
Y	Conjunción
ATRASADA	Nominal
A	Preposición
LA	Artículo
ESCUELA	Nominal

Tabla 2. Ejemplo de etiquetado por parte de la oración.

Todos los textos que comprenden el CEMC se encuentran etiquetados en el lenguaje XML. Este etiquetado permite un procesamiento más sencillo de cada uno de ellos, al conocer *a priori* características específicas de cada palabra y tener la información estructurada. Que el texto esté etiquetado es entonces una ventaja que se ve maximizada al complementarla con alguna API (*Application Programming Interface*) para el procesamiento de textos en XML. Esto aumenta y facilita aún más las posibilidades de procesamiento para los textos. Todo lo anterior combinado, el etiquetado XML, la lematización y el etiquetado POST, se traduce en un texto como el que se ve en la Figura 5.

```
<g c="adv" l="ya">YA</g>
<e/>
<g c="verb" l="haber">HABÍA</g>
<e/>
<g c="nom" l="paz">PAZ</g>
<e/>
<r c="Fpa">(</r>
<g c="pron" l="yo">YO</g>
<e/>
<g c="verb" l="nacer">NACÍ</g>
<e/>
<g c="prep" l="en">EN</g>
<e/>
<g c="otro" l="1918">1918</g>
<r c="Fpt">)</r>
<e/>
<g c="adv" l="cuando">CUANDO</g>
<e/>
<g c="verb" l="poder">PUDE</g>
<e/>
<g c="verb" l="ir">IR</g>
<r c="Fc">,</r>
<e/>
<g c="adv" l="demasiado">DEMASIADO</g>
<e/>
<g c="nom" l="grande">GRANDE</g>
<e/>
<g c="conj" l="y">Y</g>
<e/>
<g c="nom" l="atrasado">ATRASADA</g>
<r c="Fc">,</r>
<e/>
<g c="prep" l="a">A</g>
<e/>
<g c="art" l="la">LA</g>
<e/>
<g c="nom" l="escuela">ESCUELA</g>
<r c="Fp">.</r>
<e/>
<g c="verb" l="hacer">HICE</g>
<e/>
```

Figura 5. Ejemplo de un documento etiquetado del corpus

De la imagen se puede ver que hay palabras dentro de los pico-paréntesis <> (por ejemplo *adv* y *ya* en el primer renglón) y palabras rodeadas por éstos (por ejemplo, YA en el primer renglón). Las que se encuentran dentro son las etiquetas de las palabras rodeadas. Las etiquetas aquellas que hacen referencia a las partes de la oración son las que están frente a la letra *c*, y las que indican el lema son las de la letra *l*. En un explorador web, las únicas palabras que se ven son aquellas que están fuera de las etiquetas.

Tal es el caso de adv, verb, pron, prep, entre otras. Todas estas indican la parte de la oración de la palabra etiquetada, esto es, si es adverbio, verbo, pronombre y preposición respectivamente. Por otra parte, se puede notar que para las palabras HABÍA y NACÍ, se indican los lemas haber y nacer.

I.2. Ingeniería lingüística

Dejando un poco de lado los conceptos de corpus y manejo de textos, abordaré algo más general. Esto es, el concepto de Ingeniería Lingüística, mismo que ayudará a comprender aún más la relación entre la lingüística y la ingeniería. Y es que, si bien es sabido, en el argot ingenieril, que la ingeniería es un área interdisciplinaria, no resulta tan claro ver la relación entre la ingeniería en computación y la lingüística.

Dentro de las múltiples áreas de estudio y aplicación de la ingeniería en computación se encuentra la Inteligencia Artificial (IA). Área que, a grandes rasgos, busca generar sistemas informáticos con un comportamiento que en el ser humano se calificarían de inteligentes (Barceló, 2001, pp. 9). Dentro de ésta enorme área de investigación se encuentra el PLN.

Para ejemplificar de manera más clara la relación entre el PLN y la IA, tomé la definición de Sierra et al. (2004) de la IA: "que una computadora pueda comprender y producir un lenguaje natural, y por consiguiente ejecutar los procesos cognitivos requeridos para realizar esta tarea […] es algo que se liga a lo que en ciencias computacionales se ha denominado como Inteligencia Artificial".

I.2.1. Definición

Se puede definir a la ingeniería lingüística como una disciplina que aplica conocimientos sobre la lengua para el desarrollo de sistemas informáticos que permitan reconocer, comprender, interpretar y generar lenguaje en todas sus formas. Esto con el fin de emular al ser humano en su uso de la lengua (Moreno, 2009, p. 99).

A la ingeniería lingüística también se le llega a conocer como tecnologías del lenguaje, pues en la búsqueda de conseguir las distintas metas incluidas en la definición anterior, se ven envueltas diferentes aplicaciones. Algunas de éstas para permitir el tratamiento de

textos escritos, como la traducción automática o la corrección ortográfica. Otras que hacen posible el procesamiento del habla, requerido para el dictado automático o la síntesis de voz.

Su objetivo según Hilera, Bengochea, Sánchez, Gutiérrez y Martínez (2005) y con el cual coincido, es el de "proporcionar medios para ampliar y mejorar la utilización de la lengua". Hay que ver a la utilización de la lengua, como el uso que se le puede dar en su aplicación a la tecnología y no en el uso que le damos en las relaciones persona-persona.

I.2.2. Técnicas y recursos

Según el diccionario una técnica es "el conjunto de los procedimientos que se siguen para elaborar un objeto complejo o para manejar alguna cosa"[6]. Basado en éste concepto, el objeto a crear por la ingeniería lingüística resultan ser distintas aplicaciones. Así que a continuación mencionaré algunos de los procedimientos utilizados en esta área (técnicas) tomados del sitio *Ingeniería lingüística. Cómo aprovechar la fuerza del lenguaje*[7].

Del tratamiento de la voz humana se desprende la siguiente técnica: *la identificación y verificación del locutor*. Una aplicación de ésta técnica es la autenticación por voz de algunos sistemas ya que la voz de cada persona es tan única como sus huellas dactilares.

El *reconocimiento del habla* es otra técnica en la que el sistema recibe como entrada la voz para su análisis. Con este análisis se busca identificar las unidades constitutivas de las palabras. Una aplicación de esta técnica son los sistemas manejados por voz, sistemas que a partir de ciertas instrucciones habladas efectúan ciertas acciones correspondientes.

Estas dos técnicas aún tienen que mejorar y crecer ya que cuentan con muchas variables que no han sido descifradas en su totalidad. En el caso del reconocimiento del habla, por

[6] Diccionario del Español de México (DEM) http://dem.colmex.mx, El Colegio de México, A.C., [13/07/2013].
[7] http://sunsite.dcc.uchile.cl/~abassi/WWW/Lengua/ingenieria.html#tar.

ejemplo, el dominio debe ser restringido y no se pueden interpretar instrucciones a partir del habla natural de cualquier dominio. Además, hace falta que el hablante haga pausas en su discurso y emita enunciados muy puntuales sobre lo que desea.

El habla es una forma de comunicación, una manera de expresar el lenguaje. La escritura es otra manera, mas no todos los textos contienen sólo letras. Es por esto que el *reconocimiento de caracteres e imágenes* es otra más de las técnicas de la ingeniería lingüística. Dentro del reconocimiento de caracteres hay dos tipos o maneras de hacerlo:

- El Reconocimiento óptico de caracteres (ROC), hecho a partir de imágenes impresas.
- El reconocimiento inteligente de caracteres (RIC), que permite el reconocimiento de la escritura manual.

El análisis de documentos como imágenes viene muy ligado al reconocimiento de caracteres. Aunque para éste hay que tratar todo el documento previamente para obtener un resultado más preciso del análisis. Lo anterior se da porque es necesario identificar la estructura del texto, cómo se han compaginado gráficos y fotografías, líneas de separación, texto, etc.

Resulta casi natural pensar que dentro de las técnicas de la ingeniería lingüística se encuentre la *comprensión del lenguaje natural*. Y es que la comprensión del lenguaje resulta fundamental en muchas de las aplicaciones, sobre todo en la ingeniería lingüística.

Por su parte, los recursos lingüísticos son una de las principales formas de representar el conocimiento de la lengua. Dentro de los recursos de la ingeniería lingüística se encuentran

los siguientes: léxicos, léxicos especializados, gramáticas y corpus. Éstos últimos los detallaré más adelante en la tesis.

En un recurso léxico se encuentran conjuntos de palabras e información sobre ellas. Esta información puede ser su estructura gramatical, información sobre su estructura fonética o el significado de las mismas en diferentes contextos textuales.

Los recursos léxicos especializados se relacionan con los anteriores, pero suelen construirse de manera independiente como es el caso de los nombres propios, la terminología y las redes de palabras. Todos estos recursos van a permitir un procesamiento más sencillo y eficaz de los textos, dando mejores resultados para las aplicaciones. Por ejemplo, en el caso específico de los nombres propios, es importante contar con un diccionario de éstos para conocer el tratamiento para:

Los estados unidos votaron por aprobar la reforma

Estados Unidos votó por aprobar la reforma

De no contar con un diccionario de nombres propios, el sistema podría tomar como iguales a ambos enunciados, cuando en el primero se hace referencia a los estados de algún lugar que se unieron para aprobar una reforma, mientras que en el segundo se habla de un país que votó por aprobar cierta reforma.

La gramática, como parte de la lingüística, "estudia la estructura y las reglas de combinación de las palabras de una lengua"[8] así que un recurso gramatical ayudaría al sistema a

[8] Diccionario del Español de México (DEM) http://dem.colmex.mx, El Colegio de México, A.C., [21/09/2013].

conocer y/o manejar la formación de las palabras (morfología) y la formación de enunciados (sintaxis) según el significado de las palabras y su organización discursiva.

Inherente a la ingeniería lingüística es la multidisciplinariedad del área, esto se ve directamente reflejado en la naturaleza de sus técnicas. Sin embargo, no todas se han de aplicar al procesamiento de corpus textuales, siendo estos el área específica de interés en esta tesis. Algunos métodos que provienen de otras áreas para apoyar a las técnicas de la ingeniería lingüística son (Uszkoreit, s.f.):

- De software: en donde se incluyen los lenguajes de programación y los algoritmos para manejar los distintos tipos de datos.
- Algoritmos especializados: algoritmos específicos desarrollados para el análisis de textos, la traducción, el procesamiento sintáctico y morfológico, entre otras tareas.
- Estadísticos: muchas decisiones dentro de los sistemas son basadas en modelos estadísticos. Además de su utilidad, tienen un éxito especial en el procesamiento del habla y los sistemas de recuperación de información.
- Reglas o normas lingüísticas: un profundo procesamiento lingüístico requiere de emplear gramáticas y reglas lingüísticas. Esto con el fin de pasar al sistema conocimiento especializado.

Desarrollos más específicos que forman parte del procesamiento del lenguaje natural son los lematizadores, los analizadores morfológicos, los analizadores sintácticos y los modelos semánticos; mismos que resultan muy útiles para el procesamiento de corpus, entre otras aplicaciones. Estos ya fueron descritos arriba en la sección I.1 de procesamiento de lenguaje natural.

I.2.3. La ingeniería lingüística en México

El trabajo en ingeniería lingüística en México podría pensarse incipiente mas no es así. Ya desde la década de los setenta el doctor Luis Fernando Lara aplicó técnicas estadísticas para el análisis automático de un corpus, esto para crear el Diccionario básico del español de México (Santillán, 2012).

En cuanto a instituciones se refiere, también se ha desarrollado trabajo. El Instituto Politécnico Nacional (IPN), el Instituto de Astrofísica Óptica y Electrónica (INAOE), la Benemérita Universidad Autónoma de Puebla (BUAP), el Instituto de Investigación en Matemáticas Aplicadas y Sistemas (IIMAS) y el Instituto de Ingeniería de la Universidad Nacional Autónoma de México (UNAM), son algunas de las instituciones donde existe investigación en el área.

En algunos de estos institutos el trabajo se ha dado gracias a los esfuerzos de varios investigadores quienes han formado grupos y laboratorios de investigación, como es el caso del Laboratorio de Tecnologías del Lenguaje en el INAOE. En éste se ha trabajado en diccionarios electrónicos, recuperación de la información, estructuración automática de contenidos, interacción oral hombre-máquina, entre otros. Todos estos trabajos y más información se pueden encontrar en el sitio (http://ccc.inaoep.mx/grupos/lenguaje.php).

En el IPN el laboratorio a cargo de la investigación en esta área es el de Procesamiento de Lenguaje Natural y Procesamiento de Texto. Fue fundado en 1996, pertenece al centro de investigación en computación y tiene entre sus líderes a los investigadores Dr. Alexander Gelbukh y Dr. Grigori Sidorov. Desarrolla trabajos en las áreas de agregación de oraciones

preposicionales, análisis morfológico, gramáticas de árbol, etc. Se puede encontrar más información sobre este laboratorio y sus investigadores en el sitio del mismo (http://nlnp.cic.ipn.mx/lab-Spanish.htm).

En la UNAM existen dos grupos que realizan trabajos en ingeniería lingüística, uno de ellos es el Grupo Golem del doctor Luis A. Pineda, tiene su sede en el IIMAS. Aunque si bien es cierto que la ingeniería lingüística no es su línea principal de investigación, si es una línea importante dentro del grupo. Y es que el grupo centra su investigación en el modelado cognitivo de la interacción entre humanos y sistemas computacionales. De modo que para establecer esta comunicación es que hace falta el PLN y la ingeniería lingüística. El Grupo Golem tiene su sitio en la siguiente dirección http://golem.iimas.unam.mx/.

Por otra parte, en el Instituto de Ingeniería, también de la UNAM, reside el Grupo de Ingeniería Lingüística (GIL). El GIL es un grupo de 14 años de vida al haber nacido en 1999 por la iniciativa del doctor Gerardo Sierra Martínez. Éste es el fundador y director del grupo que ya ha dado varias generaciones de maestros, doctores y licenciados, quienes han colaborado en la investigación y desarrollo de distintos proyectos enfocados a la ingeniería lingüística.

El GIL durante su tiempo de vida no sólo se ha dedicado a la investigación y el desarrollo, también le ha dado su lugar a la difusión del campo. Prueba de ello es que además de participar en congresos, seminarios y talleres, ha sido también anfitrión en este tipo de eventos. Tal es el caso del 11th Biennial Conference on Forensic Lingüistics/Language and Law celebrado en junio del 2013 en la UNAM, o los distintos Coloquios de Lingüística Computacional en donde se dan a conocer los avances y estudios que al momento se realizan en los distintos laboratorios y grupos del área.

El GIL es un grupo dedicado a la investigación en ingeniería lingüística. Entre sus áreas de investigación se encuentran:

- Terminología y terminiótica.
- Bancos terminológicos.
- Lingüística de corpus.
- Lexicología y lexicografía.
- Lingüística forense.
- Resumen automático de textos.
- Análisis morfológico, sintáctico y semántico.

Es dentro de la lingüística de corpus que encuentra su lugar esta tesis como parte de la labor de investigación del GIL. Además, esta tesis es una continuación a un trabajo hecho en colaboración con El Colegio de México, A.C. (COLMEX). Trabajo que mostró frutos con la puesta en línea del CEMC, pero que ha tomado nueva fuerza por el interés de mejorar los tiempos de respuesta.

Que el GIL cuente con una arquitectura capaz de soportar corpus del tamaño del CREA, el CORDE o el Corpus del Español de Mark Davies es una meta que puede parecer ambicioso por parte de éste autor, pero que no es más que un aliciente y una presión para hacer mi mejor esfuerzo.

I.3. Corpus electrónicos

A lo largo de la historia se ha generado una gran cantidad de documentos, grabaciones e información en general. Cada filósofo, escritor, o persona incluso, que ha pasado por la historia del planeta ha contribuido de una u otra forma al crecimiento de este registro histórico

tan grande. Registro que bien puede ser utilizado con cualquier fin o simplemente omitido sin ningún otro uso más que el de incrementar su tamaño.

Para estudiar estos repositorios de datos tan grandes resulta poco eficiente y altamente impreciso hacerlo de manera manual. Incluso se abre la puerta a la pérdida de datos. Por ejemplo, cuando se manejan grandes cantidades de texto escrito, esto implica papel en un área de trabajo, en dónde el documento físico se puede perder y/o dañar. Por otra parte, el investigador puede perderse entre tantas líneas de texto. Es éste problema: el manejo de grandes cantidades de documentos, lo que van a buscar resolver los corpus electrónicos, entre otras cosas.

I.3.1. Definición

Por sí solo, el término corpus es la palabra en latín para cuerpo. Por lo tanto, como lo dicen McEnery y Wilson (2001), un corpus puede ser cualquier cuerpo de textos. Mas esta no sería una definición completa para lo que en ésta tesis se estudia. Y es que además, para efectos de la investigación basada en corpus, se tienen que cumplir con ciertos requerimientos.

Un corpus se puede definir, según lo menciona Sierra (2008), como un conjunto de textos que han sido seleccionados bajo ciertas normas y criterios con una finalidad para la cual éste conjunto de textos se vale de técnicas computacionales que le van a permitir generar resultados más exactos; además de las claras ventajas que el mundo digital ofrece, como la reducción en espacio físico, aumento en la cantidad de datos y en la velocidad de consulta.

De modo que para que un conjunto de textos pueda ser considerado como un corpus ha de cumplir con ciertos requisitos. Esto es para que el corpus pueda ser usado como muestra

del lenguaje (Sinclair, 1996). De este modo, según McEnery y Andrew (2001) la representatividad es uno de los requisitos que debe cumplir un corpus para que sea considerado como tal. Además de ser de tamaño finito, pues al ser una muestra de un lenguaje, no debe pretender abarcarlo todo, sino ser una muestra bien definida.

Otra característica más con la que debe cumplir un corpus electrónico es la de encontrarse en formato legible para la computadora. Esto, claro, con el afán de que la información contenida en él pueda manejarse a partir de un equipo de cómputo. Como último requisito y quizá éste no es tan necesario para el buen funcionamiento de un corpus, sí lo es para la validez del mismo. El requisito es contar con las referencias de los textos contenidos.

I.3.2. Concordancias

Una vez que se tiene un corpus, se puede ver como un conjunto de datos listos para ser analizados. Este puede ser un conjunto muy grande, sin duda alguna, pero que requiere de algún tipo de análisis para convertirse en información. Es como parte de la obtención de esa información que aparece el concepto de concordancia. Esta le va a permitir al usuario final ver un fragmento textual de la longitud que el mismo usuario defina, obtenido a partir de alguno de los textos del corpus.

Una concordancia según el Diccionario del español de México es un "Índice, generalmente alfabético, de todas las palabras, expresiones o citas contenidas en una obra, o en un conjunto de obras, con las indicaciones de su localización en ellas y, a veces, con su contexto"[9].

[9] Diccionario del Español de México (DEM) http://dem.colmex.mx, El Colegio de México, A.C., [21/01/2014]

Sin embargo, esta definición no resulta muy clara si no se conoce bien del tema. Es por esto que utilizaré un ejemplo de concordancias del CEMC para dejar más clara la definición, y hablaré brevemente de la diferencia entre obtener concordancias (generación de concordancias) y el proceso de recuperación de información en un corpus.

En los sistemas de consulta como el que ofrece Google, los primeros resultados contienen la o las palabras tal cual se están pidiendo, acompañadas de un fragmento del sitio en donde aparecen. Este fragmento puede o no contener las palabras buscadas, pero aporta cierta información acerca del contenido del sitio. Este tipo de búsqueda califica y ordena los resultados según ciertos criterios. Es un símil de obtener el documento más parecido a la solicitud. A esto se le conoce como recuperación de la información.

En el caso de la búsqueda realizada a los corpus lingüísticos electrónicos, el contexto de las palabras adquiere mucha más importancia. Pues según una de las definiciones anteriores, una concordancia va a permitir que un texto pueda interpretarse adecuadamente. Así que cuando se realiza la generación de concordancias dentro de un corpus, un valor importante dentro de la petición es el tamaño de la ventana de palabras, o visto de otra manera, tamaño del contexto. Esta ventana es una cantidad α de palabras a la izquierda y a la derecha de la búsqueda central. Como se ve en el ejemplo de la Figura 6.

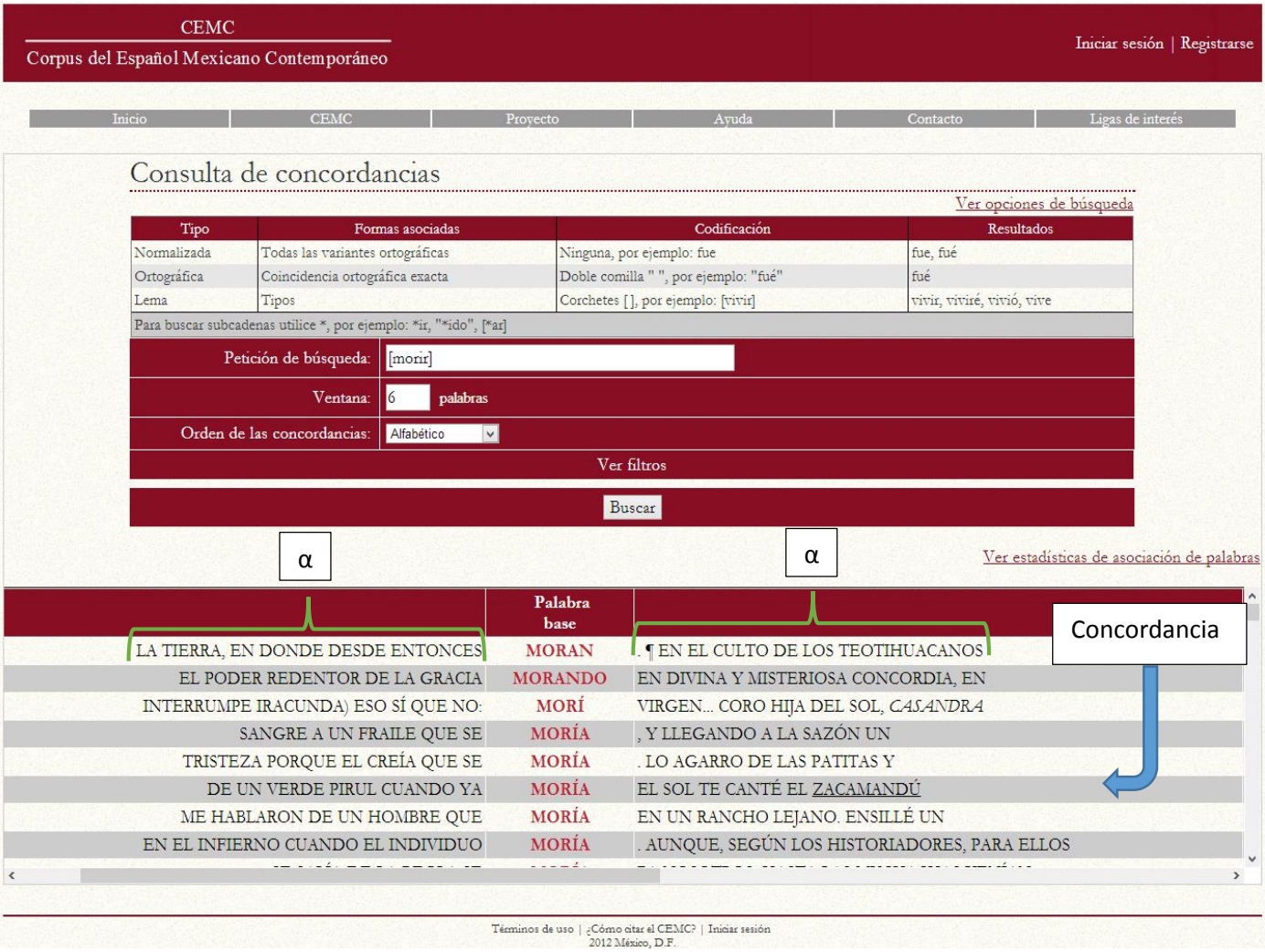


Figura 6. Ejemplo de concordancia

En la Figura 6 se observa una consulta hecha al CEMC donde la palabra buscada o *palabra base*, como lo muestra el sistema, es el lema *morir*. En el panel de búsqueda, mostrado en la parte superior de la imagen, se puede apreciar que el campo de *Ventana* es $\propto = 6$. Entonces, la búsqueda se va a realizar sobre las apariciones en el corpus del lema *morir* y las 6 palabras a la derecha y a la izquierda de la misma. A éste fragmento de 13 palabras se le conoce como concordancia. Al conjunto de fragmentos se le llama lista de concordancias.

I.3.3. Corpus existentes

En internet existen una gran cantidad de corpus electrónicos, de distintos tamaños y con distintos fines. Como por ejemplo el Corpus de Referencia del Español Actual (CREA), creado

con la intención de ofrecer una muestra representativa y equilibrada del español estándar que se utiliza actualmente en el mundo (REAL ACADEMIA ESPAÑOLA, "Corpus de referencia del español actual", s.f.). Otros ejemplos de corpus son los que se describen más adelante en la sección I.4.

I.3.4. Usos de los corpus

Una de las definiciones que se pueden encontrar de corpus es la de "muestra amplia de lengua escrita o hablada" (Ingeniería lingüística, s.f), que no contradice en nada la definición mencionada en ésta tesis, pero que entra a colación ya que permite considerar de mejor manera el uso que se le puede dar a un corpus. Dentro de los varios usos de un corpus se encuentran:

- Analizar la lengua y determinar sus características.
- Como referencia para entrenar algún sistema de cómputo. Por lo general esto se logra ya que el sistema se adapta a ciertas circunstancias específicas.
- Verificar empíricamente una teoría lingüística.
- Ensayar una técnica o aplicación de ingeniería lingüística con el fin de determinar su buen funcionamiento en la práctica.
- Conocer cuándo una palabra fue reemplazada por otra, cuándo cambió su modo de uso, cambió su ortografía, etc. Éste es el caso de un corpus diacrónico como el CHEM.
- Para crear un diccionario. En éste caso, los corpus son usados por los lexicógrafos como evidencia del uso y función de las palabras.

I.4. Arquitecturas de corpus electrónicos

A continuación expongo algunas de las arquitecturas de corpus electrónicos que se encuentran disponibles en internet. Aunque existen muchos corpus de gran tamaño disponibles para

su consulta gratuita en línea, pocos son los que exponen documentación acerca de la arquitectura de su motor de búsquedas. Tal es el caso del CREA o el CORDE, desarrollados por la Real Academia Española (RAE) y que no cuentan con documentación de ningún tipo disponible para su consulta. Sin embargo, también hay corpus como el BNC que utiliza un software llamado XAIRA y que es de código abierto, dando acceso a toda la información del mismo.

I.4.1. *The British National Corpus*

El British National Corpus (BNC) es un corpus de 100 millones de palabras. La última edición del BNC es la edición en XML publicada en 2007. El corpus utiliza un sistema de anotación intertextual, es decir, la anotación es parte del corpus (Davies, 2002. p. 22), así como sucede en el CEMC (véase sección II.1.2). El software que realiza las búsquedas en el BNC es XAIRA (XML Aware Information Retrieval Architecture). Este es una versión de SARA (SGML-Aware Retrieval Application), un software de búsqueda en corpus textuales desarrollado por *Oxford University Computing Services* para su uso sobre este corpus.

SARA o XAIRA son ambos sistemas desarrollados para el BNC y su uso va a depender del formato del corpus (SGML o XML respectivamente). El sistema completo contiene un indexador, que crea archivos-índice y léxico derivado de los documentos XML. Además un servidor, el cuál va a manejar las consultas a la colección, usando los archivos-índice y el léxico. También uno o más programas cliente, los cuales van a permitir el acceso al servidor a partir de diferentes plataformas o ambientes de software. Cabe resaltar que XAIRA es software libre y de código abierto.

El indexado entonces del BNC está hecho utilizando archivos-índice, en los cuales, los términos se encuentran relacionados con su localización en el archivo principal[10]. Haciendo uso de técnicas de hashing se obtiene un rápido acceso a estos archivos principales. El etiquetado va a indicar cómo se van a tokenizar los documentos de entrada. La tokenización puede ser implícita o explícita. En la tokenización implícita XAIRA seguirá las reglas estándar del lenguaje según la codificación UNICODE. Los idiomas utilizados son los especificados en el TEI[11].

En el caso de la tokenización explícita, las etiquetas deberán ser especificadas en el XML como elementos tokenizadores o separadores de palabras. En este caso, todas aquellas etiquetas desconocidas para el sistema, serán ignoradas. Así, los espacios en blanco no serán un indicador para tokenizar. De modo que si la cadena de entrada cuenta con uno o más espacios, éstos serán normalizados y la entrada será dada por las etiquetas reconocidas. Para esto, es necesario un documento adicional en donde se explique el etiquetado, este documento se procesa por el sistema previo al indexado del corpus.

El lenguaje de consulta del corpus es un sistema booleano de recuperación del lenguaje, con algunas características adicionales particularmente útiles para trabajar sobre corpus. Para trabajar con una amplia gama de posibilidades de consulta, el sistema divide una consulta en distintas sub-consultas. Estas sub-consultas o consultas atómicas, como se denominan en la documentación, son los casos más simples que se pueden dar. Así la unión de dos o más consultas atómicas lleva a una consulta compleja o simplemente consulta.

[10] El archivo principal es el documento del corpus donde se encuentra la palabra buscada.
[11] Text Encoding Initiative (TEI) Es un consorcio que desarrolla y mantiene un estándar para la representación de textos en formato digital. (http://www.tei-c.org/index.xml)

Los resultados pueden ser mostrados en cuatro formatos diferentes. Texto plano, en este formato se suprime todo el etiquetado. En POS, palabras individuales son mostradas en un código de colores de acuerdo con la parte de la oración y el esquema de colores que defina el usuario. En XML, todo el etiquetado XML original es mostrado sin mayores interpretaciones. De forma tradicional, el etiquetado XML es interpretado según la especificación del usuario.

I.4.2. Corpus del español

El Corpus del Español es un corpus, que tiene 100 millones de palabras de 1200 a 1900. Es el primer corpus anotado del español histórico y moderno (Davies, 2002, p.21). Según describe Davies (2002. p. 22), los 100 millones de palabras se encontraban, es su versión original, en una base de datos relacional de SQL Server 7.0. Esta base se encontraba indexada con el "Full-Text Indexing" de SQL Server.

En el Corpus del Español, hay una base de datos que contiene cada n-grama en el corpus (desde unigramas hasta trigramas). Para cada n-grama, hay información en la base de datos sobre la frecuencia de cada uno de los siglos desde el siglo XIII hasta el siglo XX, y en los registros del español moderno con los que cuenta el corpus, literatura, oral y miscelánea. Esta base de datos está compuesta de varias tablas en las que se encuentra la información de lema, categoría gramatical, sinónimos, etimologías, etc. Las tablas se vinculan dando forma al sistema de bases de datos relacionales que soporta el motor de búsqueda del Corpus del Español.

La tabla central de n-gramas se va a relacionar con las demás tablas que contienen la información lingüística, generando una especie de diccionario con cada uno de los n-gramas y la información de los unigramas correspondientes. Para darle velocidad a este diccionario,

cada una de las columnas tiene su propio índice. Estos índices son creados a nivel físico, lo que brinda mayor agilidad a la recuperación de datos.

I.4.3. Corpus Técnico del IULA

La Universitat Pompeu Fabra, con su Institut Universitari de Lingüística Aplicada (IULA) tiene en línea el Corpus Técnico del IULA. Este corpus en 2004 reportó tener más de 22 millones de palabras (Cabré y Bach, 2004, p. 174). El corpus comprende además documentos paralelos, con el objetivo de facilitar estudios de traducción. Su consulta es libre y en línea mediante el uso de un programa de exploración denominado BwnanaNet. Este corpus además de utilizar BwanaNet para su exploración, fue indexado utilizando el IMS Corpus Workbench (CWB).

El CWB es una colección de herramientas de código abierto para la administración y consulta de corpus de gran tamaño con anotaciones lingüísticas. Su componente central es un procesador de consultas CQP. Estas herramientas constan de una API en Perl desarrollada para el CWB y que contiene códigos requeridos por varias interfaces web. Además, una API desarrollada en C para un acceso de bajo nivel. Esta API da acceso directo de bajo nivel al corpus indexado del CWB. No es posible ejecutar consultas desde el CQP sin esta API.

El indexado en la biblioteca del corpus o CL (Corpus Library) está dado utilizando archivos índices y tablas hash como modelo de datos. Para el CWB un registro es igual a un token más sus anotaciones. Todas las anotaciones están hechas en una codificación ASCII/LATIN-1. El indexado está basado en operaciones de ordenamiento (Evert, s.f.).

II. La arquitectura de corpus electrónicos del GIL

En éste capítulo se hará un análisis más profundo sobre la arquitectura de los corpus electrónicos del GIL. Entendiéndose como arquitectura a todo el sistema de consulta de un corpus desde el indexado de los textos hasta la interfaz web. Aunque en cuanto a ésta última parte se refiere, no haré énfasis ya que es algo se decidió dejar fuera del alcance de ésta tesis.

Aunque el GIL ha puesto en línea distintos corpus, me voy a centrar en aquellos que comparten su arquitectura con el CHEM. Esto es, aquellos que usan el mismo generador de concordancias y un mismo diseño de la base de datos, el mismo indexador e incluso el mismo formato de etiquetado para sus textos, además claro de una interfaz casi idéntica en cuanto a funcionalidad se refiere.

II.1. Los corpus del GIL

Como ya lo mencioné previamente en la introducción de este trabajo, el GIL ha acumulado una gran cantidad de trabajos. Estos se pueden clasificar en distintas áreas, aunque todas ellas dentro de la Ingeniería Lingüística. Dentro de estos trabajos destacaré los desarrollos y esfuerzos hechos por el grupo en el área de corpus electrónicos. Y es que el GIL a lo largo de su trayectoria ha logrado poner en línea distintos corpus con todo lo que esto requiere.

Entre las tareas que deben realizarse se cuentan: la recopilación de los textos o grabaciones, la digitalización de esto, proporcionar un espacio de almacenado para el corpus, realizar una interfaz de consulta, colocar en línea. Todas estas actividades y muchas más son las que se tiene que llevar a cabo cuando se desea generar un corpus y dejarlo disponible para su consulta a través del internet.

II.1.1.El CHEM

El Corpus Histórico del Español en México es el corpus base del cual se desprenden las arquitecturas del CEMC, COCIEM, entre otros corpus del GIL. En eso radica la importancia de éste corpus para esta tesis. En cuanto a los textos que lo conforman se refiere, el corpus está compuesto por una colección de documentos diacrónicos del español de México que representan diversos géneros textuales. Los documentos con los que cuenta el CHEMC datan de los siglos XVI al XIX (Medina y Méndez, 2006).

Debido a su naturaleza diacrónica, el CHEM cuenta con un procesamiento de sus textos tal que todas las palabras se puedan encontrar sin que la ortografía sea una limitante. En la Figura 7 se puede apreciar una consulta de la palabra *ahora* para los textos comprendidos entre los años 1500 y 1600. Los resultados claramente varían en cuanto a su ortografía, como se ve en los elementos resaltados.

Figura 7. Consulta "ahora" al CHEM

Dada la necesidad de estandarizar el contenido, fue que se definió cierto etiquetado. Este se ha seguido utilizando en los corpus del GIL posteriores al CHEM. Este estándar ha servido para generar los documentos XML que forman parte de éste corpus y del CEMC.

En cuanto al indexador se refiere, el CHEM basa su sistema en las bases de datos relacionales (cuya inspiración se tomó de la propuesta de Mark Davies). Haciendo uso de tablas, vistas y secuencias es que el CHEM almacena y ordena sus tokens y sus características. El manejador utilizado es PostgreSQL, el cuál es software libre y permite la comunicación sencilla con el lenguaje de programación JAVA. Lenguaje en el que está desarrollado el sistema.

Tanto el indexador como el generador de concordancias están desarrollados en el lenguaje de programación JAVA. El lenguaje le da una característica muy importante y muy valiosa, que es la independencia del sistema operativo. Característica que se va a ver reflejada en el aumento de la vida útil del desarrollo, al permitir el rehúso de código con pequeñas modificaciones que le permitan acoplarse al sistema operativo anfitrión.

El CHEM así como el CEMC y el COCIEM, son corpus del GIL que residen en servidores del Instituto de Ingeniería, UNAM. Estos servidores cuentan con un sistema operativo Linux, el cual no tiene problema alguno para ejecutar una vez instalada la máquina virtual de JAVA[12]. Del mismo modo si se quisiera migrar cualquiera de los corpus a un anfitrión con un sistema operativo distinto sólo habría que instalar esta máquina virtual.

[12] Se conoce como máquina a virtual de java al software que emula la existencia de un equipo huésped dentro de un equipo anfitrión, completamente independiente. Que sólo va a permitir la ejecución de software JAVA. Se puede conceptualizar como dos máquinas o más en un mismo hardware.

Ya que el desarrollo tanto del indexador como del generador de concordancias está hecho en JAVA, para su puesta en línea se requiere de un servidor web que soporte la ejecución de aplicaciones en éste lenguaje. El servidor seleccionado por el GIL y que además resulta muy conveniente por su sencilla instalación e interacción con los distintos sistemas operativos es Apache Tomcat. Software libre e independiente del sistema operativo también.

El paradigma de programación utilizado es el de la programación orientada a objetos.

> Un paradigma de programación es una forma de conceptualizar en que consiste la ejecución de un programa y cómo deben estructurarse y organizarse las tareas que se llevan a cabo en esa ejecución. (Cachero, Ponce de León y Saquete, 2006, p. 2).

Así el paradigma orientado a objetos busca estructurar su código como si fuesen objetos reales, con características y actividades propias de cada uno de ellos. De éste modo es que cada objeto tiene sus propios datos y métodos o acciones. Los objetos y métodos, se vinculan mediante un mismo dominio que es la clase. Las acciones podrán ser ejecutadas desde cualquier otra clase que lo desee siempre y cuando instancie el objeto correspondiente.

Ya que la obtención de concordancias y el desglose para indexar los archivos no son tareas sencillas en lo absoluto ya que por parte de la generación de las concordancias hay que realizar un procesamiento de la consulta, la obtención de la base de datos de la misma, la obtención de la concordancia a partir del archivo, el almacenado de lo obtenido y la presentación en línea de la solución, para todos y cada uno de los resultados. La programación orientada a objetos busca dividir el problema en pequeñas partes y simplificar la solución.

II.1.2.El CEMC

El Corpus del Español Mexicano Contemporáneo (CEMC) es un corpus que fue creado debido a la necesidad de contar con una base de textos para la elaboración del Diccionario del español de México[13]. El objetivo central de éste corpus es el de reconocer y señalar las características del español mexicano contemporáneo.

El CEMC está formado por 996 obras y grabaciones, de las que se extrajeron párrafos de manera aleatoria, todas estas extracciones con un promedio de 1898.6 palabras gráficas por extracción. En total el corpus tiene un tamaño de 1,891,045 ocurrencias. Mismas que se encuentran clasificadas en catorce géneros, los cuales van a servir como filtros dentro de la búsqueda, permitiendo así una búsqueda más específica.

La arquitectura del CEMC es muy parecida al CHEM. Desde el formato de los textos contenidos ya que ambos se cuentan con un corpus de textos en XML, hasta la interfaz web. Sin embargo, no todo es tan parecido, y es que si bien ambos corpus usan el mismo estándar de etiquetas, el CEMC implementa distintas etiquetas en sus textos. Al no ser diacrónico, la etiqueta para normalizar la palabra no tiene uso y por lo tanto los textos no la van a contener. Por su parte el CHEM no utiliza el etiquetado POS.

En el CEMC, la implementación de filtros dentro de la búsqueda es otra característica que lo diferencia del CHEM. Estos filtros van a permitir hacer búsquedas más específicas al organizar los textos en distintos grupos. Estos pueden ser según el uso del lenguaje, el nivel de la lengua y el género del texto.

[13] Del Diccionario del español de México se puede encontrar una versión en línea en http://dem.colmex.mx así como una versión impresa, publicada por El Colegio de México, A.C.

Pero no sólo en el etiquetado hay diferencias entre los corpus, el tamaño es una diferencia notable entre ellos también. Y es que el CEMC es aproximadamente seis veces el tamaño del CHEM, esta diferencia se nota de inmediato en los tiempos de todos los procedimientos. El indexado, la generación de concordancias y la generación de estadísticas son los procesos que hacen de la misma manera ambos corpus y que se han visto afectados por el cambio en el volumen de datos que se está procesando. Sin embargo, será más adelante que ahondaré en los detalles de ésta arquitectura.

II.1.3. Otros

Los anteriores no son los únicos corpus del GIL, como ya lo he mencionado anteriormente. Actualmente el Corpus Básico Científico del Español de México (COCIEM) es un corpus aún más grande que el CEMC. El cuál cuenta con un aproximado de cuatro millones de palabras y la misma arquitectura del CHEM, sin embargo, aún no está disponible en línea[14].

Otros de los corpus desarrollados por el GIL, algunos hechos en colaboración con otras instituciones, son el Corpus de Contextos Definitorios (CORCODE), el Corpus de las Sexualidades en México (CSMX), el Corpus Anotado con Relaciones Discursivas (RST Spanish Treebank) y el Corpus Lingüístico en Ingeniería (CLI).

El CORCODE comparte la arquitectura de su motor de búsqueda con el CHEM y el CEMC, aunque la conformación de sus textos es distinta a la de éstos dos corpus. Éste corpus está integrado por contextos definitorios obtenidos a partir de diversos corpus (Universidad Nacional Autónoma de México, 2010).

[14] Al momento de la elaboración de ésta tesis, el COCIEM aún no se encontraba disponible para su consulta en línea.

El CSMX presenta también una interfaz de consulta y una arquitectura muy parecidas a las que presenta el CHEM. Este corpus está constituido por una colección de documentos sobre sexualidad y sexología. El Corpus Lingüístico de Ingeniería coordinado por el GIL con la participación del Instituto de Ingeniería de la UNAM, fue desarrollado antes que el CHEM y cuenta con una arquitectura y un diseño completamente distinto.

Otro corpus distinto al CHEM, CEMC, COCIEM, etc., es el RST *Spanish Treebank*, mismo que fue desarrollado por el GIL en colaboración con el grupo Iulaterm del *Institut Universitari de Lingüística Aplicada* (IULA) de la *Universitat Pompeu Fabra* (UPF) y con el quipo TALNE del *Laboratoire Informatique d' Avignon* (LIA) de la *Université d' Avignon et des Pays de Vauciuse* (UAPV). Además de ser un corpus de textos especializados en español, el RST Spanish Treebank permite al usuario subir su material para ser analizado por el corpus, siempre y cuando cumpla con los criterios requeridos (da Cunha, Torres-Moreno y Sierra, 2011).

II.2. Arquitectura del CEMC

En éste apartado, haré un análisis más detallado acerca de la arquitectura del CEMC, comenzando por el etiquetado de sus documentos, mismos que se encuentran en XML. Posteriormente hablaré del motor de búsqueda y todo lo que conlleva. El indexador, el generador de concordancias y la interfaz de consulta son partes que se verán dentro del motor de búsqueda.

II.2.1.Etiquetado

Aunque no es parte del alcance de ésta tesis el declarar si el etiquetado usado es el óptimo o no, mencionar las ventajas y desventajas que éste representa para el corpus sí lo es. Es por esto que a continuación describiré las etiquetas utilizadas dentro de los textos del corpus.

Algunas de las cuales son usadas dentro de los textos, otras en los documentos de salida[15] y otras utilizadas en los documentos de referencia.

El XML es un lenguaje de marcado de textos. Es un lenguaje que al mismo tiempo que permite estructurar un documento, da la libertad de colocar las etiquetas que el usuario desee facilitando así el uso de este lenguaje. Con XML el usuario es libre de diseñar su propio lenguaje de marcado que mejor se acomode a la información que está utilizando. Esta característica de personalización permite estructurar los textos del corpus y etiquetarlos de una manera lógica y personalizada.

Los textos que conforman el corpus cuentan con las etiquetas mostradas previamente (Tabla 1 y Figura 5) y otras en el encabezado del texto. Algunas de las cuales se muestran en la siguiente tabla (Tabla **3**).

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="h/estilo.xsl"?>
<documento xmlns="http://www.ii.unam.mx" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
    <encabezado id="00500" permisos="todos">
        <titulo>El diosero</titulo>
        <parametros genero="Literatura" nivelLengua="Lengua culta">
            <corpus>
                <CEMC textos="Obras de literatura" uso="Español estándar"/>
            </corpus>
        </parametros>
        <hablante id="h00500"/>
        <referencia id="r00500">
            <lugar>México</lugar>
            <fechaOriginal>1960</fechaOriginal>
        </referencia>
        <responsables>
            <etiquetador nombres="Automático" apellidos="Automático" fecha="2011"/>
            <revisor nombres="" apellidos=""/>
        </responsables>
    </encabezado>
    <cuerpo>
        <p>
            <g c="conj" l="y">Y</g>
            <e/>
            <bastardillas>
                <g c="nomp" l="simón">SIMÓN</g>
            </bastardillas>
            <r c="Fc">,</r>
            <e/>
            <g c="adv" l="como">COMO</g>
            <e/>
            <g c="conj" l="si">SI</g>
            <e/>
            <g c="verb" l="volver">VOLVIESE</g>
            <e/>
            <g c="cont" l="del">DEL</g>
            <e/>
            <g c="nom" l="sueño">SUEÑO</g>
            <r c="Fc">,</r>
```

DOCUMENTO		
Título:	El diosero	
Número ID:	00500	
Permisos:	todos	
Referencia bibliográfica:	Rojas González, Francisco. *El diosero*.; México, 1960	
CORPUS		
Textos:	Obras de literatura	
Uso:	Español estándar	
HABLANTE		
Nombre:	Francisco ROJAS GONZÁLEZ	
Género:	desconocido	
RESPONSABLES		
Etiquetador:	Automático AUTOMÁTICO	2011
Revisor:	Anónimo	

Y *SIMÓN* , COMO SI VOLVIESE DEL SUEÑO , COMO SI HUBIESE SIDO SUSTRAÍDO POR LAS DESTEMPLADAS PALABRAS DE U

FRENTE AL MÉDICO , UN VIEJO AMABLE Y BROMISTA , *SIMÓN* EL INDIO ZOQUE NO TUVO NECESIDAD DE HABLAR MUCHO

ELLA TAMBIÉN ERA DE *BACHAJÓN* , PEQUEÑA , REDONDITA Y SUAVE . DÍA CON DÍA , CUANDO IBA POR EL AGUA AL RIAC

CRUDO , UN CÁNTARO REDONDO Y BOTIJÓN , AL QUE NUNCA DABAN FIN AQUELLAS MANOS DISTRAS E INCANSABLES ...

Figura 8. Ejemplo 1 de tratamiento XML

[15] Le llamo documento de salida, a los documentos generados por el sistema para guardar las respuestas a las consultas y presentar estas respuestas al usuario. Como salida final del sistema.

Etiqueta	Concepto
<?xml version="1.0" encoding="iso-8859-1"?>	Muestra la versión del XML, así como la codificación de los caracteres. Esta se requiere para una correcta representación de caracteres especiales. Tales como, acentos, tildes, diéresis, etc.
<?xml-stylesheet type="text/xsl" href="h/estilo.xsl"?>	Indica la hoja de estilo utilizada como plantilla para la generación del XML.
<encabezado id="00500" permisos="todos">…</encabezado>	Entre estas etiquetas (encabezado), el texto va a contener datos de identificación del texto, así como algunos otros de clasificación. El título, el lugar de creación, la fecha, el género al que pertenece, el nivel de la lengua, son algunos de los datos que aparecen entre éstas etiquetas. *Id* y *permisos* son etiquetas que indican el nombre del documento y los permisos de consulta respectivamente.
<titulo>…</titulo>	Entre estas etiquetas se va a encontrar el título del documento.
<parametros genero="Literatura" nivelLengua="Lengua culta">…</parametros>	Estas etiquetas contienen dos de los tres filtros de búsqueda con los que cuenta el CEMC. Que además son información sobre el tipo de texto que contiene. *Parametros* establece el inicio de una estructura de etiquetas, al mismo tiempo que es un nivel dentro de la estructura *encabezado*. El género y el nivel de la lengua.
<corpus> <CEMC textos=”Obras de literatura” uso=”Español estándar” /> </corpus>	Este es un conjunto de etiquetas, en las que se puede observar una estructura que contiene el nombre del corpus al que pertenece el texto, el origen del texto y el uso del lenguaje. Ésta característica última es uno más de los filtros usados en la búsqueda.
<referencia id = “r00500”> <lugar>México</lugar> <fechaOriginal>1960</fechaOriginal> </referencia>	Debido a que algunos textos y grabaciones comparten datos de autor y título, se optó por separar estos valores en un archivo de referencias aparte, en donde el *id* que se muestra en la etiqueta, es lo que va a permitir obtener estos datos. El lugar y la fecha son datos de referencia que se encuentran contenidos dentro de la estructura *referencia* y que van a ser mostrados por el navegador web.
<responsables> <etiquetador nombres=”nombre” apellidos=”apellido” fecha=”año” /> <revisor nombres=”nombre” apellidos=”apellido” />	En estas etiquetas se guardan los datos de las personas que trabajaron en el texto, como etiquetadores y/o revisores. Estos valores son para mero control interno sobre el trabajo realizado en el texto. Ya que no se van a mostrar en el explorador.

</responsables>	
<cuerpo>	Indica el inicio del cuerpo del texto. A partir de ésta etiqueta los valores que siguen serán procesados e indexados.

Tabla 3. Etiquetas del encabezado de un texto del CEMC.

En la Figura 8 se muestra del lado izquierdo un fragmento de uno de los textos, abierto desde un procesador de textos. Del lado derecho se muestra el mismo texto y fragmento pero abierto desde un navegador web. De la imagen en el lado izquierdo se pueden observar las líneas del texto con un margen distinto cada una, esto es por la estructura a la que pertenecen y las etiquetas con que cuentan.

Del lado derecho en la Figura 8, se aprecia una tabla con los valores del encabezado del texto y abajo el texto en sí, en prosa y bajo el mismo margen. Esto es porque el navegador procesa las etiquetas y sólo muestra el contenido.

Como se definió anteriormente, el CEMC incorpora una nueva opción de búsqueda a las posibilidades heredadas de la arquitectura del CHEM: los filtros, mismos que se pueden ver indicados en el etiquetado de cada uno de los textos del corpus. Estos filtros se pueden ver como una manera de agrupar lo textos, que va a permitir realizar búsquedas más específicas sobre el corpus y un análisis distinto de los datos obtenidos.

Los filtros de los que he venido hablando son los que se pueden ver en la Figura 10 a continuación. Un documento va a pertenecer a los tres principales grupos (Uso, Nivel y Género), no así a dos sub categorías de un mismo grupo. Un ejemplo más claro se ve en la Figura 9, dónde el texto pertenece a los tres conjuntos principales y a sólo un subconjunto.

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="h/estilo.xsl"?>
<documento xmlns="http://www.ii.unam.mx" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://corpus.iingen.unam.mx/~vmijangosc/esquema.xsd">
    <encabezado id="76602" permisos="todos">
        <titulo>El Cupido (versión 1)</titulo>
        <parametros genero="Lírica popular" nivelLengua="Lengua sub-culta">
            <corpus>
                <CEMC textos="Habla regional" uso="Español estándar"/>
            </corpus>
        </parametros>
        <hablante id="Desconocido"/>
        <referencia id="r76602">
            <lugar>Veracruz</lugar>
            <fechaOriginal>1965</fechaOriginal>
        </referencia>
        <responsables>
            <etiquetador nombres="Automático" apellidos="Automático" fecha="2011"/>
            <revisor nombres="" apellidos=""/>
        </responsables>
    </encabezado>
```

Filtros

Uso: Español estándar

Género: Lírica popular

Nivel de la Lengua: Lengua sub-culta

Figura 9. Muestra de los filtros en un documento

Con el uso de éstas etiquetas, las búsquedas se pueden realizar de muchas más maneras, dejando fuera todos los datos que no pertenezcan a los filtros seleccionados. Ésta exclusión se da también en la generación de estadísticas.

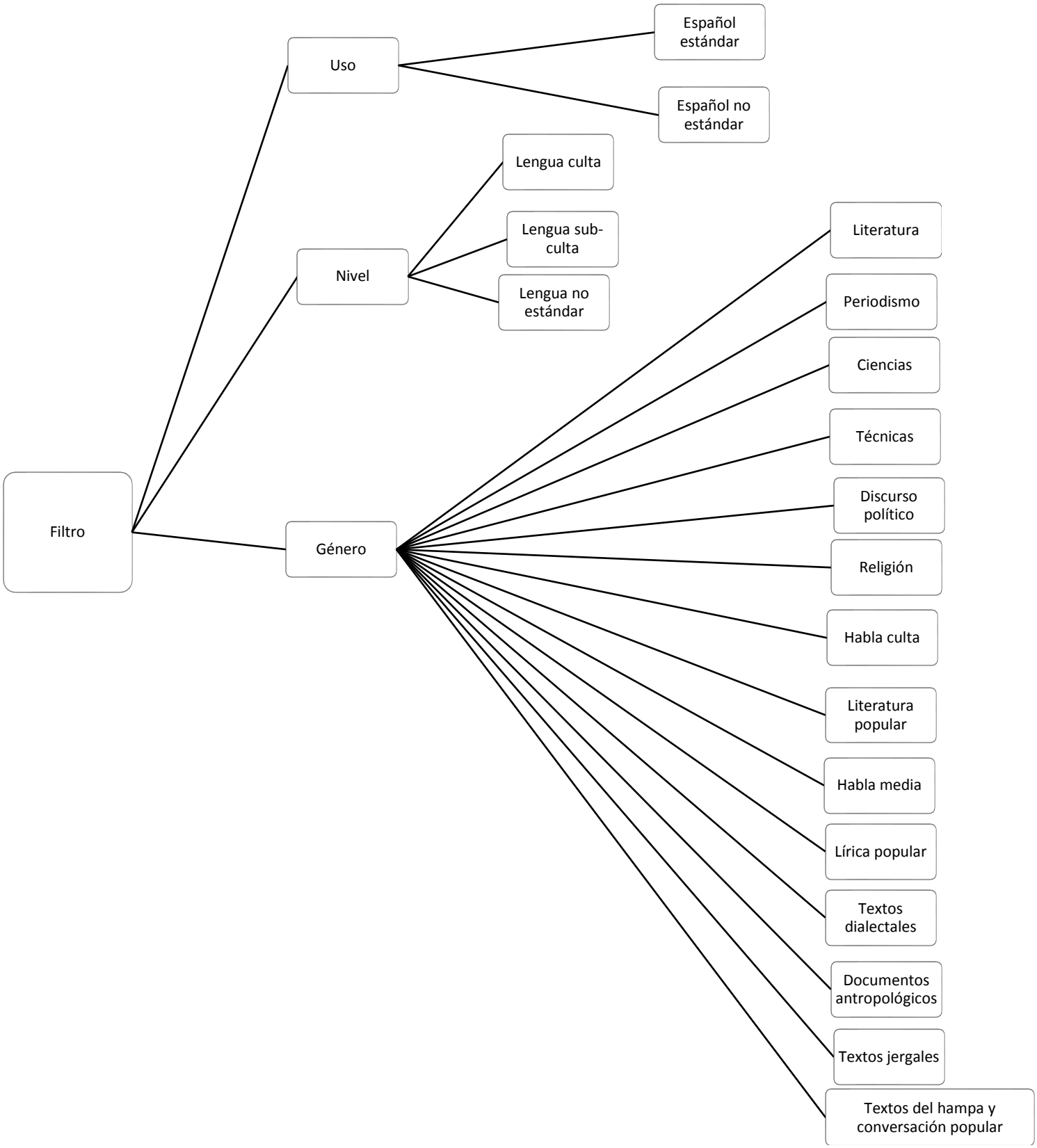


Figura 10. Esquema de filtros del CEMC

II.2.2.Motor de búsqueda

Un motor de búsqueda es una herramienta basada en web que permite al usuario localizar información en internet (What is a search engine?, 2001). Esta definición se ajusta a los motores de búsqueda más comunes en internet como son Google, Yahoo y Bing. Sin embargo, son las partes de un motor lo que incluyen a este sistema dentro de la definición, ya que se cuenta con un sistema de indexado y un mecanismo de búsqueda (Oller, 2003).

El sistema de indexado de los corpus del GIL contiene un método para organizar la información (las bases de datos) y el mecanismo de búsqueda incluye un generador de concordancias y una interfaz de consulta. Estos componentes de la arquitectura actual serán detallados en las siguientes secciones: II.2.2.1 sobre la base de datos, II.2.2.2 sobre el indexador, II.2.2.3 sobre el generador de concordancias y II.2.2.4 sobre la interfaz de consulta.

II.2.2.1. Bases de datos

Las bases de datos relacionales es la manera en la que el CEMC actualmente estructura su información. Dentro de una gran base de datos es que este corpus almacena cada token del corpus junto con su información. La base de datos del CEMC es un conjunto de tablas, vistas y procedimientos que se relacionan entre sí, para almacenar la información, realizar la búsqueda de la cadena de entrada y generar las concordancias solicitadas.

Dentro de la información almacenada en la base de datos se encuentra la del documento. En esta tabla se van a guardar desde datos físicos del documento, como ubicación (directorio) y nombre del archivo hasta los datos bibliográficos del mismo, tales como el título del texto, género, nivel de la lengua que maneja, uso entre otros. Dentro de esta tabla

se van a guardar también valores sobre los permisos del documento y estado dentro del corpus.

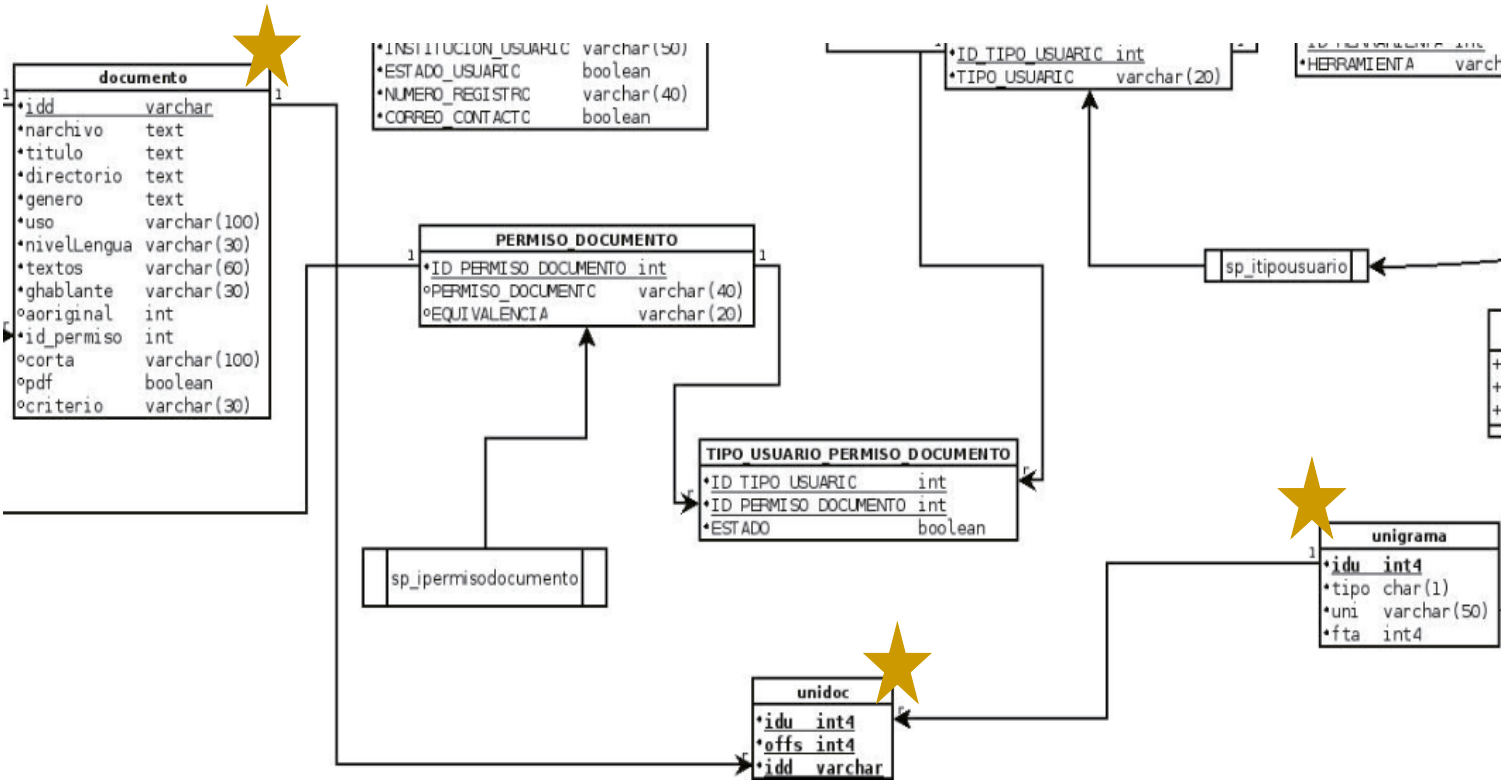


Figura 11. Tablas de documento, unigrama y unidoc de la base de datos del CEMC.

Cada palabra va a ser almacenada dentro de otra tabla aparte. En esta tabla se va a guardar un identificador único para cada type o unigrama, el tipo (palabra gráfica, lema), el type y su frecuencia total absoluta. Esta tabla junto con la tabla de documento van a dar vida a la tabla en la que se van a almacenar los archivos a manera de identificadores y bytes. Es decir, dentro de una tabla aparte se va a almacenar el identificador de un type, el identificador del documento en donde se encuentra contenido y su ubicación en bytes dentro del documento. Estas tablas son las que se encuentran marcadas con una estrella dentro de la Figura 11.

El proceso de llenado de la base de datos se lleva a cabo en dos partes. Por una parte se hace un llenado de la base bibliográfica del corpus, la cual se va a almacenar dentro de la tabla documento. Esta base se construye a partir de los encabezados de los textos en XML y contiene los datos bibliográficos de cada documento. Por otra parte, se lleva a cabo la construcción de la tabla de unigramas.

En la tabla *unigrama* se va a guardar una vez cada type del corpus y sobre este registro se va a incrementar su frecuencia absoluta según la cantidad de apariciones del type dentro del corpus. Una vez que una palabra ha sido ingresada dentro de la tabla unigrama y se cuenta con su información bibliográfica, dentro de la tabla unidoc se va a generar la relación entre ambos al guardarse el identificador de la palabra, el identificador del documento y el número de bytes necesarios para acceder al n-grama en el documento.

Estas son algunas de las tablas que se encuentran dentro de la base de datos. La base también cuenta con información sobre los permisos de los documentos y de los usuarios registrados al CEMC, del mismo modo lleva un registro sobre las consultas hechas por usuarios no registrados. También lleva una bitácora sobre las consultas realizadas y sobre las estadísticas previamente generadas. Estas últimas dos tablas se van a consultar en dos momentos distintos durante un acceso al sistema. Antes de generar las concordancias y después de haberlas generado.

La primera tabla que se va a consultar es la que guarda las peticiones o búsquedas previas. Esta consulta se va a realizar después de codificar la consulta del usuario, en este momento se verifica que sea una consulta nueva, de lo contrario el sistema únicamente buscará en un repositorio de archivos el XML el documento de concordancias correspondiente previamente almacenado (lo que agiliza el tiempo de respuesta). Este mismo proceso se sigue

con las estadísticas. En case de que se encuentre que las estadísticas fueron realizadas previamente, las estadísticas ya no se recalculan y únicamente se extraen del documento almacenado.

Un segundo momento en que estas tablas se vuelven a consultar se da cuando la generación de las concordancias y de estadísticas ha terminado. Cuando se recibió una nueva consulta, se guarda el registro dentro de una de las tablas de la base de datos con la clave de la consulta y un contador para conocer la cantidad de búsquedas que ha tenido esta expresión. La generación de las concordancias se da en un proceso intermedio entre estas dos consultas y es el que definiré a continuación.

II.2.2.2. Indexador

El indexador del CEMC es un software realizado en JAVA bajo el paradigma de la programación orientada a objetos. Fue un proyecto desarrollado originalmente para realizar el indexado a partir de una interfaz web. Posteriormente fue modificado para que el indexado se hiciera vía terminal de comandos y fuera de línea. Esto por el tiempo que tarda el indexado y también por la cantidad de intentos que en ocasiones podía requerirse para lograr un indexado correcto.

El indexado comienza identificando el sistema operativo en el que se encuentra, esto es de importancia por las direcciones que se deben guardar dentro de la base de datos para indicar la ubicación de los archivos. Durante el indexado, el sistema realiza muchas conexiones a la base de datos, algunas para la inserción de datos y otras para la obtención de datos.

El procesamiento del corpus es secuencial, de modo que hasta que no se termina de indexar por completo un archivo no se comienza con el siguiente. El procedimiento general

para el indexado del corpus comienza con una decisión, en ella se especifica si se desea ejecutar el procedimiento para reiniciar la base de datos. En algún momento esta fue la única opción dentro del indexado, teniendo que reiniciar la base de datos cada vez que se deseaba agregar un nuevo documento al corpus.

El siguiente paso es la apertura y lectura del directorio que contiene el corpus. En este paso se van a obtener las rutas absolutas de todos los documentos en XML del corpus, he aquí la importancia de conocer el sistema operativo de la máquina anfitrión. La razón por la que se obtienen las rutas de los documentos en XML es porque se debe conocer la estructura del directorio donde se encuentra el corpus. En esta estructura están los documentos en XML que forman el corpus y una carpeta de nombre *h* en la que se van a encontrar documentos de apoyo para la carga. El ejemplo del directorio se ve en la Figura 12.

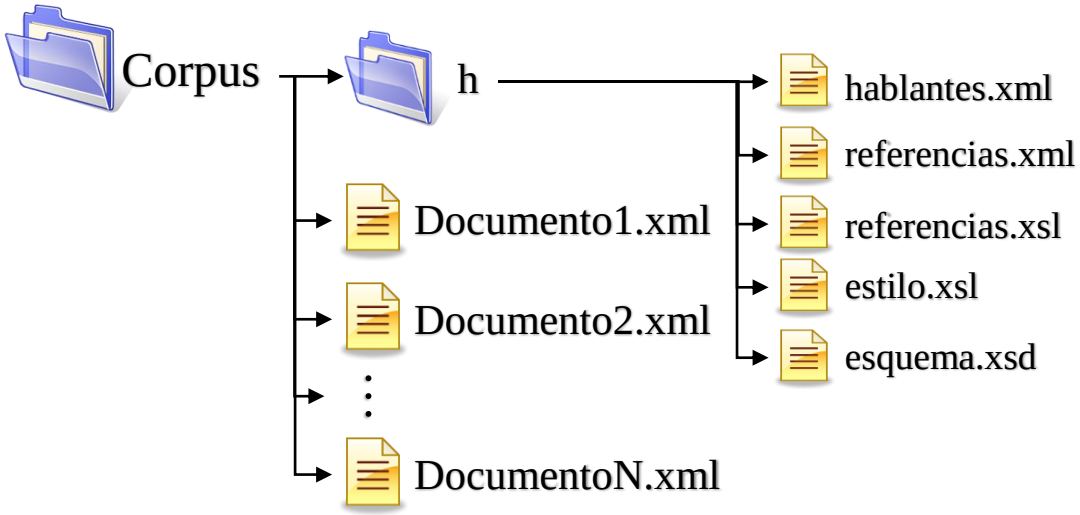


Figura 12. Ejemplo del organigrama del directorio de un corpus para la arquitectura actual.

Los documentos que se pueden observar dentro de la carpeta *h* en la Figura 12 van a servir como referencia durante el indexado y como plantilla durante la generación de concordancias. Los que tienen extensión XML son documentos con información de apoyo durante

el indexado y también en la generación de las concordancias, ya que contienen información bibliográfica relacionada con códigos llamados: referencias cortas (estas indican la fuente documental de donde se obtiene la concordancia). En el caso de los documentos con extensión XSL y XSD son plantillas de diseño y formato XML que se usan en la generación de nuevos XML durante la generación de las concordancias.

Una vez que se ha leído el directorio del corpus y se tienen en memoria temporal todos los nombres de los archivos XML, el siguiente paso es el procesamiento de cada uno de estos archivos. Uno por uno, cada archivo es procesado para llenar con su información la base de datos. Como ya mencioné previamente, el primer proceso a realizar dentro del indexado es la creación de la base bibliográfica, ya que sin esta información el resto del indexado generaría un error y no se llevaría a cabo. Mediante el uso de una función recursiva, se va buscando dentro de las etiquetas hasta encontrar las correspondientes a los siguientes valores:

- El identificador del archivo.
- El título del texto.
- Género del texto.
- Uso del documento.
- Nivel de la lengua.
- Tipo de texto.
- Identificador del hablante.
- Género del hablante.
- Fecha original.
- Permisos del documento.
- Permisos dentro de la base de datos.

- Referencia de algún documento en formato PDF, en caso de que exista.
- Referencia corta

El siguiente paso es el llenado de la tabla de unigramas. Este proceso se realiza en conjunto entre el sistema en JAVA y procedimientos almacenados en Postgresql. En el sistema se van a extraer los unigramas junto con su información lingüística y una vez que se insertan se ejecutan los procedimientos en Postgresql que van a actualizar la frecuencia y sus relaciones con las demás tablas. Una vez indexado el corpus, el sistema está listo para realizar consultas. Este procedimiento ya se hace estando en línea y es ejecutado por el generador de concordancias detallado en la siguiente sección.

II.2.2.3. Generador de concordancias

Ya con el indexado realizado, el siguiente paso es la extracción de la información. Pero en este caso específico, la extracción de la información no se puede realizar tan solo con consultas en Postgresql que obtengan la palabra, sus datos lingüísticos y bibliográficos. La información que se requiere son concordancias, por lo que la consulta va a variar porque se requiere de conocer más de una palabra, la información bibliográfica de estas y extraerlas en el formato en el que se encuentran en el texto. Es para esto que se desarrolló este generador de concordancias.

Al igual que el indexador, el generador de concordancias es un sistema desarrollado en JAVA y siguiendo el paradigma de la programación orientada a objetos. El software requiere de un servidor con Apache Tomcat para poder ejecutarse correctamente. En un principio realiza una serie de comprobaciones antes de estructurar una consulta para la base de datos. Estas comprobaciones incluyen que la ventana de palabras sólo contenga números y

que esta cantidad se encuentre entre 0 y 100. Si la ventana se encuentra vacía entonces le asigna en automático un valor de 10.

Otras verificaciones que se realizan se hacen contra una lista de palabras no permitidas tales como insert, truncate, drop, entre otras y ciertos caracteres especiales (para evitar SQL "inyectado"). Una vez que la consulta ha pasado las comprobaciones entonces el sistema tomará la petición y en base a la palabra o palabras de entrada, la ventana, el tipo de búsqueda seleccionado, el orden deseado y los filtros se va a generar una cadena de identificación para esta consulta. Con esta cadena se van a verificar dos cosas antes de la extracción de las concordancias, si la petición ya fue realizada y si las estadísticas ya fueron realizadas también. Estas consultas son hechas a la base de datos.

El siguiente paso es la construcción de la consulta, en este paso se van a cambiar los comodines leídos en la interfaz de consulta por los aceptados por SQL (por ejemplo, * cambia a %) y se van a agregar los filtros a la consulta así como las condiciones de selección y el orden en caso de que existan todos estos datos. Todo esto para obtener una tabla con las palabras buscadas y las posiciones en bytes de: la primera palabra de la concordancia, la palabra inmediata a la izquierda y derecha de la palabra buscada y la última palabra de la concordancia.

A partir de esta tabla es que se va a abrir el archivo y a extraer la concordancia que se ajuste a los valores obtenidos. Mediante el uso de flujos de datos se van a abrir los archivos todos ellos con una codificación ISO-8859-1. La información de la referencia será extraída de la base de datos y comparada con la ubicada en el archivo referencias.xml, definido en la sección anterior, para ser plasmada en el documento XML de salida. El siguiente paso es el llenado de un arreglo con la concordancia. Este llenado se hace de manera cíclica y dentro

del arreglo se van a incluir palabras y etiquetas, tanto de información lingüística como de estilo.

Una vez extraídas las concordancias, la información va a ser escrita en un documento XML de salida el cuál será leído por el explorador web. Posterior a la escritura y antes de continuar a mostrar la información al usuario, se obtienen las estadísticas de la palabra consultada en caso de que no existan. Las estadísticas también van a ser escritas dentro de un documento XML que quedará listo para su carga en línea, pero que no se va a mostrar en la misma pantalla que las concordancias.

Ya con toda esta información capturada, el sistema que fue ejecutado desde un archivo JSP en línea, guarda el documento XML en la carpeta correspondiente dentro del servidor y posteriormente muestra la información de las concordancias obtenidas. El siguiente paso, que es transparente para el usuario, es el almacenamiento de las estadísticas en memoria temporal, en espera de que el usuario cambie de ventana para acceder al documento XML correspondiente. La muestra y captura de la información corresponden a la interfaz de consulta del sistema, misma que describo en la siguiente sección.

II.2.2.4. Interfaz de consulta

La interfaz de consulta del CEMC es, en funcionalidad la misma que la de su predecesor: el CHEM. Cabe mencionar que este es un punto dentro del motor de búsqueda que no se va a considerar dentro de la propuesta de optimización. A pesar de lo anterior, en esta sección voy a describir la interfaz que utiliza el CEMC y el procesamiento que se da en ella.

Según la definición del Diccionario del español de México[16], una interfaz es una parte de un programa que permite la comunicación entre el usuario y la aplicación. Según esta definición que considero bastante acertada, la interfaz de consulta del CEMC será la parte dentro del motor de búsqueda que nos va a permitir interactuar con todo el resto del sistema, esta parte es la página web. Esta página se encuentra alojada en un servidor Apache Tomcat, debido a que su desarrollo está hecho en Java.

El sitio cuenta con una sección de Inicio de sesión y otra de Registro. Estas secciones van a cumplir con dos funciones adicionales a la generación de concordancias, una de ellas es el permitir el acceso a los usuarios registrados a más secciones dentro del sitio y a más documentos. Otra de las funciones es para los administradores del sitio, a los que les va a permitir conocer la demanda del sitio, a los usuarios más frecuentes del mismo, y otras estadísticas. Estos registros, se van a almacenar dentro de la tabla correspondiente, como se mencionó previamente (Sección II.2.2.1).

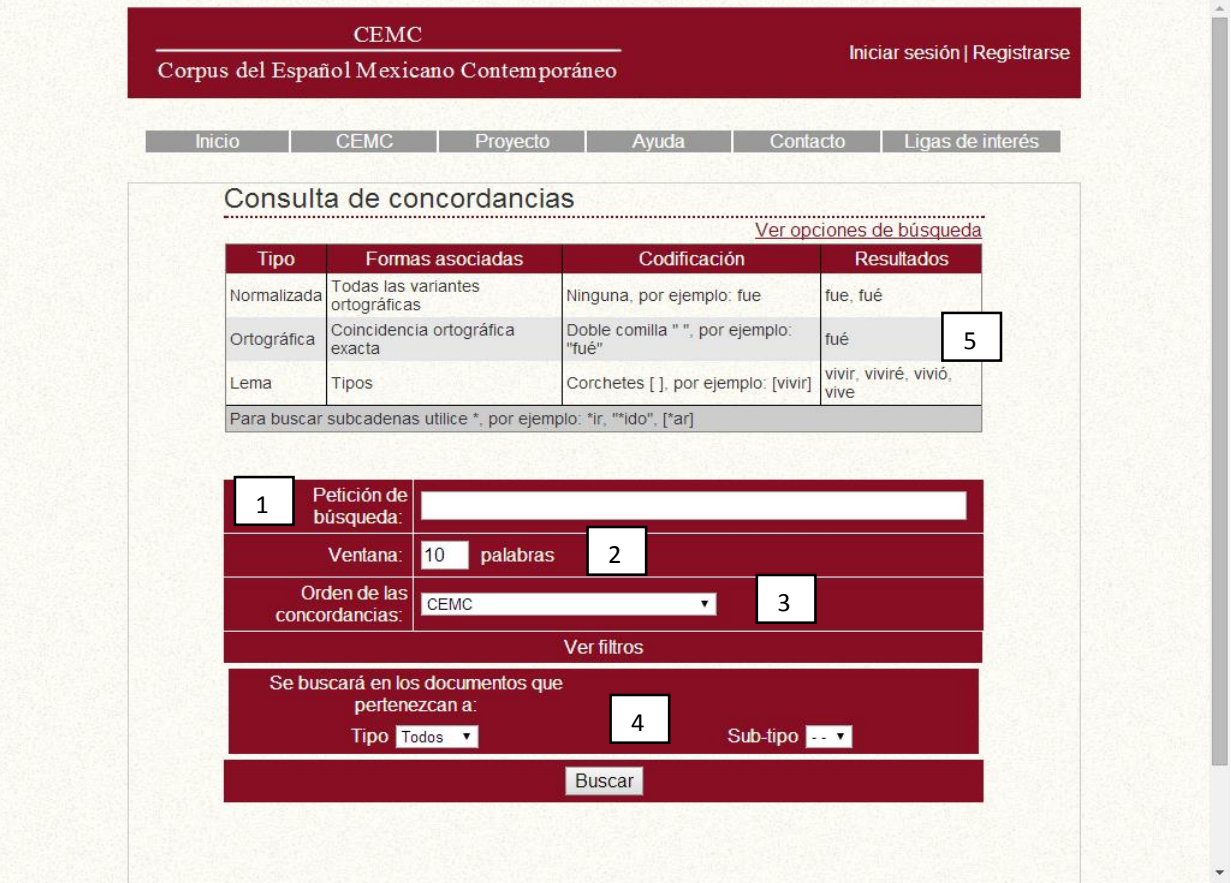
Además de estas secciones en donde hay una retroalimentación del usuario al sistema, el sitio cuenta con más secciones en las que se pueden encontrar un resumen del corpus, los participantes del corpus, ligas de interés, publicaciones en las que fue utilizado el corpus, las instituciones participantes, una sección de preguntas frecuentes y una más de contacto. En esta última como su nombre lo indica, el usuario puede ponerse en contacto con los administradores del sitio, para exponer dudas, sugerencias, etc.

[16] Diccionario del Español de México (DEM) http://dem.colmex.mx, El Colegio de México, A.C., [22/04/2014].

Entrando en el tema de la consulta al corpus, hay una página que va a tener un cuadro de texto para la petición de búsqueda. Dentro de este control, el usuario va a poder ingresar cualquiera de las opciones de consulta existentes. Estas son:

- Consulta por lema. En esta opción se hace una búsqueda del lema ingresado, entonces los resultados serán todas aquellas palabras que contengan el lema deseado. Los caracteres especiales que indican que se desea este tipo de consulta son los corchetes. Ejemplo: para el lema *[vivir]* los resultados serían *vivir, viviré, vivió, vive.*
- Consulta ortográfica. En esta opción se busca una coincidencia ortográfica exacta de la palabra ingresada. Para ingresar esta opción es necesario escribir la palabra deseada dentro de comillas dobles. Ejemplo: *"ahora"*.
- Consulta normalizada. La palabra deseada se ingresa sin caracteres especiales y los resultados que se devuelven son todos los que guarden la misma forma aunque varíen en la ortografía de la palabra. Ejemplo: *fue*, los resultados serían *fue, fué.*
- Consulta con comodines. En esta opción de búsqueda el sistema acepta dos comodines (? y *) para todos los tipos antes mencionados. En el caso del signo de interrogación, el sistema interpreta este símbolo como una o ninguna letra en esta posición, así una búsqueda como: *?ola* dará las concordancias de palabras como *bola, cola, hola, lola, mola, rola, sola y yola* entre otros. En el caso del asterisco, el sistema va a interpretar como que la cadena debe contener de cero a infinito caracteres en esa posición. Así la consulta anterior se escribiría como **ola* y algunos de los resultados serían las concordancias de *abrazándola, acarreándola, acompañándola, aflojándola, agrícola, hola, etc*.

Dentro del procesamiento de la página se evalúa la sintaxis de la consulta para que en caso de que no sea válida, la consulta no pase a un siguiente nivel de procesamiento en el que



se haría una conexión con la base de datos. El cuadro donde se ingresa la consulta se encuentra indicado con el número 1 en la Figura 13. La tabla que indica las opciones de búsqueda es el indicado con el número 5 dentro de la misma figura.

El siguiente control es el indicador del tamaño de la ventana de palabras para la concordancia, es el cuadro con el número dos dentro de la Figura 13. La lista que permite seleccionar el orden con el que van a aparecer las concordancias se encuentra indicada con el número

Figura 13. Interfaz de consulta del CEMC.

tres dentro de la misma figura. En el número 4 se encuentran los filtros de nivel, género y uso del lenguaje, los cuales permiten una búsqueda más específica dentro del corpus.

Como parte de la interfaz de consulta se encuentra la pantalla una vez que se han generado las concordancias. Esta pantalla muestra el XML generado durante la consulta en un formato con estilo y de fácil lectura para el usuario final. La información que se muestra dentro del documento con los resultados es el número del renglón, la concordancia y dentro de esta indicada en color rojo la palabra buscada o palabra base. En el extremo derecho del documento de resultados se encuentran los datos del texto a partir del cual fue recuperada la concordancia. Un ejemplo de esta pantalla se puede ver en la Figura 14 en la que se pueden observar los resultados de consultar la palabra *vez* con una ventana de tres palabras y ordenadas alfabéticamente por la palabra inmediata a la izquierda de la palabra base.

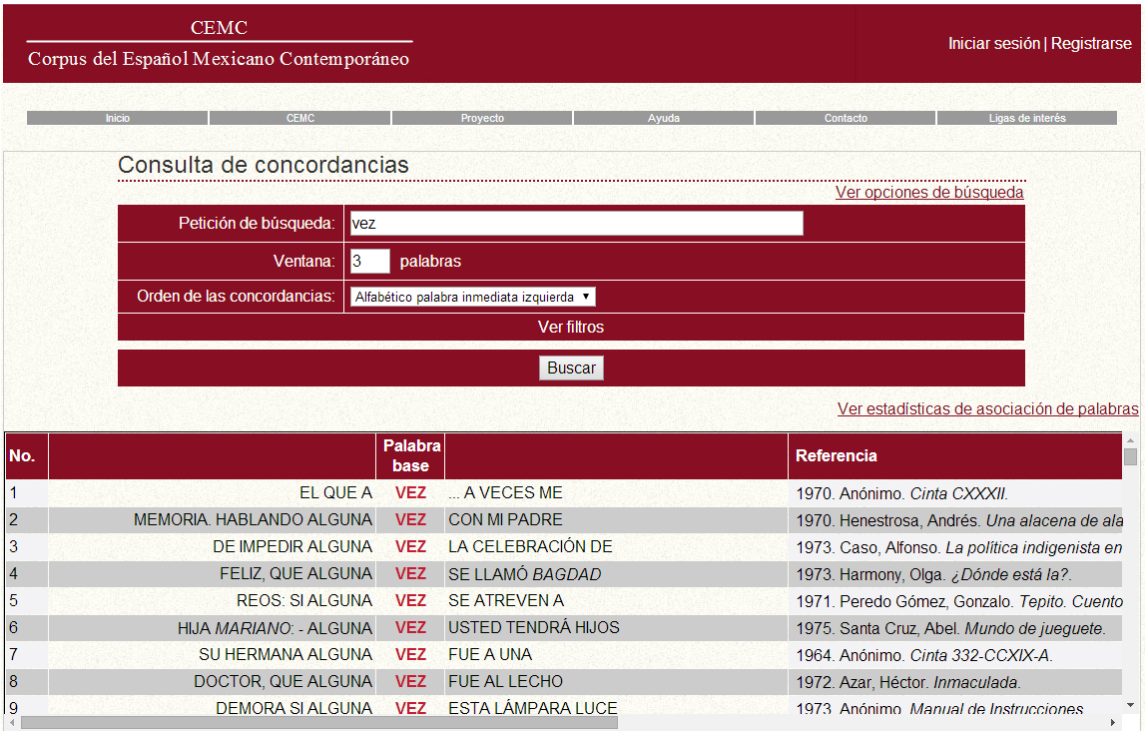


Figura 14. Ejemplo de los resultados del CEMC.

Durante este capítulo he buscado describir con una mayor profundidad los detalles de la arquitectura actual. A partir del siguiente capítulo, voy a mostrar y definir mi nueva propuesta

de indexador y generador de concordancias con la que busco cumplir con los objetivos planteados en un principio en esta tesis.

III. Propuesta de optimización

Una vez analizada la arquitectura actual, sus áreas de oportunidad y fortalezas, en este capítulo explicaré la arquitectura que propongo. Entraré en detalle en cada una de sus partes, desde el diseño hasta la instalación, continuando con el desarrollo. Es necesario hacer notar que la propuesta que desarrollé varía en gran medida con el sistema actual, desde el lenguaje de programación utilizado hasta el enfoque para la recuperación de los datos.

III.1. Análisis preliminar

Uno de mis requerimientos principales es una lista de concordancias obtenida a partir de textos etiquetados. Debido a que el modo en que se realizan las peticiones no es algo que deba modificar, comencé con el indexador. En éste punto surgieron varias opciones. Una opción era la de continuar con las bases de datos relaciones y el sistema tal cuál se encontraba en ese momento. Otra era la de usar un enfoque de recuperación de la información usando índices invertidos[17] y archivos para organizar la información. Otra era la de usar un software ya existente moldeándolo a mis necesidades. Por último también surgió la idea de manejar ontologías.

En la tesis de Vargas Mejía "*Optimización del generador de concordancias del Corpus Histórico del Español en México (CHEM) mediante una comparación entre manejadores de bases de datos relacionales y administración de desempeño de bases de datos*" se hace una

[17] Esta idea surgió a partir de una discusión de mi propuesta de investigación en el seminario interno del GIL.

comparación entre distintos manejadores de bases de datos. En esa investigación fueron comparados los manejadores de MySQL, ORACLE y PostgreSQL.

De los tres manejadores analizados, ORACLE resultó como el más rápido, arrojando, según esta tesis, resultados en un tiempo dos veces menor que el manejador actual PostgreSQL. Además se realizaron otro tipo de pruebas, como separar los índices a una nueva ubicación, agregar parámetros adicionales en el filtro de la consulta; sin embargo, los tiempos de respuesta aumentaron milisegundos para todos los manejadores.

Otra de las pruebas consistió en distribuir los datos en segmentos más pequeños. De ésta prueba se observó que los manejadores de MySQL y de ORACLE resultaron más rápidos. Por todo lo anterior, la opción obvia sería cambiar de manejador de bases de datos; sin embargo, los recursos económicos son una variable que no se ha considerado y ORACLE es un software licenciado, que involucraría un gasto que puede ser evitado. Por estas razones no seguí esta alternativa y continué buscando[18].

Otra de las opciones para desarrollar mi propuesta era la de utilizar software existente. Dentro de los sistemas existentes, encontré a Xaira, SARA y *The IMS Corpus Workbench* (CWB). Aunque el CWB es de código abierto[19] y los otros dos sistemas son usados en corpus de gran tamaño como el que se pretende tener en el grupo, ninguna de estas opciones se ajustó a los requerimientos del GIL.

[18] No se debe descartar por completo el uso de bases de datos para manejo de corpus, ya que existen corpus modernos y de gran tamaño que las utilizan como El Corpus del Español (http://www.corpusdelespanol.org/).

[19] Se denomina software de código abierto a los sistemas que ponen su código fuente a disposición del usuario, permitiendo así que lo altere a su beneficio sin ningún costo adicional.

SARA es el sistema que da vida al BNC, desde el indexado del corpus hasta la plataforma de consulta. Una de las características de SARA es que el indexador trabaja con textos etiquetados. Esto resultó muy atractivo ya que los documentos del GIL están etiquetados también; sin embargo, no coinciden los estándares de etiquetado entre los corpus. Otro dato de SARA que llamó mi atención es que utiliza archivos-índice para administrar la información del corpus. En cuanto a la velocidad de respuesta, usa tablas hash para resolver este tema.

Entonces, una pregunta que surgió en este punto fue la siguiente: ¿es más conveniente ajustar el sistema al corpus o el corpus al sistema? La respuesta a esta pregunta se encuentra al recordar que aunque esta propuesta está enfocada en el CEMC, la arquitectura está planeada para dar servicio a más corpus del GIL. Un cambio al etiquetado o estructura de los textos del CEMC, implicaría una modificación a los textos de los demás corpus. Esta situación me impidió tomar a SARA como solución a mi problema de investigación.

Mas no todo fueron negativas para este enfoque, ya que noté que los tres sistemas utilizan archivos índices para organizar la información contenida, en lugar de bases de datos relacionales. Esto me impulsó a investigar sobre este aspecto y decidir utilizar una API para indexar los documentos del CEMC.

III.1.1. Lucy

Lucy es una API para Perl que nace como un enfoque de Lucene (API desarrollado en y para JAVA). Lucy es una biblioteca que contiene un motor de búsqueda para realizar búsquedas en textos completos. Debido a que mi nueva propuesta utiliza prácticamente en todas sus áreas esta biblioteca, en la siguiente sección expondré algunas de las características de Lucy. Cabe mencionar que toda la información expuesta sobre la biblioteca Lucy fue extraída del sitio web de la misma (http://lucy.apache.org/).

III.1.1.1. Definición

Lucy es una biblioteca modular de un motor de búsqueda de alto rendimiento. Esta es la definición que se encuentra en la página del software (http://lucy.apache.org/docs/perl/Lucy.html). Pese a que la definición parece breve, no deja fuera ninguna de las características del software. Dentro de los módulos de este motor de búsqueda se encuentran: un analizador de textos, un indexador, una interfaz de consulta y un módulo para realizar esta consulta.

Dentro de las características del software se encuentran la de ser un indexador incremental. Entendido esto como la capacidad de agregar o eliminar documentos de un indexado previo. Otra característica es la de soportar los operadores lógicos *AND*, *OR* y *AND NOT*. Así también dentro de las características del software se encuentra la de realizar búsquedas con los signos de + o – antepuestos a la consulta. Por último, la posibilidad de personalizar el ordenamiento y de búsqueda por frases son características también de Lucy. Aunque muchas de estas opciones dependen directamente del indexado.

En un modo básico, Lucy cuenta con dos módulos centrales: el indexador (Lucy:Index:Indexer) y el buscador (Lucy:Search:IndexSearcher). El primero es el que va a realizar el indexado de los documentos, generando los índices invertidos que posteriormente y como función de la misma clase se guardarán en archivos-índice. El segundo módulo es el que va a leer estos archivos-índice para realizar sobre ellos las consultas. Ambos, tanto el indexador como el buscador cuentan con una serie de características que se pueden personalizar, algunas de estas son descritas a continuación.

III.1.1.2. Características (esquema)

La clase Indexer, va a necesitar de ciertas variables para poder ejecutarse correctamente. Estas variables son:

- El esquema. Es un diseño de cómo será el indexado. En él se indica cómo serán tratados los elementos del corpus entre otras cosas.
- El directorio para el índice. En esta variable se ingresa la ruta de un índice existente o de un folder dónde guardar el índice.
- La opción de crear el directorio para el índice. Esta es una variable booleana que de ser cierta y el directorio para índice no exista, el sistema intentará crearlo.
- La opción de dejar o no como incremental el indexado. Esta es otra variable booleana que en caso de ser cierta, el sistema procederá a descartar toda la información previa. Cabe mencionar que la información vieja permanecerá intacta y visible hasta que el método *commit*() se ejecute.
- Un administrador de indexadores. En el caso de que se busque realizar un indexado desde distintas máquinas, el administrador regulará el acceso de cada máquina al índice y la generación de los índices.

El esquema de indexado es el elemento personalizable dentro de las variables de entrada para el indexador. El esquema es una especificación creada por el usuario para un índice invertido. Un esquema es entonces, una especificación que indica cómo otras entidades deben interpretar los renglones de datos del índice y cómo interactuar con él. Para declarar el tratamiento de cada campo, se utiliza el método spec_*field*() que acepta dos variables. Una de estas variables es el nombre del campo y la otra el tipo del campo. El tipo del campo admite a su vez seis opciones más.

El tipo del campo es una especificación por campo sobre las propiedades y comportamientos del mismo. Aquí se puede elegir entre dejar el tipo predefinido de Lucy o generar uno o varios tipos no tokenizables por Lucy. Ambas opciones fueron utilizadas dentro de mi propuesta. El segundo tipo es utilizado para cadenas con una coincidencia exacta y sus parámetros son:

- Boost. Es un valor flotante para el momento de la clasificación de los resultados y la selección del mejor.
- Indexed. Es un valor booleano que indica el caso en el que el campo debe ser indexado.
- Stored. Es un valor booleano que indica el caso en el que el campo debe ser almacenado. En caso de que un elemento sea marcado como verdadero en esta opción y falso en la anterior, el elemento servirá como información adicional al índice.
- Sortable. Es un valor booleano que índice el caso en el que el campo debe de ser tomado en cuenta para el ordenamiento de los resultados.

Haciendo un resumen del esquema de indexado, el esquema va a indicar cómo será tratado el índice de manera global y el tipo del campo indica cómo será tratado cada campo por el esquema dentro del índice. Para aclarar este punto, muestro en la Figura 15 la declaración del esquema y de los tipos para cada campo del corpus.

```
my $esquema = Lucy::Plan::Schema->new;
my $polyanalyzer = Lucy::Analysis::PolyAnalyzer->new(
language => 'es',
);
      #Declaración de los tipos de campo>>>>>>>>>>>>
my $tipoIndiceToken = Lucy::Plan::StringType->new(
      boost => 1.0,
      indexed => 1,
      sortable => 1,
      stored => 1,
);
```

```
my $tipoIndiceLema = Lucy::Plan::StringType->new(
      boost => 0.75,
      indexed => 1,
      sortable => 1,
      stored => 1,
);
my $tipoIndicePOS = Lucy::Plan::FullTextType->new(
      boost => 0.5,
      analyzer => $polyanalyzer,
);
my $tipoIndiceFiltro = Lucy::Plan::FullTextType->new(
      boost => 0.25,
      analyzer => $polyanalyzer,
);
my $tipoOrden = Lucy::Plan::StringType->new(
      indexed => 0,
    sortable => 1,
    stored => 1,
);
my $tipoComplemento = Lucy::Plan::StringType->new(
      indexed => 0,
    stored => 1,
);
      #Declaración de los tipos de campo<<<<<<<<<<<
      #Declaración del esquema de indexado
$esquema->spec_field( name => 'lema', type => $tipoIndiceLema );
$esquema->spec_field( name => 'POS', type => $tipoIndicePOS );
$esquema->spec_field( name => 'token', type => $tipoIndiceToken );
      #De aqui en adelante se obtienen del encabezado:
      #Filtros>>>>>>>>>>
$esquema->spec_field( name => 'genero', type => $tipoIndiceFiltro );
$esquema->spec_field( name => 'nLengua', type => $tipoIndiceFiltro );
$esquema->spec_field( name => 'uso', type => $tipoIndiceFiltro );
      #<<<<<<<<<<Filtros
$esquema->spec_field( name => 'documento', type => $tipoComplemento );
$esquema->spec_field( name => 'hablante', type => $tipoComplemento );
$esquema->spec_field( name => 'permisos', type => $tipoComplemento );
$esquema->spec_field( name => 'referencia', type => $tipoComplemento );
$esquema->spec_field( name => 'titulo', type => $tipoOrden );
$esquema->spec_field( name => 'lugar', type => $tipoOrden );
$esquema->spec_field( name => 'fecha', type => $tipoOrden );
```

Figura 15. Bloque de declaración del esquema de indexado.

Dentro de la Figura 15 se encuentra en la primera línea la inicialización de la variable *esquema*. A continuación se encuentra la variable *polyanalyzer*, esta es un tipo de campo predefinido en Lucy y dentro de sus parámetros de entrada está el lenguaje, que en este caso es el español. Las siguientes declaraciones (*StringType*), son para especificar los tipos de campos requeridos dentro del esquema de indexado. Estos tipos son:

- Token (*$tipoIndiceToken*). Este tipo va a tener un valor de 1.0 en el *boost*, va a ser indexado, almacenado y usado como parámetro para el ordenamiento. Está enfocado a los tokens del texto.
- Lema (*$tipoIndiceLema*). Este tipo va a tener un valor de 0.75 en el *boost*, buscando darle una mayor prioridad al tipo token en caso de obtener respuestas en ambos campos. El campo con este tipo va a servir como índice, como información adicional y como parámetro para el ordenamiento. Está enfocado a la información del lema de cada token.
- Parte de la oración ($*tipoIndicePOS*). Este tipo va a tener un valor de 0.5 en el *boost*, dejando así en tercer lugar de prioridad a los valores en este campo. A diferencia de los tipos anteriores este es *polyanalyzer*, de modo que este tipo será analizado bajo las reglas de Lucy, que no son transparentes ni claras en muchos de los casos. Está enfocado a la información de la parte de la oración de cada token.
- Filtro ($*tipoIndiceFiltro*). Este tipo va a tener un valor de 0.25 en el *boost*, esto indica que los filtros serán los parámetros con el menor peso dentro de la selección de las posibles respuestas correctas. Igual que el tipo anterior este es indicado como *polyanalyzer*, buscando con esto ampliar la búsqueda cuando se utilicen los filtros (nivel de la lengua, uso y género literario) en la consulta a través de la interfaz web.
- Orden ($*tipoOrden*). Este tipo está pensado en aquellos valores que únicamente sirven para ordenar los resultados. Es por esto que el parámetro *indexed*

de este tipo se encuentra en 0 y los parámetros que se encuentran encendidos son los de almacenamiento y ordenamiento.

- Complemento ($*tipoComplemento*). Este tipo es para aquellos valores que sirven únicamente como información adicional, tal es el caso del nombre del documento, los permisos del documento y la referencia. Estos valores son considerados como complemento ya que no serán consultados de manera directa por el usuario desde la interfaz web.

Todos los tipos anteriores son ingresados junto con el nombre del campo al que pertenecen, generando así un esquema de indexado. En la Figura 15 se muestra esta asignación en cada línea donde se encuentra la variable *$esquema*. El esquema es una parte importante dentro de todo el motor de búsqueda ya que sin él, no podría haber un indexado y el buscador de Lucy tampoco tendría modo alguno de leer el archivo-índice generado.

Ya con el esquema declarado y el objeto *Indexer* inicializado, el método para agregar una nueva entrada al archivo índice es *add_doc*. Sus parámetros de entrada son cada uno de los campos declarados en el esquema, ahora con los valores correspondientes. Pero para que todos los cambios tengan efecto dentro del índice, el objeto *indexer* debe ejecutar el método *commit*. De no ejecutar este método cualquier modificación realizada al índice se perderá y éste quedará como en un principio del programa.

En el caso de la búsqueda, la declaración del objeto buscador de Lucy es más sencilla ya que solo requiere de conocer el directorio del archivo índice. A partir del cual va a leer el esquema para poder realizar las consultas deseadas. El procedimiento para realizar las consultas requiere del término a buscar, definido dentro de un objeto *TermQuery*. Este objeto va a almacenar el término deseado y el campo dentro del esquema al que pertenece. El siguiente

paso corresponde al objeto buscador de Lucy, el cuál va a realizar la consulta mediante el método hits y con el *TermQuery* como parámetro de entrada para entregar un conjunto de datos con los resultados obtenidos (Figura 16).

Si bien Lucy cuenta con muchas más características que no voy a detallar en esta tesis, las características aquí mostradas son las que fueron utilizadas dentro de la nueva propuesta. De este modo busco aclarar y facilitar la comprensión del diseño y desarrollo de la nueva propuesta. A continuación listaré algunas de las ventajas por las cuales elegí utilizar Lucy y una breve definición sobre índices invertidos.

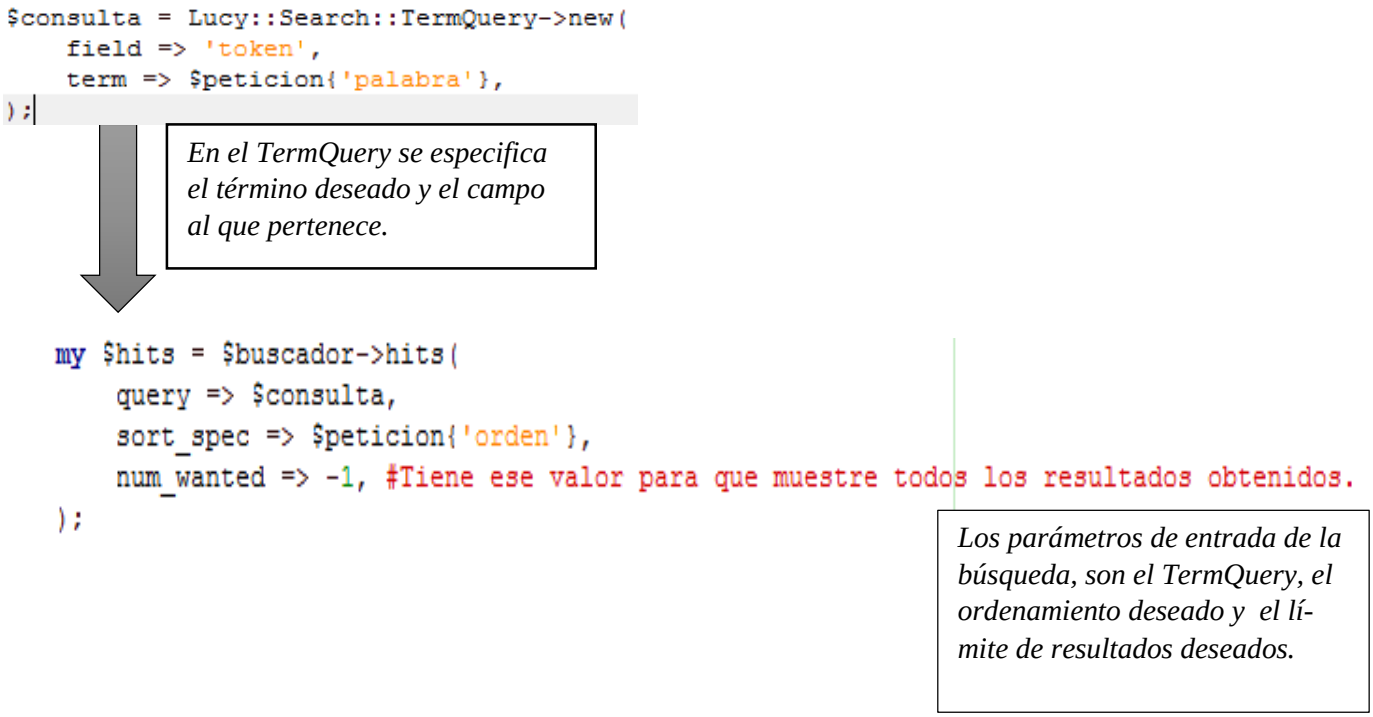


Figura 16. Proceso de búsqueda en Lucy.

III.1.1.3. Ventajas

Sin lugar a dudas dentro del proceso de diseño surgieron muchas opciones de solución. Sin embargo había que tomar una decisión y elegir entre todas aquella que mejor se acoplara a

mis necesidades de desarrollo. Aunque hubo distintas opciones elegí a Lucy por ciertas ventajas que presentó sobre las otras opciones. Una de ellas, y la principal sin duda, es la flexibilidad que tiene, y es que al ser de código abierto, la biblioteca se puede ajustar tanto como el desarrollador lo desee.

Otra de las ventajas es el lenguaje en el que está desarrollado, ya que el cambio en el lenguaje de programación fue una decisión que tomé desde un principio el que Lucy esté desarrollada en y para Perl representó una gran ventaja. Otro punto a favor de Lucy fue el formato de los archivos-índice (JSON), este formato es sumamente accesible y puede ser abierto, leído y consultado por algún software en JAVA en caso de ser necesario. Una última ventaja, y que no resulta menos importante, es la de que es una biblioteca diseñada específicamente para el procesamiento de corpus de gran tamaño.

III.1.1.4. Índices invertidos

Como lo mencioné previamente Lucy genera índices invertidos para estructurar la información indexada. Los índices invertidos o archivos invertidos (Monz y de Rijke, 2002, p. 9) son como lo menciona Kowalski (1997, p. 76) una estructura de datos utilizada en la administración de bases de datos y en los sistemas de recuperación de información. Esta estructura se forma básicamente por una colección de listas, una por término, en las que se almacenan los identificadores de los documentos que contienen el término (Zobel y Moffat, 2006, p.8).

La estructura de archivos de los índices invertidos está compuesta por tres tipos básicos de archivos: los documentos fuente, el diccionario y las listas de inversión o también llamadas listas de publicación (Kowalski, 1997). A cada documento dentro de la estructura se le asigna un identificador único, con el cuál se le va a hacer referencia dentro de las listas de inversión. De modo que cada lista de inversión tiene como dato llave el nombre de la

palabra y como información adicional todos los identificadores de los documentos en los que se encuentra esta palabra.

La manera de llegar a estas listas de inversión es a través del diccionario, en él se van a almacenar todas las palabras únicas y un apuntador a estas listas de inversión. Cuando una búsqueda es ejecutada, las listas de inversión de los términos en la consulta se localizan y el resultado final es una lista de documentos que satisfacen la consulta.

Lucy ejecuta este mismo procedimiento sobre sus archivos-índice y el resultado como ya lo he mencionado es un conjunto de datos con los documentos que contienen las palabras, es por esto que debo aplicar un procesamiento posterior (III.3.2) para la extracción y generación de las concordancias. En las siguientes secciones desarrollaré el diseño y desarrollo de esta nueva propuesta utilizando en gran medida los conceptos aquí expuestos.

III.2. Diseño

En todas las asignaturas donde se ha de programar, la primera instrucción es diseñar el algoritmo antes de comenzar a escribir líneas de código. En el diseño entonces, voy a considerar los algoritmos de cada una de las partes de éste motor de búsqueda y los componentes que cada una de ellas requieren.

El dominio del problema facilita de alguna manera el planteamiento de la solución. Pues aunque la propuesta debe de ser pensada para dar soporte a todos los corpus del GIL que sean necesarios y para estar en línea disponible a cualquier persona, el usuario final resulta ser un usuario especializado. De modo que las consultas que se harán al sistema serán siempre por usuarios que conocen sus necesidades de información, por lo que no será nece-

saria una transformación de la petición en lenguaje natural a un lenguaje de consulta especializado. Por ejemplo, si la petición es “la casa roja”, lo que se busca es la misma cadena, sin modificación, en los documentos.

Por su parte, el indexador va a manejar textos etiquetados, lo que de cierto modo también representa una ventaja. El etiquetado entonces, me permitió identificar ciertos patrones dentro de los textos que utilicé para la extracción de los tokens y de la información lingüística de estos, lema y parte de la oración. Haciendo uso de las expresiones regulares y de estos patrones en los textos fue que surgió mi propuesta de solución al indexado del corpus.

En el siguiente diagrama de flujo se muestra el algoritmo que sigue el indexador desarrollado y utilizado por esta propuesta. Cabe recalcar que en el siguiente diagrama no se describen ni se señalan funciones en específico del programa, pero sí el funcionamiento en conjunto de estas (Figura 17).En esta primera parte del diagrama de flujo se encuentran la declaración del esquema de indexado, la generación de una nueva traza de ejecución y la apertura de los documentos del corpus.

Entonces, en los primeros procesos se define y crea el esquema de indexado, además de crearse también el directorio en el que se van a guardar los archivos-índice de los bigramas. El siguiente proceso declara un nuevo hilo, logrando tener en este punto dos procesos con las mismas líneas de código.

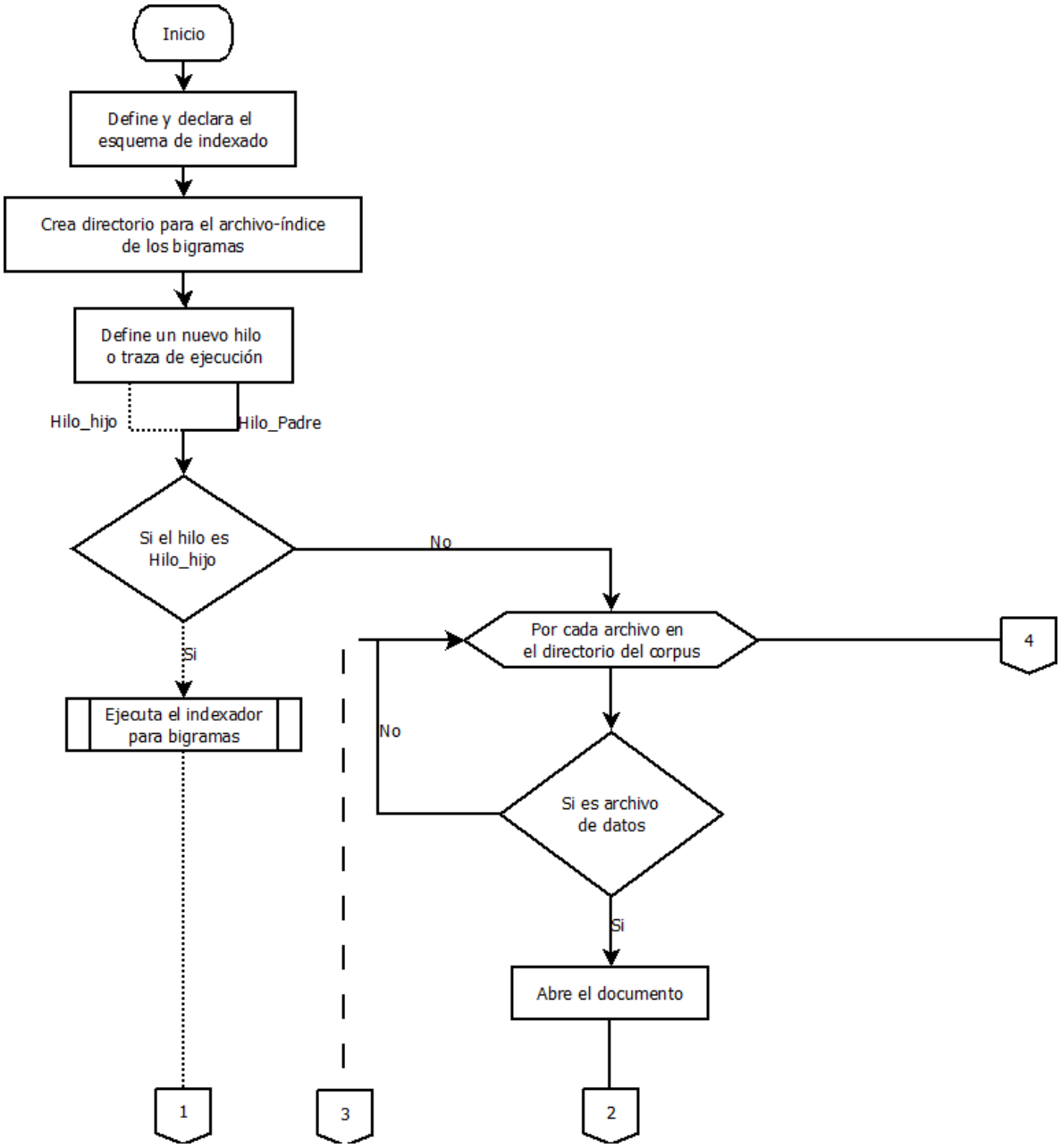


Figura 17. Diagrama de flujo del indexador. Parte I

En la Figura 17 se ven dos líneas que parten del mismo proceso. Con esto busco mostrar el camino que recorren la traza creadora (Hilo_Padre) y la traza creada (Hilo_hijo). El camino de ambas trazas no va a distar mucho en principio, ya que inmediatamente que se definen los hilos, el programa identifica a cada uno de ellos y asigna tareas distintas para

ambos. A partir de este punto el hilo creador va a realizar el indexado por unigramas y el creado realizará el indexado de bigramas.

Ya que el algoritmo para indexar los bigramas es básicamente el mismo que para los unigramas, continuaré con la traza del hilo creador. El siguiente paso es abrir cada archivo de datos dentro del directorio de corpus e indexarlo. El primer paso se realiza dos veces, en la primera lectura se va a obtener toda la información contenida en el encabezado del archivo. Esta obtención se va a realizar mediante comparaciones (Figura **18** y Figura **19**). Una vez que se lee la última línea del encabezado, marcada por la etiqueta </encabezado> el programa detiene la lectura y continúa con el indexado de los tokens.

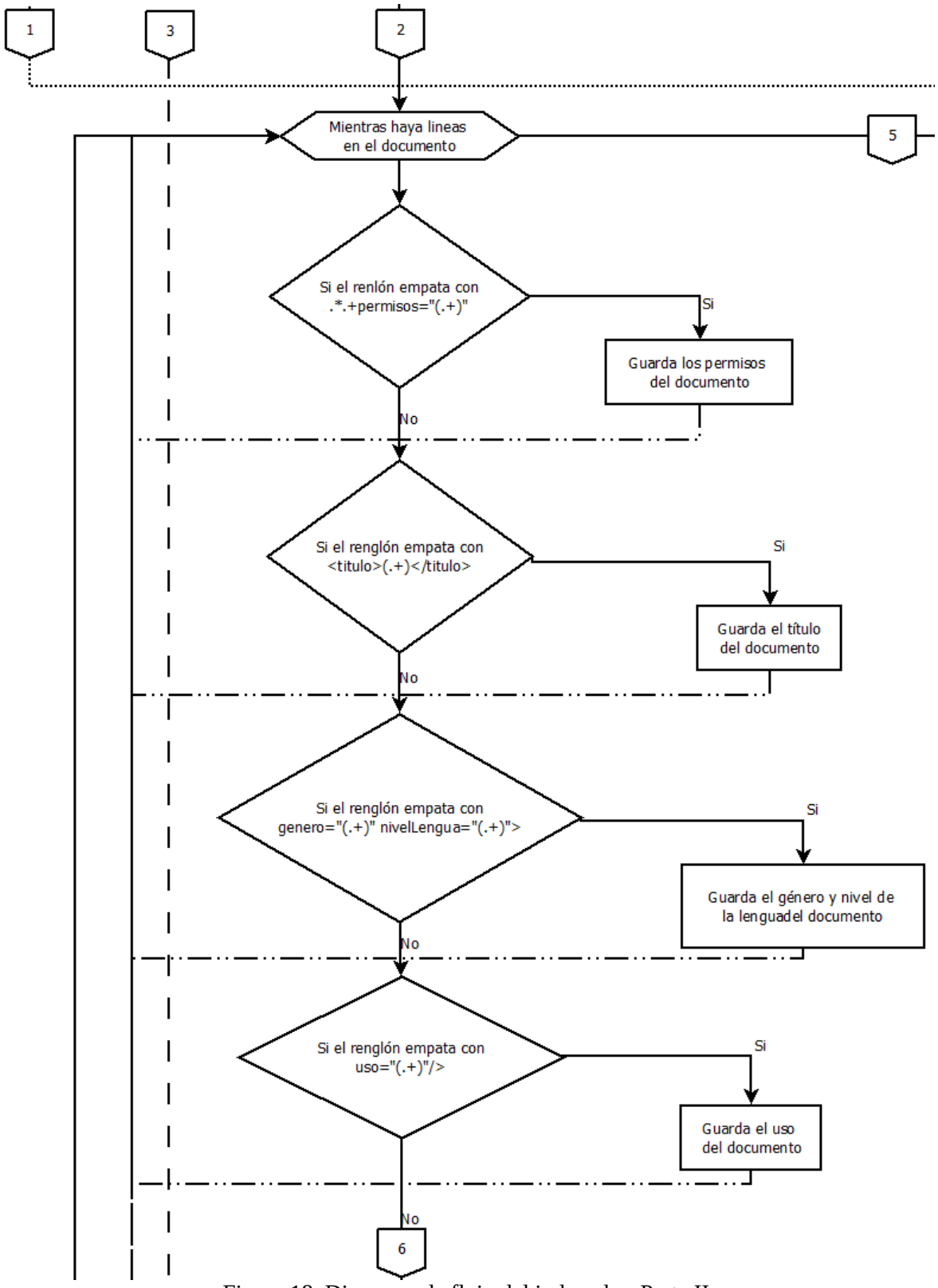


Figura 18. Diagrama de flujo del indexador. Parte II

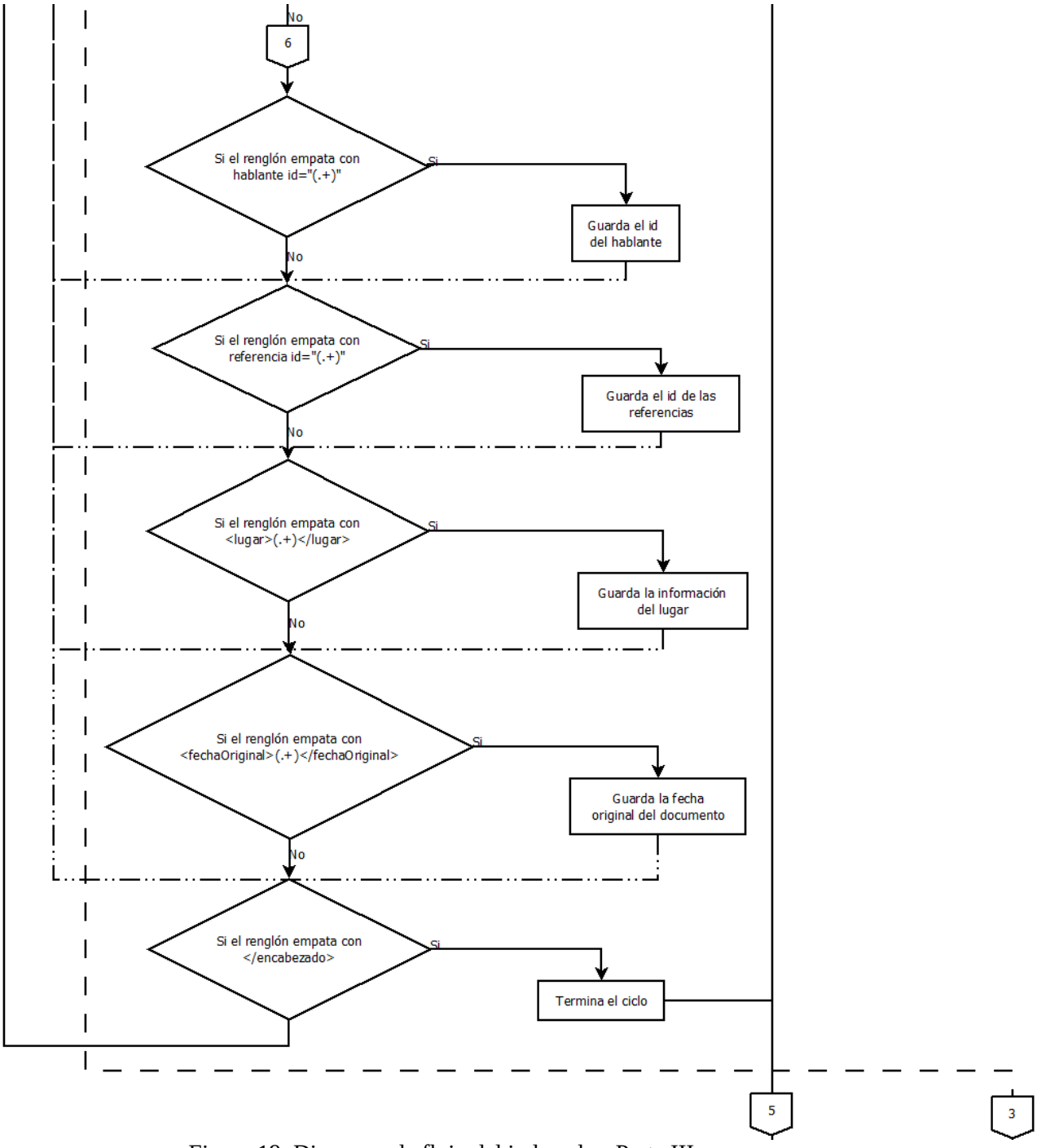


Figura 19. Diagrama de flujo del indexador. Parte III

Una vez que se obtiene la información del encabezado del archivo, se almacena en una tabla hash que va a servir como plantilla para guardar temporalmente los valores a indexar. De este modo los valores dentro de la plantilla serán la información bibliográfica del

texto y la información lingüística del token. La información lingüística y el token serán las partes de la plantilla que van a estar en constante cambio, como lo muestra la Figura 20.

En esta parte del diagrama de flujo (Figura 20) se observa una sola instrucción de decisión ejecutada para cada renglón del archivo. Dentro de esta decisión se separan los valores de token, parte de la oración y lema hallados en el renglón, para su posterior indexado. Ya que los textos se encuentran etiquetados en XML y con una estructura bastante clara, la expresión regular encontrada resulta bastante efectiva para realizar la tarea de separar la información.

Una vez obtenida la información, el paso siguiente es el de agregar un nuevo archivo índice al directorio. Esto se realiza mediante la instrucción correspondiente de Lucy, explicada más adelante en esta tesis. Este procedimiento se realiza para cada archivo dentro del directorio del corpus. Una vez que el sistema termina de indexar los unigramas espera al hilo que se encuentra indexando a los bigramas. Al terminar los bigramas, el sistema ha terminado por completo el proceso de indexado (Figura 21).

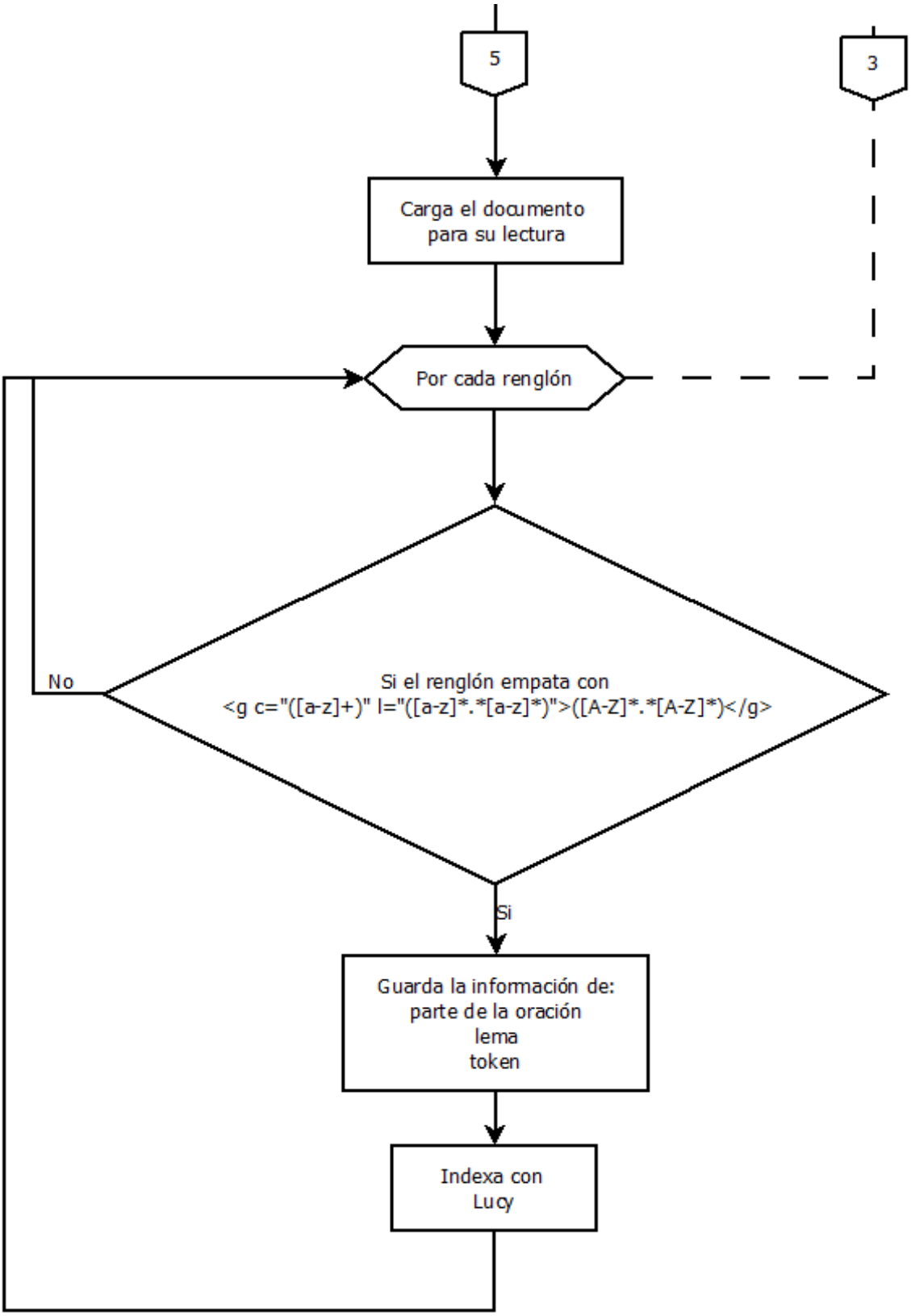


Figura 20. Diagrama de flujo del indexador. Parte IV

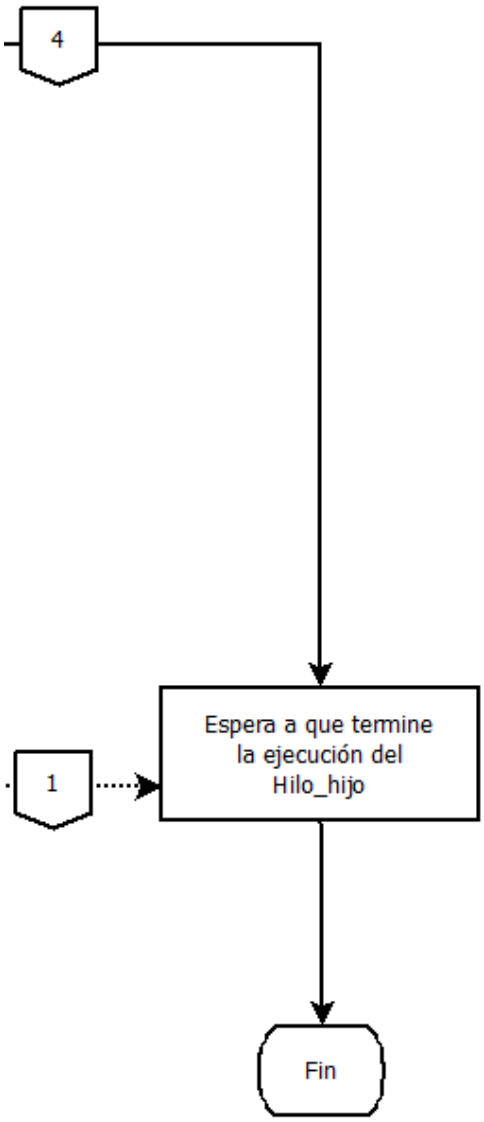


Figura 21. Diagrama de flujo del indexador. Parte V

Por otra parte, el buscador lo voy a dividir en dos diagramas, cada uno fragmentado en varias figuras para mejorar su explicación. En el primer diagrama detallo una línea central en la que se lee la petición, se interpretan las necesidades, se hace la consulta mediante Lucy y se inicia el segundo diagrama. En este segundo diagrama el proceso que se ejecuta es el de

la generación y recuperación de las concordancias a partir de los archivos donde se encuentren resultados, y la impresión de éstos en los documentos XML de salida.

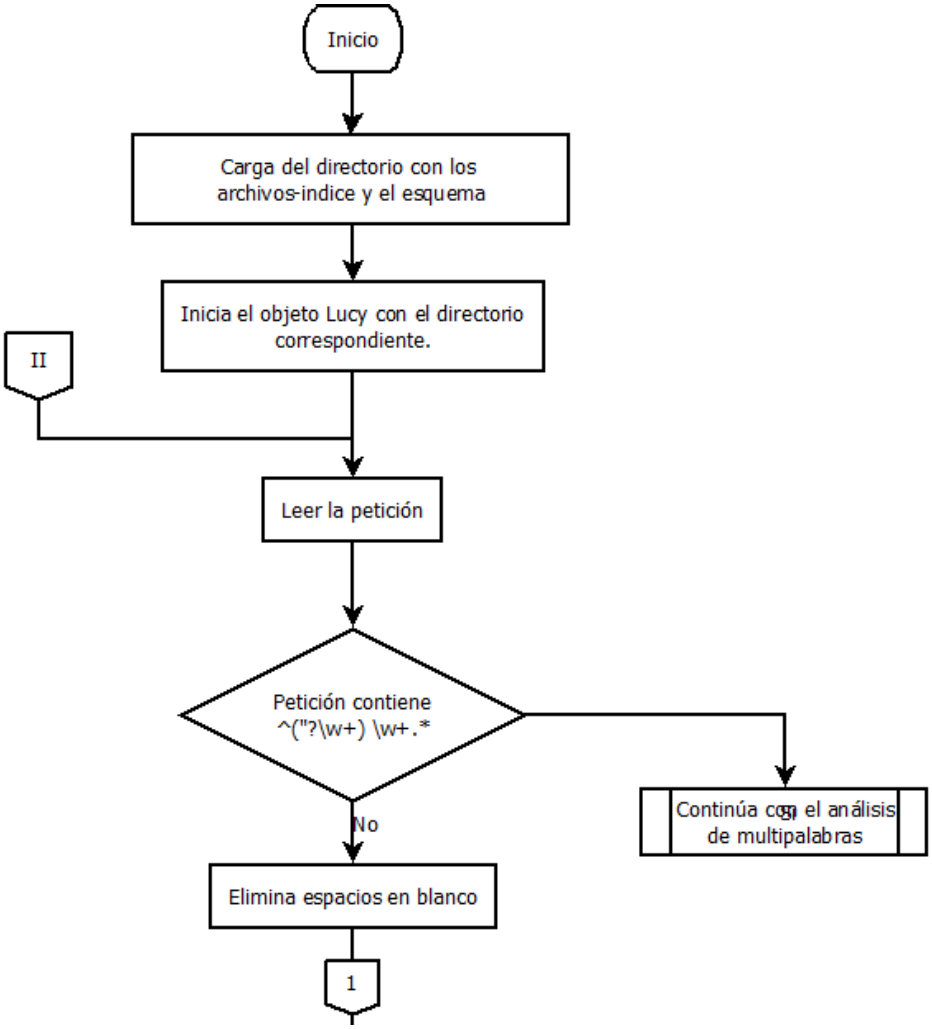


Figura 22. Diagrama de flujo de la búsqueda. Parte I

En la Figura 22 se encuentra la creación del objeto Lucy, que va a permitir realizar la consulta sobre los archivos-índice generados previamente en el indexado. En los primeros procesos se encuentra la carga del directorio donde se ubican: los archivos índice y el esquema de indexado; dentro del objeto Lucy para buscar. Así también se observa un conector de página con el número II, puesto allí porque en caso de que el sistema requiera hacer otra consulta en ese momento, el sistema regresará al punto de lectura de la petición. La lectura

de la petición incluye la obtención de los filtros, orden deseado y revisión de una petición válida.

Ya que se obtiene una correcta petición (esta sería sin números, con los metacaracteres y formatos aceptados), es entonces que el sistema va a determinar si se trata de una consulta unipalabra o multipalabra. Conocer esto es importante pues los esquemas y directorios son distintos para búsquedas unipalabra y multipalabra, como lo definí en el algoritmo de indexado. La búsqueda multipalabras aún no se encuentra terminada, por lo que continuaré con la búsqueda unipalabra.

Continuando entonces con la búsqueda unipalabra, es que el programa identifica el campo o característica del índice en donde va a buscar. Por ejemplo: si se trata de una búsqueda por la cadena exacta, el sistema va a buscar en el campo de tokens. Si es una búsqueda por lema, el campo será el lema. Así respectivamente según sea el caso. En la Figura 23 se muestran las decisiones para las búsquedas por coincidencia exacta, por lema y por token con comodines.

El proceso mostrado en la Figura 24 es para una búsqueda sobre las partes de la oración. Cabe mencionar que ésta sería una nueva opción de búsqueda ya que aunque el texto etiquetado es el mismo, el sistema actual aún no tiene implementada esta alternativa. En este caso, el proceso es un poco más elaborado ya que en el caso de que la estructura sea *<POST>token* la búsqueda se hace con dos objetos de búsqueda Lucy en paralelo, buscando en el campo *POST* y en el campo *token*. En caso de que la estructura sólo sea *<POST>*, el sistema también debe identificarla y realizar la búsqueda sólo por este campo.

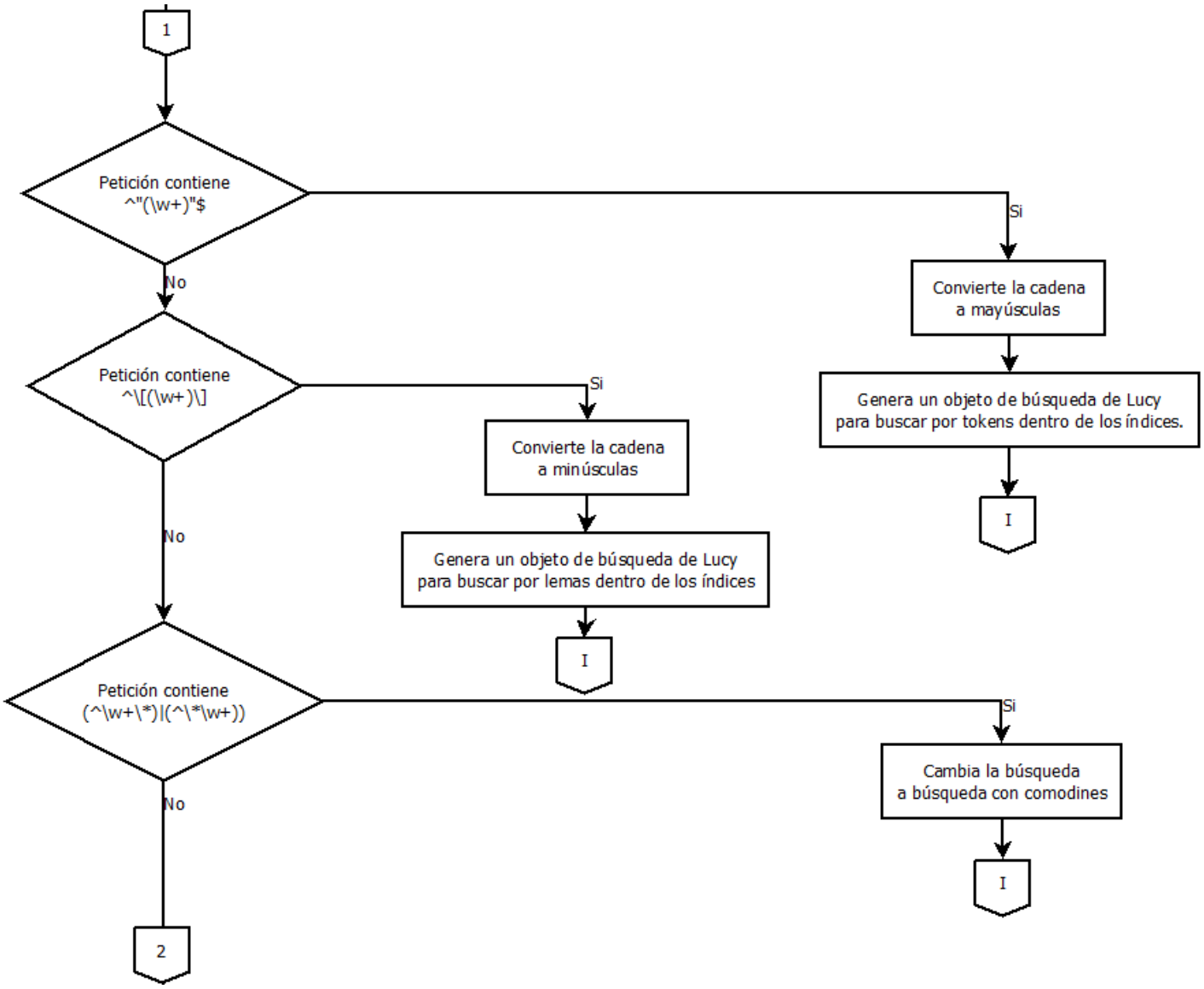


Figura 23. Diagrama de flujo de la búsqueda. Parte II

También dentro de la Figura 24 se encuentra la decisión para realizar una búsqueda por lemas con comodines y una última opción que sería la de buscar en el campo de tokens. Para asegurar que esta última opción sea ejecutada únicamente en caso de que la petición no encuadre con ninguna de las anteriores, es que se incorpora el conector con el valor I. Este conector se encuentra en las Figura 23 y Figura 24, al final de cada proceso de selección y va a indicar un brinco hasta el final del bloque de selecciones.

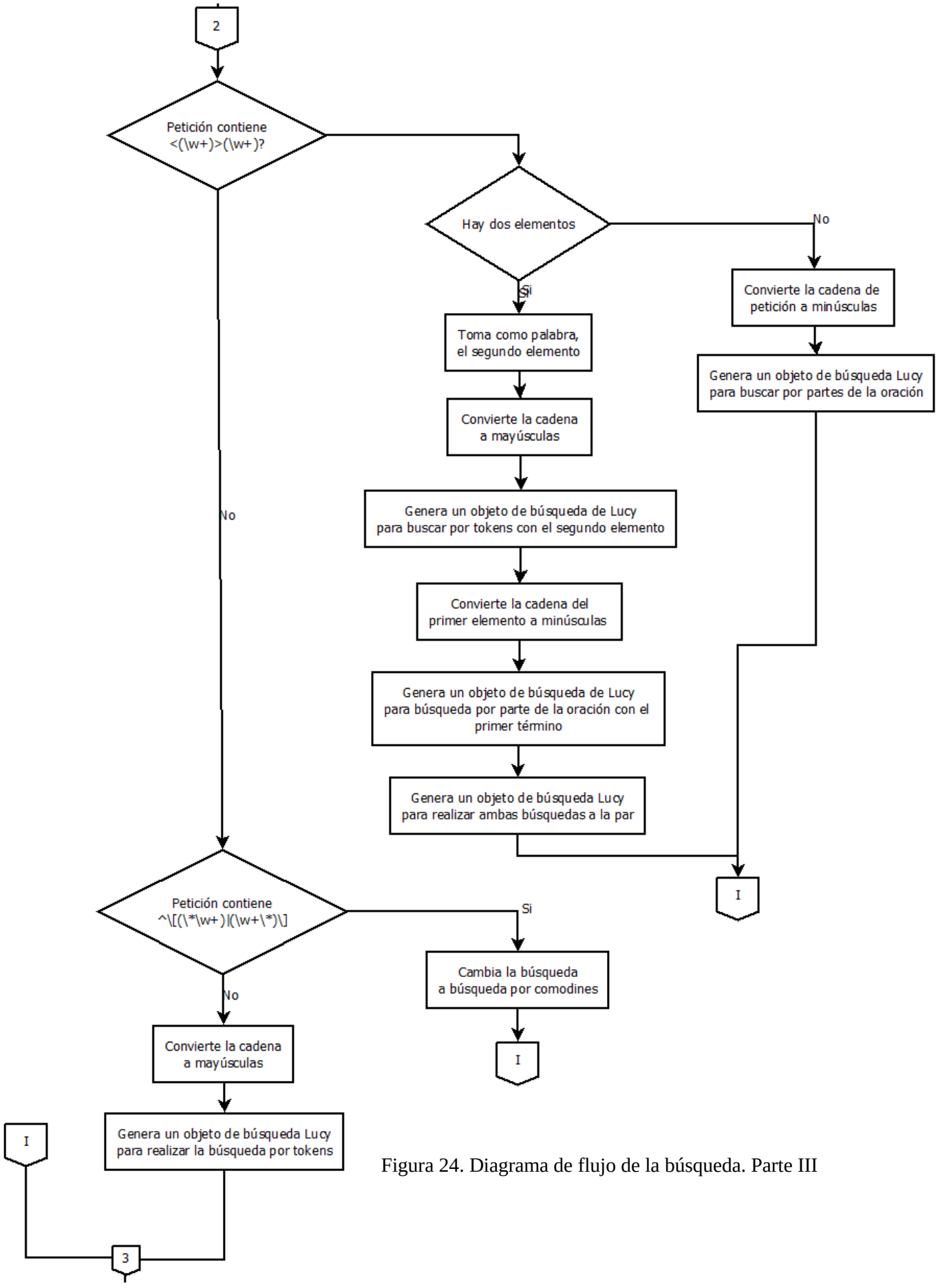


Figura 24. Diagrama de flujo de la búsqueda. Parte III

Con todos los parámetros definidos, directorio, esquema, petición, orden, filtros, tipo de búsqueda y campo de búsqueda, el siguiente paso es la ejecución de la búsqueda mediante el método de Lucy correspondiente (Figura 25). Antes de continuar, he de recordar que los

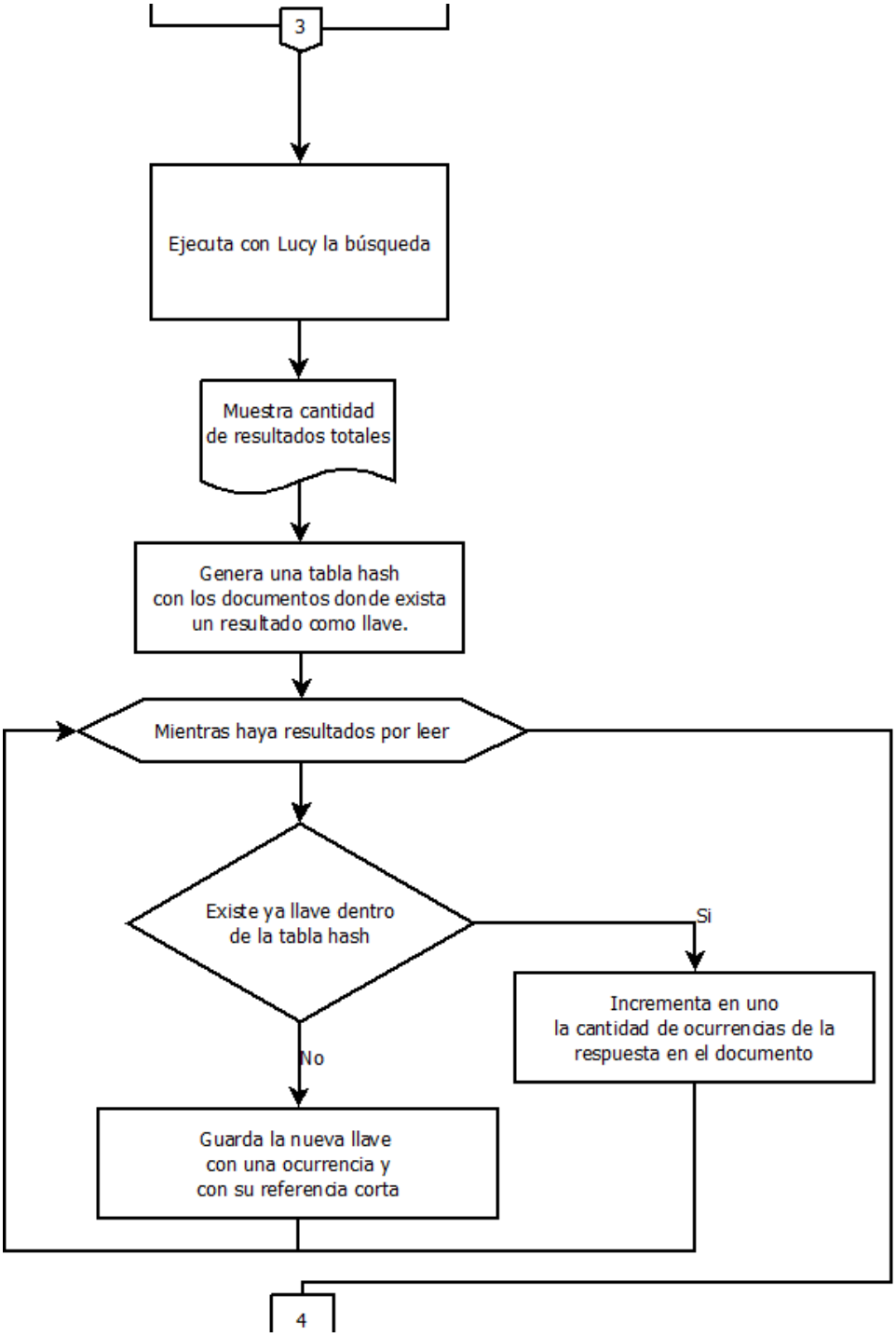


Figura 25. Diagrama de flujo de la búsqueda. Parte IV

resultados arrojados por Lucy son una lista de palabras coincidentes con la búsqueda y sus características. Esto lo menciono porque el ciclo que se muestra en la Figura 25 tiene como objetivo el de concentrar sólo los nombres de los archivos en donde se encuentran los resultados obtenidos y no las concordancias deseadas. Las concordancias son obtenidas en un método distinto.

Una vez que se obtiene la lista con los nombres de los archivos y la cantidad de ocurrencias o resultados dentro de ellos, en el siguiente ciclo, mostrado en la Figura 26, es dónde se van a imprimir las concordancias encontradas en el documento XML correspondiente. Antes de comenzar, se establece un límite de concordancias por archivo XML, este paso es con miras a un paginado en la muestra de los resultados vía web.

El ciclo de la generación comienza con un ciclo por cada documento que contiene una o más concordancias. Lo que se va a realizar es comprobar si es el primer documento en escribir dentro del XML de salida[20] en turno. En caso de que así sea, el programa deberá de tomar una acción extra a la obtención e impresión de las concordancias y es la de imprimir el encabezado del XML. Esto con el fin de que el XML pueda ser correctamente interpretado por el explorador web.

Si el documento no es el primero, entonces sólo se van a sumar la cantidad de ocurrencias dentro de él más la cantidad de ocurrencias de los documentos anteriores, en caso de que existan. Esta suma se hace para llevar un control y poder comparar este número con el límite previamente establecido. Si el límite es superior o igual a la suma anterior, entonces el

[20] Llamé XML de salida al archivo XML que se va a construir con las concordancias y que será leído por el sitio web para su presentación al usuario.

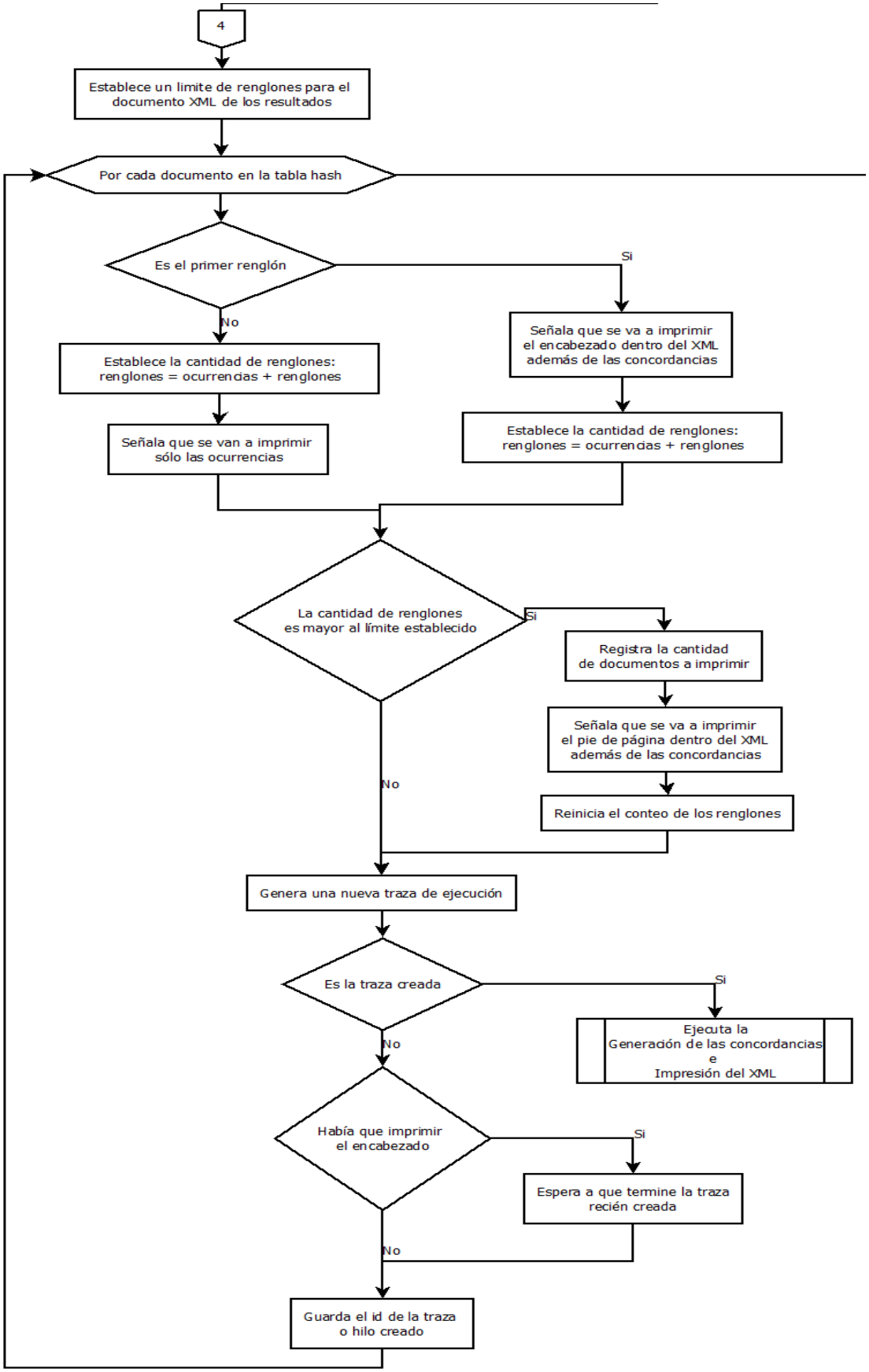
conteo se reinicia para el siguiente documento y el documento actual va a imprimir al final de sus concordancias el pie de página del XML de salida.

Con las instrucciones anteriores establecidas para el documento en turno, una nueva traza de ejecución es creada. Esta traza va a ejecutar la generación de las concordancias e impresión de la parte del XML correspondiente para el documento en turno. El siguiente paso para el hilo creador es el de esperar al hilo creado, en caso de que este último tenga que escribir el encabezado para el XML. En caso contrario, el programa sólo guardara el id del hilo creado y continuará con el ciclo.

Dentro del ciclo mostrado en la Figura 26 y detallado en los párrafos anteriores, es que se van a generar tantos hilos como documentos tengan resultados (concordancias). Lo anterior da como resultado una disminución considerable en el tiempo de escritura y de recuperación de las concordancias. Cada XML de salida será escrito por un grupo de hilos, la cantidad de hilos en el grupo estará dada por la cantidad de resultados dentro de los documentos y el límite de resultados a escribir en un XML.

Para terminar todo el proceso, la traza creadora o hilo padre se va a encargar de esperar a que todos los hilos creados terminen y mueran para evitar dejar procesos inconclusos dentro del servidor. Este paso es mostrado en la Figura 27. Una vez que todos los hilos creados han terminado de escribir, el hilo creador imprime en todos los archivos creados una etiqueta de cierre. Esto con el fin de que el XML sea mostrado correctamente por el servidor web.

Figura 26. Diagrama de flujo de la búsqueda. Parte V



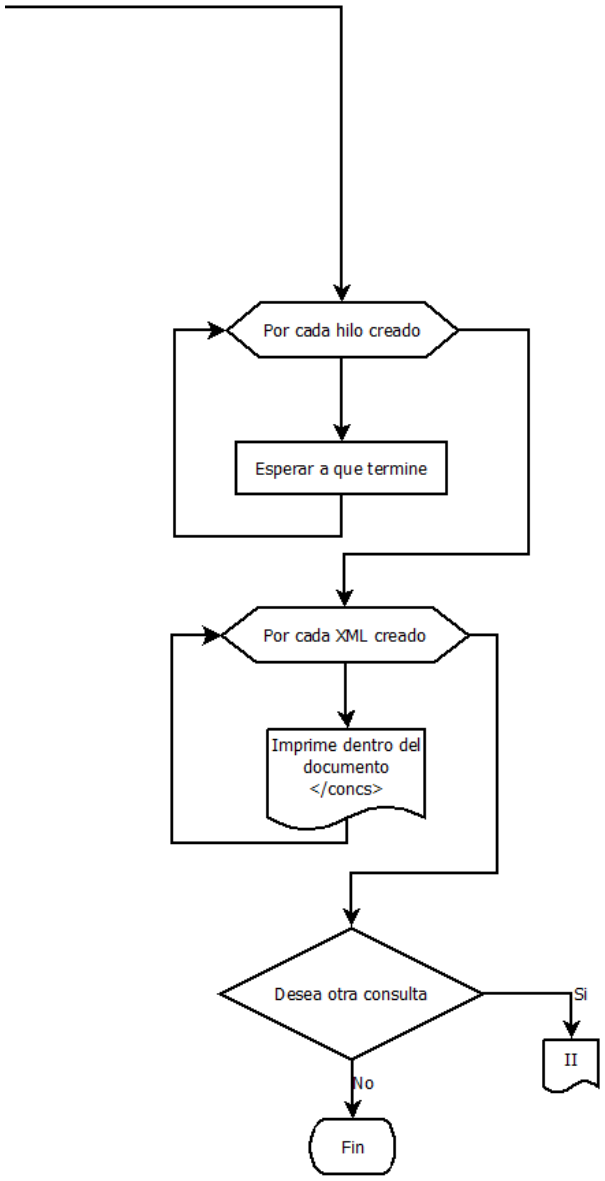


Figura 27. Diagrama de flujo de la búsqueda. Parte VI

Como se puede observar dentro de los diagramas mostrados, el proceso de la generación de las concordancias no se encuentra detallado en ninguno de ellos. Esto es porque lo haré con mayor detenimiento en la sección posterior con el nombre de este proceso (véase sección III.3.2).

III.3. Desarrollo

En esta sección ahondaré en las partes del motor de búsqueda. En una primera sección se verá el indexador, desde el almacenamiento y ordenamiento de las entradas hasta el análisis y obtención de la información de los documentos del corpus. En una segunda parte se verá la generación de concordancias, desde la obtención de la palabra buscada hasta la generación de los documentos XML de respuesta.

III.3.1. El indexador

El indexador fue lo primero que comencé a desarrollar. En este punto es que debí considerar alguna estrategia para explotar de mejor manera el hardware existente. El indexado es uno de los procesos con más trabajo para el procesador. Por lo tanto, pensé que implementar el uso de hilos o trazas de ejecución para un proceso tan cíclico como lo era el indexado me ayudaría a mejorar el tiempo total de ejecución.

Un punto que consideré fue la ejecución e interacción con el software a partir de una terminal de un sistema Linux, ya que tiempo atrás se me había pedido implementar esta posibilidad para el sistema de indexado actual. Entonces el indexado lo desarrollé por etapas, esto me dio un mayor control sobre los errores que pudieran surgir. En una primera etapa abordé el uso de Lucy, en una segunda el manejo de los archivos, en una tercera el manejo de los hilos y la última etapa consistió en la interacción con el usuario.

Una vez que comprendí las características de Lucy, al menos lo suficiente para usarlo, tenía que usar el software para conocer su comportamiento contrastándolo con lo que necesitaba. Gracias a esto encontré que debía personalizar las opciones que habría de utilizar de este software, pues no todas me eran útiles. Por ejemplo, ***polyanalyzer*** es una opción dentro

de los esquemas con la que Lucy organiza y administra la información de tal forma que permite obtener resultados muy amplios, pero que en mis pruebas resultaron inesperados.

En un principio utilicé la opción de ***polyanalyzer*** como una manera de facilitar mi tarea y de realizarla con múltiples trazas de ejecución. Sin embargo, esta opción lo que permite es que una misma palabra sea utilizada en múltiples campos y con múltiples funciones, de modo que cuando yo buscaba *maestra*, podía obtener resultados como *maestra*, *maestro*, *maestría* y *maestrita*. Este tipo de resultados no se veían erráticos o tan problemáticos como cuando hacía consultas para obtener palabras como *los*, aquí los resultados eran tan amplios que no logré identificar la relación entre mi consulta y los resultados.

Viendo que la opción ***polyanalyzer*** me daría más problemas que ventajas, opté por personalizar los esquemas de cada uno de los tipos de palabras que se encuentran en el corpus. Con esto logré obtener búsquedas más precisas y limitadas, además de que la opción de búsqueda exacta ya estaba definida por defecto. Por esto, en la primer parte del código dedico tantas líneas únicamente para declarar e inicializar el indexador (

Figura 28Figura 28).

```
my $esquema = Lucy::Plan::Schema->new;
my $polyanalyzer = Lucy::Analysis::PolyAnalyzer->new(
language => 'es',
);
#Declaración de los tipos de campo>>>>>>>>>>>>
my $tipoIndiceToken = Lucy::Plan::StringType->new(
      boost => 1.0,
      indexed => 1,
      sortable => 1,
      stored => 1,
);
my $tipoIndiceLema = Lucy::Plan::StringType->new(
      boost => 0.75,
      indexed => 1,
      sortable => 1,
      stored => 1,
);
```

```
my $tipoIndicePOS = Lucy::Plan::FullTextType->new(
      boost => 0.5,
      analyzer => $polyanalyzer,
);
my $tipoIndiceFiltro = Lucy::Plan::FullTextType->new(
      boost => 0.25,
      analyzer => $polyanalyzer,
);
my $tipoOrden = Lucy::Plan::StringType->new(
      indexed => 0,
    sortable => 1,
    stored => 1,
);
my $tipoComplemento = Lucy::Plan::StringType->new(
      indexed => 0,
    stored => 1,
);
#Declaración de los tipos de campo<<<<<<<<<<<<
#Declaración del esquema de indexado
$esquema->spec_field( name => 'lema',   type => $tipoIndiceLema );
$esquema->spec_field( name => 'POS', type => $tipoIndicePOS );
$esquema->spec_field( name => 'token', type => $tipoIndiceToken );
#Filtros>>>>>>>>>>>
$esquema->spec_field( name => 'genero', type => $tipoIndiceFiltro );
$esquema->spec_field( name => 'nLengua', type => $tipoIndiceFiltro );
$esquema->spec_field( name => 'uso', type => $tipoIndiceFiltro );
#<<<<<<<<<<Filtros
$esquema->spec_field( name => 'documento', type => $tipoComplemento );
$esquema->spec_field( name => 'hablante', type => $tipoComplemento );
$esquema->spec_field( name => 'permisos', type => $tipoComplemento );
$esquema->spec_field( name => 'referencia', type => $tipoComplemento );
$esquema->spec_field( name => 'titulo', type => $tipoOrden );
$esquema->spec_field( name => 'lugar', type => $tipoOrden );
$esquema->spec_field( name => 'fecha', type => $tipoOrden );
my $index = Lucy::Index::Indexer->new(
      index => $dirDelIndex,
      schema => $esquema,);
```

Figura 28. Bloque de declaraciones del indexador

En la primera etapa también observé como es que Lucy recupera la información. Con esto me di cuenta de que aún con todas las facilidades que el software ofrece, la obtención de las concordancias las tendría que realizar trabajando los archivos por mi parte. A través del uso frecuente del software noté que puede tokenizar de manera automática los textos en lenguaje natural, incluso los textos en español. La pregunta entonces fue si convendría cambiar los textos etiquetados del corpus por textos no etiquetados.

Lucy puede analizar los textos en lenguaje natural y realizar el tokenizado de manera automática basada en sus propias reglas. Esto podría haber ayudado a obtener las concordancias a partir de los archivos índices y no de los archivos del corpus. Sin embargo, tenía que considerar que el desarrollo buscaba impactar en más de un corpus del GIL, por lo que pensar en quitar el etiquetado de un corpus era pensar en quitar el etiquetado y la estructura de creación ya definida para los demás corpus del GIL que quisieran utilizar la arquitectura propuesta.

La opción que ofrecía Lucy para manejar los textos de una manera más automática y compacta resultaba poco eficiente para el caso específico de los corpus del GIL. Pues implicaba un mayor trabajo sobre los textos y modificar la estructura de creación y el conocimiento ya adquirido sobre esto para el grupo. Además de que las reglas con las que cuenta Lucy para tokenizar no son las que el grupo requería, de modo que, opté por usar a Lucy únicamente para indexar las entradas, y la tokenización y todo el análisis de los documentos XML los haría por mi cuenta.

Para analizar los documentos en XML probé con dos opciones. La primera fue usar la biblioteca para análisis de XML que ofrece perl. Sin embargo, esta opción requería de una gran capacidad de memoria RAM y además de que el tiempo de procesamiento era mayor que el de la segunda opción. El uso de expresiones regulares fue la segunda opción para realizar el análisis de los textos, resultó más rápida y con menos errores en el momento de la ejecución que su competencia. Mientras que con 100 documentos el análisis con expresiones regulares terminaba en uno o dos segundos, el otro terminaba en 2 o 3 minutos aproximadamente, dependiendo claro las otras operaciones del procesador.

El porqué del retraso en el tiempo lo atribuyo a la estructura del etiquetado, pues con el uso de la biblioteca, perl va a representar el XML en arreglos anidados. El acceso a cada una de estas estructuras anidadas hasta conseguir el valor deseado resulta en un algoritmo lleno de ciclos, en donde algunos resultarán más largo que otros. Con esto, el tiempo de ejecución aumenta de manera exponencial.

La longitud de una estructura va a variar según la cantidad de etiquetas que encierren a la palabra. Los arreglos contienen en su último punto la información a indexar. Por ejemplo, de la palabra *pude*, que sería la información en la última parte de la estructura, se van a guardar como valores del arreglo el lema *poder*, y la parte de la oración *verbo*. Todos estos dentro de una estructura de arreglos anidados. Además, en el caso de que se encuentren etiquetas de estilo, estas van a generar dentro del análisis un nuevo nivel en la estructura.

En la Figura 29 se puede observar un diagrama con el que se busca explicar la estructura que se genera al analizar el XML vía la biblioteca predefinida de perl. El primer bloque "*g*", contiene todas las palabras del texto. Dentro de *g* se encuentran los arreglos para cada palabra. Hasta este punto no habría ningún problema con la lectura del archivo; sin embargo, hay situaciones en las que se pueden encontrar una o más palabras dentro de una etiqueta de estilo y/o una o más etiquetas de estilo dentro. Esto da aleatoriedad y mayor profundidad a la conformación de la estructura, complicando así la lectura de la misma y la obtención de resultados, además de tomar mayor tiempo de procesamiento.

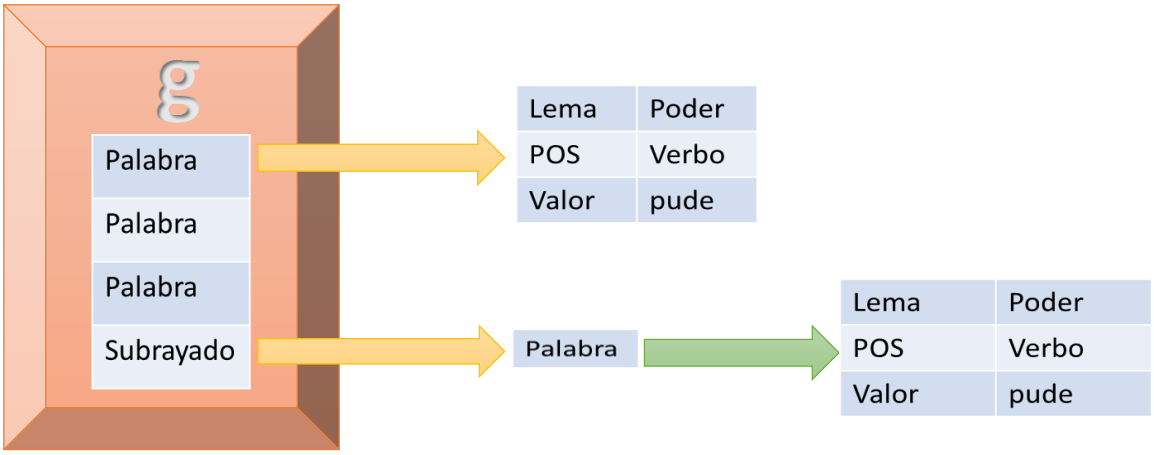


Figura 29. Ejemplo de estructura de la biblioteca para XML

Otra opción para el análisis fue el uso de expresiones regulares, usadas también por el sistema actual, aunque sin tanto peso como en mi propuesta. Una expresión regular es, como lo menciona Fitzgerald, "una cadena de texto especialmente codificada que es usada como patrones para hacer coincidir a conjuntos de cadenas" (2012, p. 1). Entonces, dado que la estructura de cada renglón es la misma para todos los textos del corpus, tener expresiones regulares para analizar los XML se volvió una buena opción. Opción que tomé y exploté para esta propuesta.

Así, el procesamiento del texto se dividió en dos partes: la del encabezado, de donde se obtienen los datos de origen del texto, y la del cuerpo, donde se toman las palabras que son parte del corpus. De modo que lo que opté por hacer fue leer el archivo XML y cargarlo en una tabla hash, misma que va a pasar a Lucy para su indexado, renglón por renglón.

En el método que se ve en la Figura 30 se observa un ciclo en el que mientras existan líneas por leer dentro del archivo, el ciclo continúa. Ya que no tendría sentido continuar leyendo después de la etiqueta que marca el fin del encabezado, es por eso que cuando se lee esta línea, el método termina. La estructura de casos que se ve (*given* - *when*) contiene en

cada uno de estos una expresión regular de la que se obtendrá la información deseada. Esta información es: permisos del documento, título, género y nivel de la lengua, uso, hablante, referencia corta, lugar y fecha.

```
sub parseXML{
    my $documento = shift;
    my $genero = 'sin datos';
    my $nLengua = 'sin datos';
    my $uso = 'sin datos';
    my $hablante = 'sin datos';
    my $titulo = 'sin datos';
    my $referencia = 'sin datos';
    my $lugar = 'sin datos';
    my $fecha = 'sin datos';
    my $permisos = 'sin datos';
    open (XML, "<:encoding(iso-8859-1)", $documento) or die "Problemas para abrir el archivo $!";
    while(<XML>){
        my $reng = Unicode::String->new($_);
        given($reng->utf8){
            when(/.*<.+permisos="(.+)"/i){
                $permisos = $1;
            }
            when(/<titulo>(.+)<\/titulo>/i){
                $titulo = $1;
            }
            when(/genero="(.+)"\snivelLengua="(.+)">/i){
                $genero = $1;
                $nLengua = $2;
            }
            when(/uso="(.+)"\/>/i){
                $uso = $1;
            }
            when(/hablante\sid="(.+)"/i){
                $hablante = $1;
                }
            when(/referencia\sid="(.+)"/i){
                $referencia = $1;
            }
            when(/<lugar>(.+)<\/lugar>/i){
                $lugar = $1;
            }
            when(/<fechaOriginal>(.+)<\/fechaOriginal>/){
                $fecha = $1;
            }
            when(/<\/encabezado>/){
                last;
            }
        }
    }
    close(XML);
    return($genero, $nLengua, $uso, $hablante, $titulo, $referencia, $lugar, $fecha, $permisos);
}
```

Figura 30. Método para obtener encabezados

El siguiente paso en el indexado es el de obtener los tokens del documento. Para el indexado dejé de lado las etiquetas de estilo, a diferencia del proceso actual en donde sí se guardan ciertos valores de estilo de las palabras. Por lo tanto, de la expresión regular sólo se van a obtener: el token, el lema y la parte de la oración. Así, utilizando un ciclo parecido al

de la Figura 30, el programa lee el archivo y por cada renglón que contenga la etiqueta <*g*>, se va a obtener el token correspondiente. Estos datos, más el encabezado, son dados al software Lucy para armar los archivos índice. Éste proceso se realiza para cada archivo del corpus (Figura 31).

En la Figura 31 se puede observar en la parte superior que todo el contenido del archivo es guardado en un arreglo en memoria temporal. Este arreglo va a ser recorrido renglón por renglón hasta que haya sido comparado por completo. Dentro del *If* se encuentra la expresión regular de la que se obtienen toda la información contenida de la palabra (lema, token y parte de la oración). Ya con la información extraída, el siguiente paso es agregar una nueva entrada dentro de Lucy. Éste último paso se ve en la instrucción *add_doc*.

```
open (XML, "<:encoding(iso-8859-1)", $documento) or die "Problemas para abrir el archivo $!";
my @contenido = <XML>;
foreach my $renglon (@contenido){
    my $uniRenglon = Unicode::String->new($renglon);
    if($uniRenglon->utf8 =~ /^\s+<g\sc="([a-z]+)"\sl="([a-z]*.*[a-z]*)">([A-Z]*\p{All}*[A-Z]*)<\/g>/){
        $etiquetas{'lema'} = $2;
        $etiquetas{'POS'} = $1;
        $etiquetas{'token'} = $3;
        $index->add_doc(
        {
        'POS'=>$etiquetas{'POS'},
        'lema'=>$etiquetas{'lema'},
        'token'=>$etiquetas{'token'},
        'documento'=>$etiquetas{'documento'},
        'genero'=>$etiquetas{'genero'},
        'nLengua'=>$etiquetas{'nLengua'},
        'uso'=>$etiquetas{'uso'},
        'hablante'=>$etiquetas{'hablante'},
        'titulo'=>$etiquetas{'titulo'},
        'referencia'=>$etiquetas{'referencia'},
        'lugar'=>$etiquetas{'lugar'},
        'fecha'=>$etiquetas{'fecha'},
        'permisos'=>$etiquetas{'permisos'}
    }
        );

    }

}
close(XML);
```

Figura 31. Ciclo para la obtención de tokens

El proceso anterior se repite hasta que todo el archivo es comparado e indexado. Todos y cada uno de los documentos del corpus realizan el mismo procedimiento. Para indexar los bigramas la diferencia se encuentra en dos puntos: el tiempo en que se van a indexar las palabras y la cantidad de esquemas en el índice. El momento de indexado, en el caso de los bigramas, será cada que dos renglones pasen por la expresión regular. Por su parte, la cantidad de esquemas aumenta dos valores: el lema y la parte de la oración de la segunda palabra

```
my @contenido = <XML>;
foreach my $renglon (@contenido){
    my $uniRenglon = Unicode::String->new($renglon);
    if($uniRenglon->utf8 =~ /^\s+<g\sc="([a-z]+)"\sl="([a-z]*.*[a-z]*)">([A-Z]*\p{All}*[A-Z]*)<\/g>/){
        $etiquetas{'lema'} = $2;
        $etiquetas{'POS'} = $1;
        $etiquetas{'token'} = $3;
        #Carga del último token para juntar los bigramas
        if($biAC == 0){
            $bietiqueta{'lema'} = $etiquetas{'lema'};
            $bietiqueta{'POS'} = $etiquetas{'POS'};
            $bietiqueta{'token'} = $etiquetas{'token'};
            $biAC = 1;
        }elsif($biAC == 1){
                $index->add_doc(
                {
                    'POS2'=>$etiquetas{'POS'},
                    'lema2'=>$etiquetas{'lema'},
                    'bigrama'=>$bietiqueta{'token'}." ".$etiquetas{'token'},
                    'POS'=>$bietiqueta{'POS'},
                    'lema'=>$bietiqueta{'lema'},
                    'documento'=>$etiquetas{'documento'},
                    'genero'=>$etiquetas{'genero'},
                    'nLengua'=>$etiquetas{'nLengua'},
                    'uso'=>$etiquetas{'uso'},
                    'hablante'=>$etiquetas{'hablante'},
                    'titulo'=>$etiquetas{'titulo'},
                    'referencia'=>$etiquetas{'referencia'},
                    'lugar'=>$etiquetas{'lugar'},
                    'fecha'=>$etiquetas{'fecha'},
                    'permisos'=>$etiquetas{'permisos'}
                }
                );
            $bietiqueta{'lema'} = $etiquetas{'lema'};
            $bietiqueta{'POS'} = $etiquetas{'POS'};
            $bietiqueta{'token'} = $etiquetas{'token'};
        }
    }

}
```

Figura 32. Ciclo de análisis para bigramas

(Figura 32).

El tiempo total de ejecución se contabiliza, mediante un cronómetro, desde que el ciclo de lectura de archivos es iniciado en los unigramas, ya que este análisis es iniciado un

poco antes que el de los bigramas. El cronómetro se detiene entonces cuando el archivo de bigramas termina el indexado e informa de esto. En el siguiente capítulo se muestran los tiempos de indexado obtenidos en las pruebas.

III.3.2. El generador de concordancias

Un problema al que me enfrenté para diseñar una propuesta fue que el sistema necesitaba generar concordancias (fragmentos textuales) y no únicamente recuperar los documentos completos, como es lo normal en una estrategia de RI. Si bien se pueden ver como procesos muy parecidos, la RI y la generación de concordancias son dos procesos distintos. Sin embargo, en un principio para mí eran lo mismo, incluso creía que la recuperación debía ser un proceso más completo. Al final, reconocí que estaba equivocado, pero se puede encontrar en la RI un gran aliado para obtener las concordancias.

Como ya lo he mencionado, Lucy es una herramienta enfocada mayormente para la RI, así que había que adaptarla para obtener las concordancias del corpus. Con esta decisión ya tomada, tuve que armar las piezas que me ayudarían con la recuperación de las concordancias. Mi meta en esta fase fue la de obtener documentos XML con las concordancias, como los que obtiene el sistema actual.

Al obtener una salida similar a la del sistema actual busqué facilitar la transición entre uno y otro motor de búsqueda, evitando así tocar la interfaz web actual. Otra ventaja que me daría tener los documentos de las concordancias en XML es que, al tener los sistemas la misma salida, esto me permitiría medir el tiempo de respuesta entre los dos sistemas hasta un punto en común.

En este momento del desarrollo tengo cuatro piezas (problemas) que resolver. Por un lado, la salida de Lucy es una lista de palabras y su información, incluido el título del texto

y el documento; no son los fragmentos textuales que necesito para formar las concordancias. Por otro lado, tengo la necesidad de obtener la información de las referencias (información bibliográfica que acompaña la concordancia) cruzando claves con el contenido del documento en donde están almacenadas todas las referencias. La obtención de las concordancias a partir de cada uno de los archivos del corpus que se encuentran en la lista de resultados de Lucy es otra de las piezas. Finalmente, la última pieza es la impresión de las concordancias en un documento XML.

Dada la salida de resultados que me entregaba Lucy, opté por realizar la recopilación de cada uno de los nombres de los archivos en donde se encuentran los resultados, para posteriormente abrirlos y procesarlos para obtener las concordancias. De modo que lo que tengo después de hacer la consulta a Lucy es un arreglo con los nombres de los archivos y los resultados dentro de cada uno de ellos. Un ejemplo más claro se puede ver en la Tabla 4.

Archivo	**Resultados**	
Documento1	**Token1**	Información
	Token2	Información
	Token2	Información
Documento2	**Token3**	Información
	Token2	Información
Documento3	**Token5**	Información

Tabla 4. Ejemplo de arreglo de resultados

De la Tabla 4 se observa que un documento puede contener una o más apariciones del token buscado. Sin embargo, cada token tiene su propia información por lo que es necesario extraer de manera individual cada una de estas apariciones en los archivos. Para esto,

consideré en primera instancia el uso de la biblioteca XML de perl, pero al hacer las pruebas de ejecución con el 25% de los documentos del corpus, el resultado fueron 5 minutos para la extracción de concordancias.

El tiempo obtenido de esta prueba resulta inaceptable pues la generación de concordancias es la parte que se encuentra en contacto con el usuario final, además de que este tiempo está dado sin la escritura del documento XML final. También observé que incluía muchos errores y omisiones en el análisis por lo que la opción que tomé de nuevo fue la de usar expresiones regulares para identificar el renglón buscado y la concordancia deseada.

Con la tabla de resultados arrojada por Lucy y la tarea cíclica y repetitiva que representa la obtención de las concordancias, decidí realizar la tarea en distintas trazas de ejecución. De este modo cada archivo sería abierto por un hilo distinto, generando así un mayor consumo de tiempo del procesador y al mismo tiempo un menor tiempo de ejecución. Aunque es cierto que si la palabra buscada se encuentra en 150 archivos, este mismo número es la cantidad de hilos que son lanzados, afortunadamente el tiempo de ejecución aun así resultó aceptable.

Con el fin de explicar más a detalle el algoritmo para la recuperación de las concordancias, me apoyaré en el código tal cual lo hice con el indexador. En esta ocasión lo primero que realicé fue inicializar un nuevo objeto de Lucy, objeto de consulta. Este objeto está prediseñado para leer los archivos-índice y hacer la búsqueda correspondiente. El único paso para inicializarlo es ingresar el directorio donde se encuentran los archivos-índice.

El siguiente paso es leer la petición del usuario, este paso consiste en una lectura de valores a partir del teclado. Estos valores comienzan por la palabra o frase, dentro de la palabra se incluyen el lema o la parte de la oración según sea el caso de la búsqueda. Continúan con la ventana de palabras deseada. Estos dos valores, palabra y ventana, van a formar la concordancia a recuperar. También van a ser necesarios otros valores, complementos en la búsqueda, como el orden, el nivel de la lengua, uso y género.

Aunque el programa cuenta con el código para leer la petición directamente del usuario, no lo presento ya que una meta es leer estos datos con la interfaz web. Por esto es que prefiero únicamente plantear la idea del algoritmo y no mostrar el código como lo he venido haciendo. Como se ha venido planteando a lo largo de esta tesis, un objetivo es no tocar la interfaz web y poder realizar una migración transparente para el usuario final y simple para los administradores del sistema.

Continuando con el algoritmo del sistema, el primer punto después de la obtención de la petición es la clasificación de la misma. Esto es, identificar si se requiere una o más palabras es importante, pues los archivos-índice a consultar son distintos para cada caso (Figura 33). Además, el objeto de consulta que se requiere de Lucy también va a depender de éste dato.

```
sub ObjetoConsulta{
    if($peticion{'palabra'} =~ /^("?\w+)\s\w+.*/i){
        print "------------------->Petición multipalabra\t¬".$peticion{'palabra'}."\n";
        MultiPalabra();
    }else{
        $peticion{'palabra'} =~ s/ +//g;
        print "------------------->Petición unipalabra\t¬".$peticion{'palabra'}."\n";
        UniPalabra();
    }
}
```

Figura 33. Bloque para identificar la cantidad de palabras de la consulta.

Entonces, el sistema identifica qué tipo de consulta es y procede con el siguiente análisis. Este es para saber si se va a buscar por lema, por parte de la oración, búsqueda exacta o por fragmento de la palabra. El análisis en este punto se realiza con una instrucción de *switch*, con la que el sistema va a comparar la cadena de entrada contra una serie de expresiones regulares para determinar los parámetros de búsqueda dentro del esquema Lucy (Figura 34). Estos parámetros: el término de búsqueda y el campo donde se va a buscar, van a ser utilizados más adelante al momento de instanciar un objeto de búsqueda Lucy (Figura 36).

```
SWITCH: {
        $peticion{'palabra'} =~ /^"(\w+)"$/i && do{ #Funciona correctamente, resta convertir cadena a mayús
            $peticion{'palabra'} = $1;
            $peticion{'palabra'} =~ tr/a-z/A-Z/;
            print "Objeto TermQuery-token. \n $peticion{palabra} \t A la base de datos de unigrama\n";
            $consulta = Lucy::Search::TermQuery->new(
                field => 'token',
                term => $peticion{'palabra'},
            );
            last SWITCH;
        };
        $peticion{'palabra'} =~ /^\[(\w+)\]$/i && do{ #Funciona correctamente 23/sep/2013
            $peticion{'palabra'} = $1;
            $peticion{'palabra'} =~ tr/A-Z/a-z/;
            print "Objeto TermQuery-lema. \n $peticion{palabra} \t A la base de datos de unigrama\n";
            $consulta = Lucy::Search::TermQuery->new(
                field => 'lema',
                term => $peticion{'palabra'},
            );
            last SWITCH;
        };
        $peticion{'palabra'} =~ /(^\w+\*)|(^\*\w+)/i && do{
            $campo = 'token';
            $termino = $1;
            $termino = $2 if $2;
            print "Objeto LucyX->WildcardQuery. \n $termino \t A la base de datos de unigrama\n";
            last SWITCH;
        };
        $peticion{'palabra'} =~ /<(\w+)>(\w+)?/i && do{#Funciona correctamente 23/sep/2013
            $campo = 'POS';
            $termino = $1;
            if($2){
                $peticion{'palabra'} = $2;
                $peticion{'palabra'} =~ tr/a-z/A-Z/;
                 my $TOKEN_consulta = Lucy::Search::TermQuery->new(
                        field => 'token',
                        term => $peticion{'palabra'},
                    );
                $1 =~ tr/A-Z/a-z/;
                 my $POS_consulta = Lucy::Search::TermQuery->new(
                        field => 'POS',
                        term => $1,
                    );
```

Figura 34. Fragmento del bloque de identificación del parámetro de búsqueda Lucy.

Un bloque de instrucciones que cabe resaltar dentro de estas opciones de decisión, es la de búsqueda por parte de la oración (Figura 35). Y es que esta búsqueda no se encuentra aún disponible dentro de las opciones del sistema actual y tiene dos formas de realizarse. Por una parte se pueden buscar todas las concordancias de cualquier palabra que se encuentre en la parte de la oración deseada. Un ejemplo sería el de obtener todas las palabras que sean verbo.

```
$peticion{'palabra'} =~ /<(\w+)>(\w+)?/i && do{#Funciona correctamente 23/sep/2013
    $campo = 'POS';
    $termino = $1;
    if($2){
        $peticion{'palabra'} = $2;
        $peticion{'palabra'} =~ tr/a-z/A-Z/;
         my $TOKEN_consulta = Lucy::Search::TermQuery->new(
                field => 'token',
                term => $peticion{'palabra'},
            );
        $1 =~ tr/A-Z/a-z/;
         my $POS_consulta = Lucy::Search::TermQuery->new(
                field => 'POS',
                term => $1,
            );
        $consulta = Lucy::Search::ANDQuery->new(
            children => [ $TOKEN_consulta, $POS_consulta],
        );
    }
    else{
        $peticion{'palabra'} = $1;
        $peticion{'palabra'} =~ tr/A-Z/a-z/;
        $consulta = Lucy::Search::TermQuery->new(
            field => 'POS',
            term => $peticion{'palabra'},
        );
    }
    print "Objeto TermQuery-POS. \n A la base de datos de unigrama\n";
    last SWITCH;
};
```

Figura 35. Bloque de decisión para la consulta POS.

Por otra parte, también se puede especificar la palabra y la parte de la oración dentro de la consulta, acotando más el rango de resultados. La diferencia en esta decisión radica en el objeto de búsqueda Lucy que se va a crear, ya que para conseguir resultados en esta búsqueda hay dos opciones de objetos. Una es la de intersectar los múltiples conjuntos de resultados, mediante el uso de objetos “hijo” de búsqueda (instrucciones de la parte superior de la Figura 35). La segunda opción es la de continuar con el objeto usado para las demás opciones (instrucciones de la parte inferior de la Figura 35).

Una vez que se han terminado de identificar los parámetros del objeto de búsqueda Lucy (término y campo), el siguiente paso es realizar la consulta. Esta instrucción es relativamente corta y sencilla, con ella se va a obtener un conjunto de resultados a partir de los cuales se

van a obtener las concordancias. Esta instrucción, con el paso de los parámetros de búsqueda se muestra en la Figura 36.

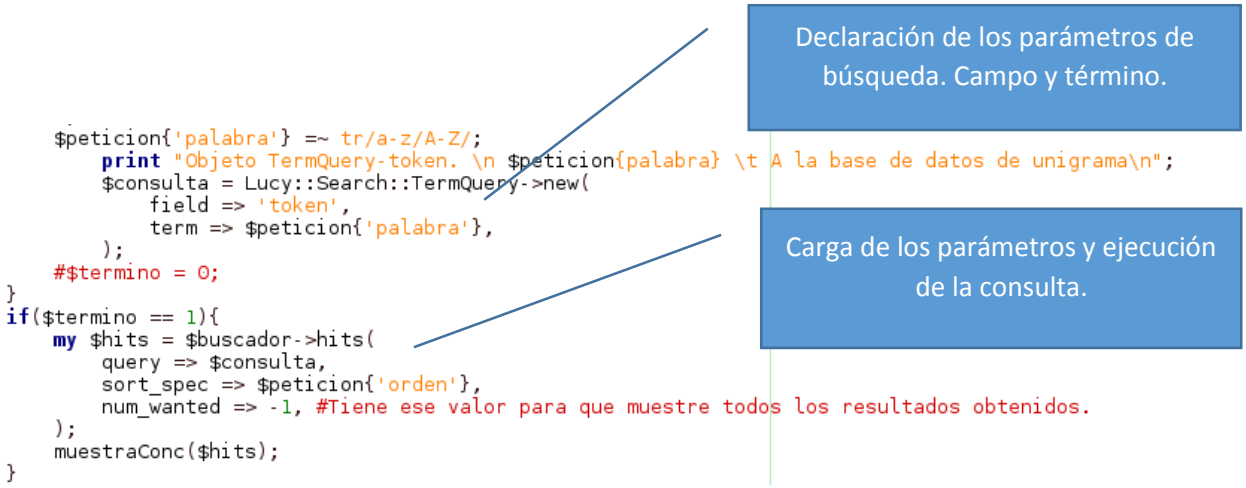


Figura 36. Bloque de consulta Lucy.

Ya con el conjunto de resultados, el siguiente proceso es el de mostrar las concordancias. En el diagrama de flujo, dentro de la sección III.2 Diseño, mostré el procedimiento para imprimir el documento XML. Este procedimiento es únicamente para mostrar los resultados, no para la generación de las concordancias, por lo que no continuaré detallando este punto. En cambio, entraré en detalle en la generación de las concordancias.

Recordando un poco sobre el proceso que se sigue para la generación de las concordancias, una vez que se tienen todos los documentos en donde hay resultados, se va a generar un hilo por cada uno de estos archivos. El hilo creado va a imprimir dentro del documento XML todas las concordancias encontradas dentro del documento que le fue asignado. Para esto, los parámetros asignados al hilo creado son:

1. El archivo de los resultados (archivo que contiene concordancias).
2. Nombre del archivo XML final.
3. Tamaño de la ventana.
4. Petición de la búsqueda.

5. Fecha del documento.

6. ID de referencia bibliográfica del documento.

7. Parte del texto XML que le toca imprimir (encabezado, cuerpo o pie de página).

```
my $pid = fork();
      if($pid == 0){
            system("perl ObtenConcvTh3.pl ".$docs." ".$peticion{'pala-
      bra'}."V".$peticion{'ventana'}."N".$ac." ".$peticion{'ventana'}."
      ".$peticion{'palabra'}." ".$documentos{$docs}{'rc'}." ".$encabe-
      zado);
                  exit 0;
            }
```

Figura 37. Bloque de creación y ejecución de un hilo hijo.

En la Figura 37 se observa el bloque de código donde se crea un hilo con la instrucción *fork()*. El hilo creador recibe un número identificador del nuevo proceso creado mientras que el hilo hijo obtiene un cero como número identificador. La decisión que sigue al *fork()* permite conocer si la traza es la creadora o la creada, separando así el camino a seguir para cada hilo. Dentro del bloque de código de la instrucción de decisión, es el hilo creado el que va a ejecuta la obtención de las concordancias. Con la instrucción *system*, este hilo ejecuta el programa que va a recuperar las concordancias mediante una instrucción del propio sistema operativo (Figura 37).

Una vez ejecutado el programa, lo primero que se ha de realizar es el almacenamiento de los parámetros de entrada dentro de las variables correspondientes del programa. Un punto importante en esta inicialización de variables es la inicialización del documento con los resultados, ya que este documento va a ser abierto y se van a extraer y guardar dentro de un arreglo los renglones que formen parte del cuerpo del documento. Este arreglo va a permitir más adelante hacer una búsqueda mediante una función nativa de Perl y extraer las concordancias a partir de los índices de éste arreglo (Figura 38).

```
sub abreArchivo{
```

```
	my $doc = shift;
	open FILE, "<:encoding(iso-8859-1)", $doc or die "Problemas para
leer el archivo $!";
	my @lineas;
	my $texto = 0;
	while(<FILE>){
		chomp;
		if(/<g/){
			$texto = 1;
		}elsif(/<p>|<\/p>/){
			next;
		}
		push @lineas, Unicode::String->new($_)->utf8 unless $texto ==
0;
		if(/<\/cuerpo>/){
```

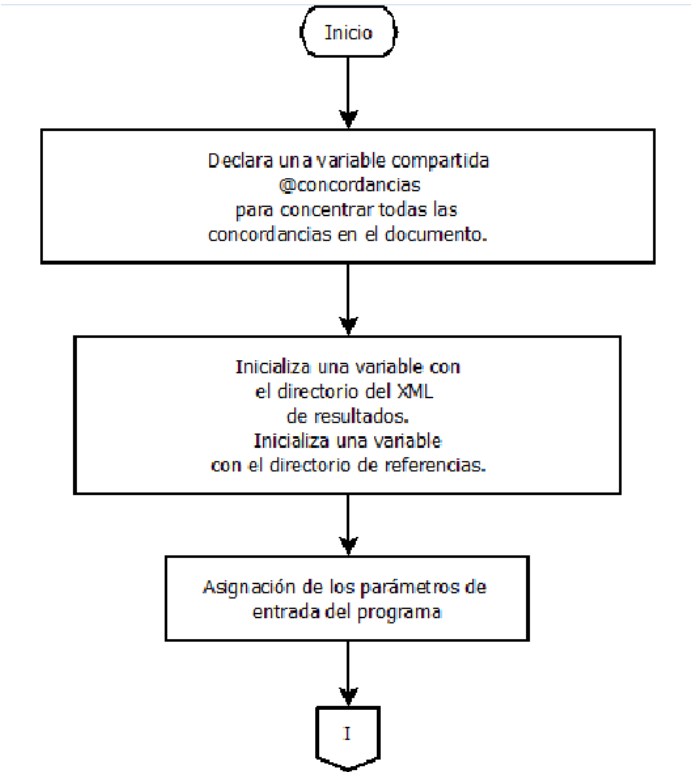


Figura 39. Diagrama de flujo de la generación de concordancias. Parte I

```
			last;
		}
	}
	close FILE;
	return @lineas;
}
```

Figura 38. Subrutina para la lectura del cuerpo del documento.

En el fragmento del diagrama de flujo mostrado en la Figura 39 se ve la declaración de una variable compartida (*@concordancias*) en la cual, todos los hilos creados dentro de este

programa van a almacenar las concordancias generadas. Este tipo de variables en Perl permiten mantener la integridad de la información en un programa multihilos como éste. La declaración de la variable compartida y de los parámetros de entrada, a manera de diagrama de flujo, se muestra en la Figura 39.

El siguiente paso es el de obtener la referencia bibliográfica en formato reducido (referencia corta, por ejemplo “Rulfo, Juan. Pedro Páramo”) del documento a partir del ID de la referencia bibliográfica recibida dentro de los parámetros. Hay que recordar que este ID es una clave dentro de cada documento del corpus que va a permitir ubicar algunos datos bibliográficos de éste. Estos datos se encuentran almacenados en un archivo aparte, con la finalidad de evitar redundancia de información, ya que muchos documentos comparten ciertos valores (para los datos bibliográficos del ejemplo anterior el ID sería "r00100").

Del documento de referencias entonces, el programa va a obtener el autor del documento como se muestra en la Figura 40. El procedimiento es el mismo que se ha venido utilizando en los códigos anteriores, el archivo es abierto y cargado en memoria temporal, se lee cada renglón y se compara con una expresión regular. La diferencia en este caso es que una vez que es encontrado el ID, únicamente se indica que el renglón con la etiqueta <corta> es el que contiene la información requerida.

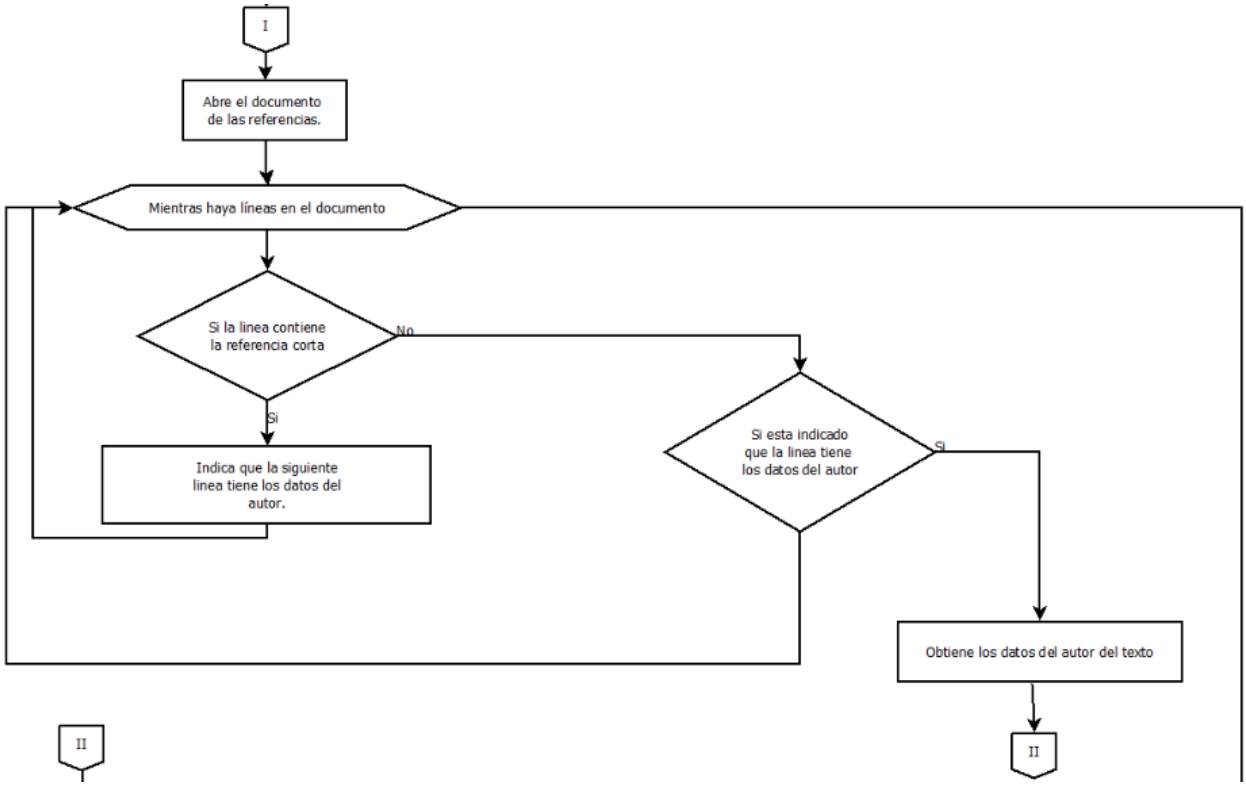


Figura 40. Diagrama de flujo de la generación de concordancias. Parte II

Una vez que se cuenta con los datos bibliográficos del documento, mismos que se van a imprimir junto con cada una de las concordancias contenidas en él, el siguiente paso es obtener las concordancias. Para esto utilizo la función *grep()*, con la que obtengo el índice de cada una de las ocurrencias de la palabra buscada. Este índice es obtenido del arreglo donde se guardó, en un principio del programa, el cuerpo del documento. Es decir que si dentro de

un documento se encuentra cuatro veces la palabra buscada, cuatro índices diferentes son las que se van a obtener.

```
sub HijosParaConc {
      my $target = shift;
      my @resultados = grep{$contenido[$_] =~ /$target/i} 0..$#contenido;
      my @hijos;
      my $num = 1;
      foreach my $indice (@resultados){
            my $th = threads->new(\&GuardaConc, $indice, $num);
            $num++;
            push @hijos, $th;
      }
      foreach (@hijos){
            $_->join;
      }
}
```

Figura 41. Subrutina HijosParaConc.

En la Figura 41 se observa la subrutina HijosParaConc, dentro de la cual se encuentra la función *grep()* antes mencionada, así como la variable *@contenido* la cuál es el arreglo en el que se encuentra el cuerpo del documento. Esta subrutina va a extraer todos los índices de la palabra buscada (en el código es la variable *$target*), para posteriormente ejecutar por cada índice un nuevo hilo que va a generar cada concordancia. Esta instrucción se encuentra expresada en *threads->new()*. En el siguiente ciclo, la traza central espera a cada uno de los hilos creados para continuar con la impresión de todas las concordancias.

Estos ciclos también se pueden apreciar a manera de diagrama de flujo dentro de la Figura 42. Algo a resaltar de este segmento del diagrama es el área con líneas punteadas, pues esta parte la van a ejecutar cada uno de los hilos creados. Esta misma sección es lo que en el código de la Figura 41 se ve como la instrucción *threads->new()*. Aunque en el diagrama de flujo se ven dos líneas más, una que sale y otra que entra, estas indican el proceder de los

hilos creados. Este segmento del diagrama termina con la apertura del XML de las concordancias justo después de haber esperado a que cada una de las trazas creadas terminara con su tarea.

La razón para esperar a cada una de las trazas para imprimir las concordancias en un archivo en lugar de que cada hilo imprima directamente dentro del XML de salida, radica en el orden y la cantidad de hilos creados. Es necesario en este punto recordar que éste programa está siendo ejecutado por una hilo hijo creado del programa que realizó la búsqueda. Por lo que estas mismas líneas de código están siendo ejecutadas por tantos hilos como documentos tienen respuestas.

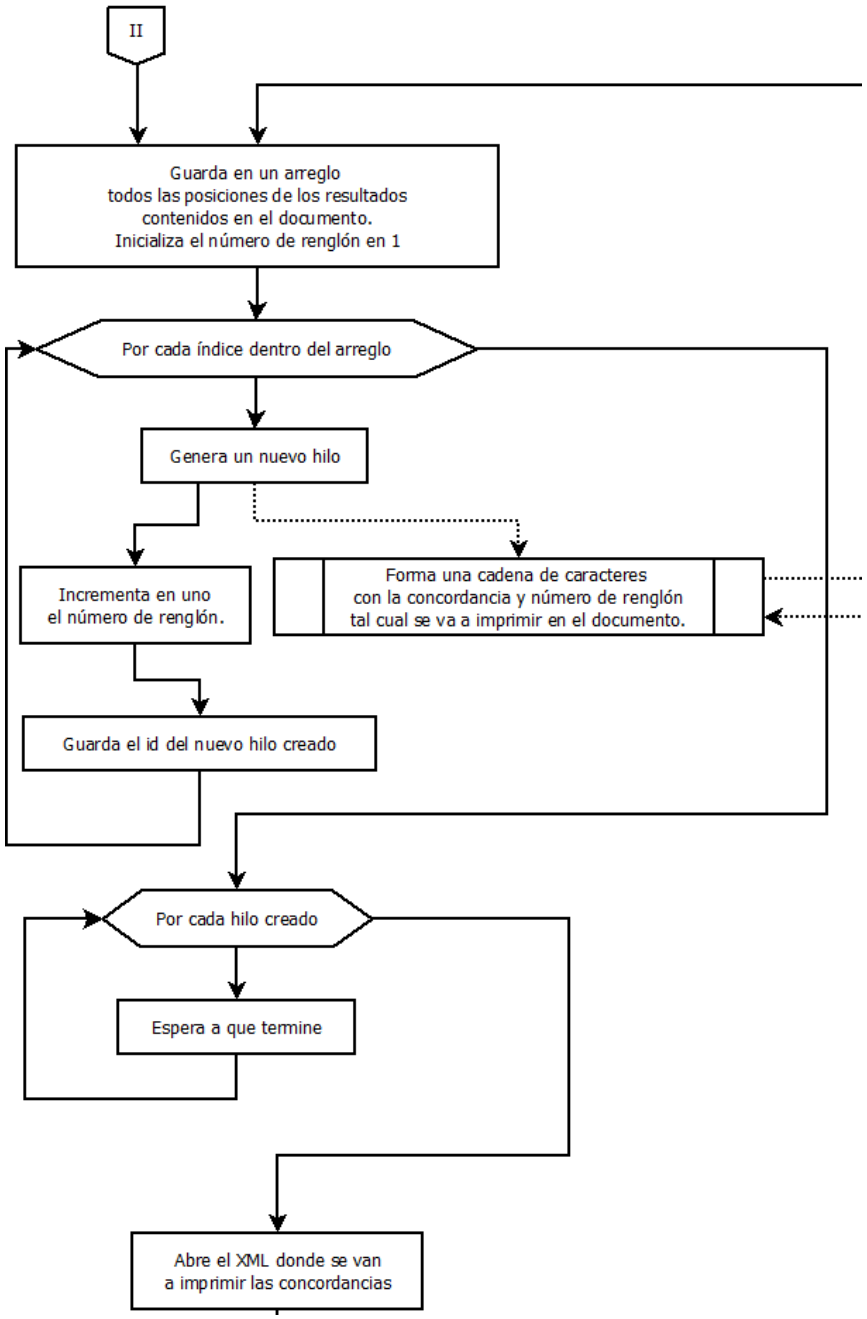


Figura 42. Diagrama de flujo de la generación de concordancias. Parte III

Considerando el punto anterior, más la cantidad de hilos que van a escribir en el mismo XML, la integridad de las concordancias dentro del archivo se vería comprometida si cada hilo escribiera de manera paralela en el archivo XML de salida. Ya que al momento de la impresión se tendrían $\alpha+\beta+1$ cantidad de hilos abiertos. De donde α es igual al número de

resultados, β es el número de documentos con resultados y el 1 es la traza principal que realizó la consulta.

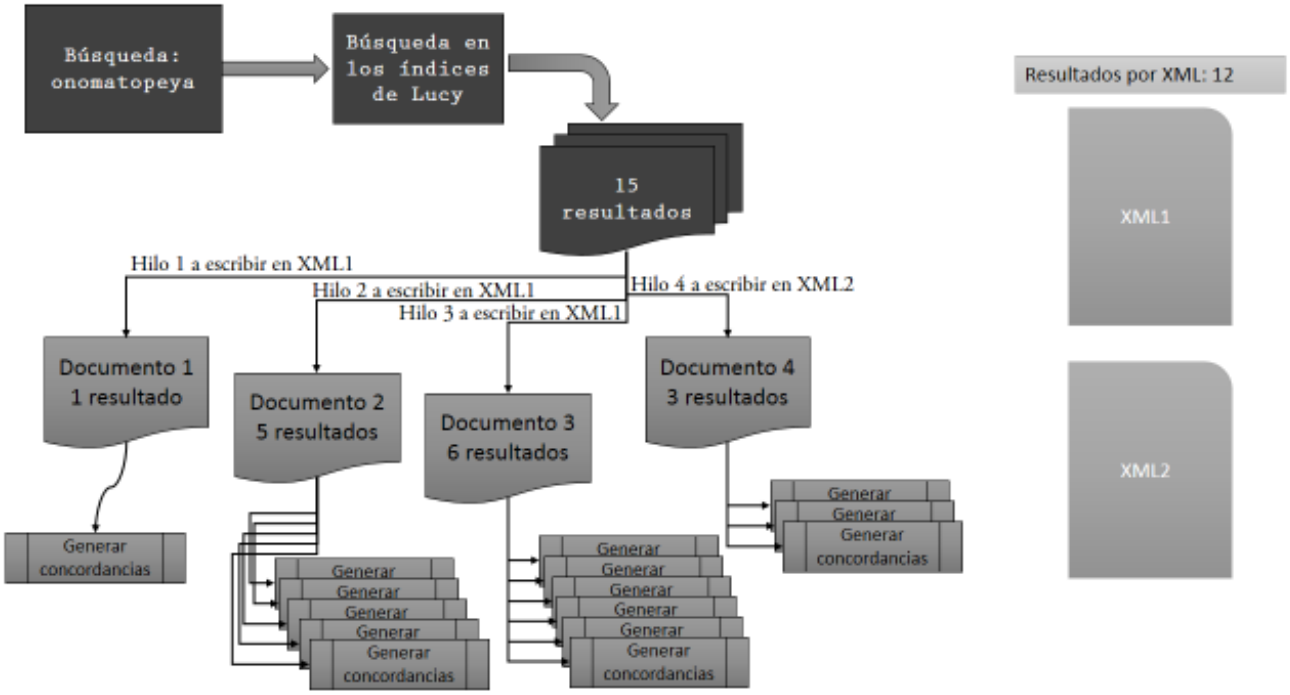


Figura 43. Ejemplo de una consulta y la generación de hilos dentro de ella.

Un ejemplo más claro sobre la cantidad total de hilos creados y el trabajo de cada uno de ellos se puede ver en la Figura 43. Para este ejemplo, la palabra buscada es *onomatopeya*. Según el procedimiento explicado, lo primero que se realiza es la búsqueda con Lucy para de aquí obtener un listado de los documentos que contienen resultados. En el ejemplo se obtuvieron 15 resultados en 4 documentos distintos. El siguiente paso es abrir cada uno de estos documentos con una traza diferente, en este punto hay 5 hilos en ejecución. Cada uno de estos hilos a su vez va a crear un hilo por cada ocurrencia del resultado, para llegar a tener hasta 20 hilos en ejecución.

Si cada uno de los quince hilos quisiera escribir sus concordancias directamente en el XML de salida, habría concordancias que no completarían su escritura en un solo renglón

ocasionando así un XML incorrecto e incapaz de ser leído por un explorador web. Además de que las concordancias no estarían bien formadas. Por esto es que utilicé una variable compartida dentro de cada hilo de los documentos, en la que se almacenan todas las concordancias obtenidas por los hijos de la traza, para evitar estos problemas de concurrencia.

Ya en el hilo generador de la concordancia, el primer paso es la declaración del inicio y del fin del renglón. Ya que cada palabra del documento está escrita en un renglón distinto, en primera instancia será un arreglo el que almacene la concordancia. Para esto, en un primer ciclo se van a obtener las palabras del lado izquierdo de la concordancia, hasta cumplir con la cantidad de palabras de la ventana o que se haya recorrido todo el arreglo, lo que suceda primero. Esto se puede observar dentro de la Figura 44.

De esta misma figura (Figura 44), se observa antes de comenzar el ciclo la declaración de una variable con el nombre de *etiqueta*. El objetivo de esta declaración es el de identificar qué etiquetas de estilo fueron abiertas y no fueron cerradas adecuadamente. La comprobación de esta condición es realizada ya dentro del ciclo de llenado, como se muestra en la primera decisión de la Figura 44. En caso de que el renglón en turno cuente con alguna etiqueta, la siguiente decisión será para ver si la etiqueta se cerró antes de ser abierta. Estas verificaciones se realizan mediante el uso de operadores booleanos.

Podrá parecer extraño que en la decisión sobre el cierre antes de una apertura la opción verdadera sea la operación correcta, tal como se muestra en la Figura 44. Sin embargo, como el llenado de la parte izquierda de la concordancia se hace a partir de la obtención de los

renglones de un arreglo, los elementos se van obteniendo del último al primero. Así que si el elemento ha pasado correctamente por estas decisiones será agregado a las concordancias.

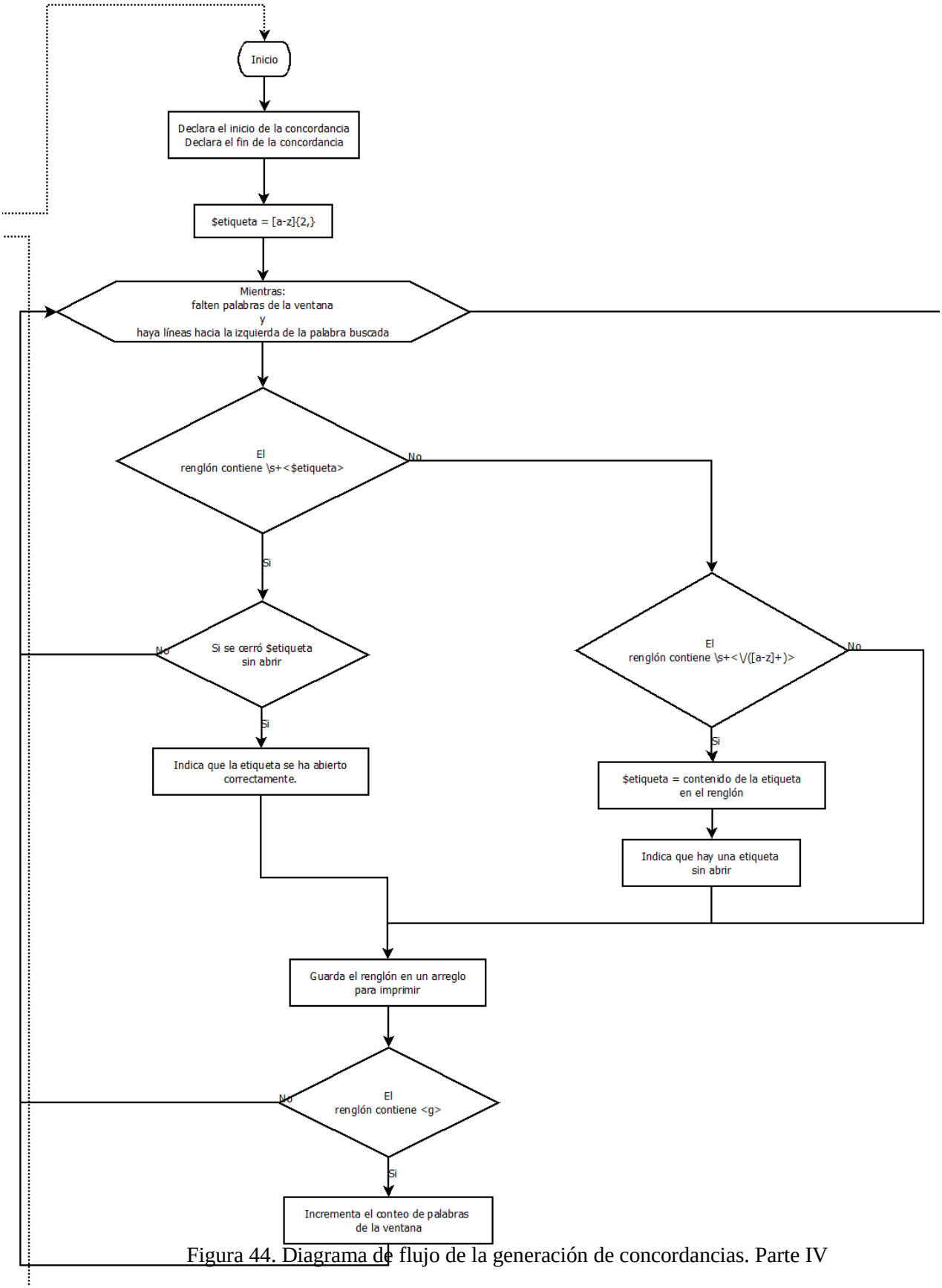


Figura 44. Diagrama de flujo de la generación de concordancias. Parte IV

Por otra parte se verifica si la etiqueta es de cerradura, en caso positivo, se levanta una bandera para indicar que hay una etiqueta cerrada sin abrirse y se continúa hasta guardar el renglón dentro de la concordancia. En caso de que el elemento no contenga ninguna etiqueta de estilo, entonces esto indicaría que es una palabra, lo que se verifica en la última decisión. En caso de ser positiva la verificación, el conteo de palabras de la ventana se incrementa y el ciclo continúa.

Este mismo ciclo se realiza para el llenado del lado derecho de la concordancia, con la variante de que los elementos en este caso serán recorridos de principio a fin (hacia la derecha) y no al revés como en el caso anterior. A continuación se muestra en la Figura 45 la programación para el almacenamiento del lado derecho de una concordancia. Estas líneas ocurren una vez que se ha guardado el lado izquierdo de la concordancia y la palabra buscada, el centro de la concordancia.

```
$etiqueta = "[a-z]{2,}";
      $palabras = $peticion{'ventana'};
      my $auxD = $indice+1;
      while(($palabras != 0)&&($auxD < @contenido)){
            if($contenido[$auxD]=~/<([a-z]+)>/i){
                  $etiqueta = $1;
                  $cerrar[0] = 1;
            }elsif($contenido[$auxD]=~/<\/$etiqueta>/i){
                  if ($cerrar[0] == 0){
                        $cerrar[1] = 0;
                        $auxD++;
                        next;
                  }
                  else{
                        $cerrar[1] = 1;
                  }
            }
            push @concordancia, Unicode::String->new($contenido[$auxD])-
>utf8;
            if($contenido[$auxD] =~ /<\/g>/i){
                  $palabras--;
            }
            @cerrar = (0,0) unless ($cerrar[0] != $cerrar[1]);
            $auxD++;
      }
      if(($cerrar[0] == 1) && ($cerrar[1]==0)){
```

```
        push @concordancia,"</$etiqueta>";
    }
```

Figura 45. Segmento de código que muestra la generación del lado derecho de una concordancia.

En la Figura 45 se observa la variable *etiqueta*, que contiene una cadena de caracteres en forma de expresión regular. Esta inicialización es necesaria ya que en la instrucción elseif se hace la comparación para conocer si hay etiquetas de estilo (por ejemplo <subrayado>) cerrando otra etiqueta abierta previamente. De no inicializar *etiqueta*, el sistema puede parase por un error de referencia al estar referenciando una variable vacía. Ya que en el primer *if* se guarda en esta misma variable (*etiqueta*) la etiqueta de estilo encontrada, los valores de la inicialización no van a durar mucho tiempo dentro de la variable.

El *if* que prosigue al almacenamiento del renglón dentro de la concordancia (push @concordancia…), verifica si el renglón insertado corresponde a una palabra, para disminuir en uno la cantidad de palabras de la ventana. Esto con el fin de conocer la cantidad de palabras faltantes para completar la ventana. Una vez completado el llenado del lado derecho de la concordancia, el siguiente *if* verifica que no haya quedado ninguna etiqueta abierta sin cerrar. En caso contrario, cierra la etiqueta.

La Figura 46 muestra las últimas líneas de la subrutina que genera la concordancia. La primera de esas líneas ingresa la parte inicial de la concordancia y la segunda línea ingresa el fin de la misma. La línea siguiente va a generar una cadena de caracteres simple a partir del arreglo donde se ha guardado la concordancia generada previamente. El siguiente grupo de instrucciones va a bloquear el acceso a la variable compartida *concordancias*, para que el hilo correspondiente escriba sin errores e interrupciones la concordancia generada. Una vez terminada la escritura, el hilo termina su tarea, libera la variable compartida y muere.

```
unshift @concordancia, @encabezado;
push @concordancia, @pieDeP;
```

```
my $renglonResultado = join "\t", @concordancia;
{
      lock (@concordancias);
      push @concordancias, $renglonResultado;
}
```

Figura 46.Segmento de código que muestra el final de la generación de una concordancia.

Ya en este punto del código, lo que resta para la generación de las concordancias es la impresión dentro del archivo XML de salida. Esta acción no es muy complicada, ya que como se muestra en la Figura 47, el sistema espera a que terminen todas las trazas e imprime la parte del texto que le corresponde, a partir de la variable compartida. Una vez realizada esta acción, esta traza muere y la traza central, que es la que realizó la búsqueda con Lucy, espera

a que todas sus trazas terminen para indicar que la generación de las concordancias ha terminado.

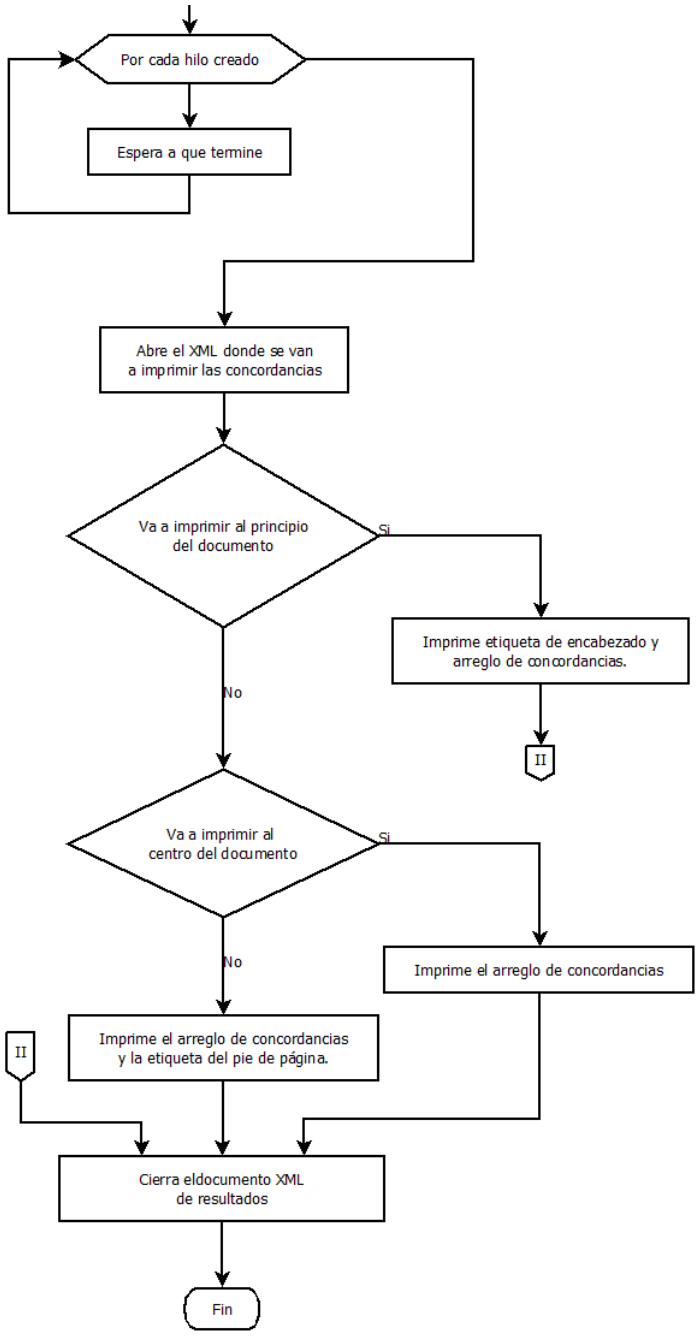


Figura 47. Diagrama de flujo de la generación de concordancias. Parte V.

En las subsecciones anteriores se han descrito los procedimientos propuestos para indexar los documentos del corpus y generar las concordancias a partir de la petición del usuario. Los resultados y evaluación de estos procedimientos se presentarán en una sección posterior.

III.4. Requerimientos para el uso de los programas

Tanto la instalación del indexador como del generador de concordancias únicamente requiere de cubrir los requerimientos básicos del sistema. Los requerimientos de software son:

1. Perl 5, versión 16, subversión 3 (v5.16.3) o superior.
2. Bibliotecas para compilar C y C++ (requeridas para Lucy).
3. Apache Lucy 0.3.2 o superior. La API se puede descargar de http://lucy.apache.org/.
4. Bibliotecas de la red de archivos integrales de perl o CPAN por sus siglas en inglés (*Comprehensive Perl Archive Network*). En el caso de que los programas se ejecuten en un sistema Linux, se puede instalar el paquete que permite acceso directo a este repositorio. Esta instalación depende de la versión de Linux que se tenga.

Los módulos de CPAN necesarios para ejecutar correctamente los programas son:

1. Unicode::String. Para el tratamiento de los archivos con esta codificación (Unicode).
2. Switch:Plain
3. Bundle::CPAN
4. Tcl::Tk

En la estructura de las carpetas sólo se deben considerar una carpeta para almacenar los archivos-índice de los unigramas y una carpeta más, que va a almacenar los archivos-índice de los bigramas. Una vez que se cuente con estos requisitos, el sistema se puede comenzar a utilizar. Bastará con escribir los comandos que permitan ejecutar archivos de Perl

IV. Resultado y evaluación

En este capítulo expondré los resultados obtenidos en los experimentos con la nueva propuesta de arquitectura de corpus y una evaluación de ellos.

Al comienzo de este trabajo pensaba que una buena opción para evaluar mi propuesta sería comparar ambos sistemas y elegir al mejor; sin embargo, al final esto no fue una opción justa para ninguno de los dos sistemas. Esto se dio ya que, con el transcurso de la investigación, los sistemas se fueron separando (diferenciando) en muchos aspectos, desde el paradigma de programación hasta el enfoque con el que se obtienen las concordancias. Es por esto que preferí evaluar de manera más individual y menos comparativa la propuesta presentada en esta tesis.

Uno de los objetivos de esta propuesta ha sido el de tener un sistema capaz de soportar corpus de gran tamaño, por lo que es importante predecir el tiempo de respuesta del sistema para distintos tamaños de corpus. Para esto, utilicé dos corpus: el CEMC y un corpus artificial de aproximadamente 30 millones de palabras. Ambos fueron divididos en cuatro tamaños distintos. Todos los detalles de estas pruebas los defino a continuación.

El corpus artificial utilizado para las pruebas de esta propuesta es un corpus realizado en el GIL, con las siguientes características:

1. 1001 documentos en XML con el mismo formato de etiquetado que presenta el CEMC.
2. 30,414,322 líneas escritas. Esta cantidad de líneas fueron contadas de manera automática. En ellas se incluyen las líneas de encabezados, del cuerpo del documento y pie de página.
3. 30,378,286 ocurrencias. Las ocurrencias en este corpus son un conjunto de caracteres alfanuméricos agrupados y generados de manera aleatoria.
4. Cada archivo maneja una cantidad de 30,347.93 palabras en promedio. Todos los archivos manejan la misma cantidad de palabras gracias a que el corpus está hecho automáticamente y con estos valores controlados.
5. La generación de los textos fue realizada de manera automática, manteniendo la densidad de palabras de un texto en español[21].

Con el uso de este corpus artificial busqué tener una idea de la reacción del sistema ante un corpus real de un tamaño similar. El indexado lo realicé sobre las cantidades de documentos mostradas en la Tabla 5.

Número de archivos	Tiempo [s]	Tiempo [min]
1	4.55	0.08
250	736.77	12.28
500	1553.20	25.89
750	2449.74	40.83
1001	3269.25	54.49

Tabla 5. Tiempos de indexado para los documentos del corpus artificial.

[21] La creación de este corpus artificial fue una tarea asignada a uno de los becarios del GIL (Alonso Michel Cruz Corona). Este becario desarrolló un programa en Java para generar aleatoriamente cadenas de caracteres a manera de palabras de una lengua artificial. Utilizó los caracteres del español e imitó la distribución de frecuencias de tamaños de palabras del mismo idioma.

Un punto a resaltar de la tabla es el tiempo total de indexado para los 1001 documentos, pues es de menos de una hora. En un estimado del tiempo de indexado para el sistema actual, el total de horas de procesamiento sería de 16,584.5. Este valor lo calculé a partir de los siguientes datos tomados del indexador en ejecución.

- Lanzamiento del indexador: 30/03/12 13:24
- Momento de la toma de los datos: 17/04/12 12:10
- Cantidad de archivos procesados: 26 archivos o 2.5%
- Tiempo de procesamiento: 430.766 [h] o 25,846 [min]

A partir de estos valores fue que calculé un aproximado del tiempo total de procesamiento. Para esto consideré el tiempo de procesamiento por archivo, operación mostrada en la Ecuación 1. Este resultado lo multipliqué por la cantidad de archivos totales, como se ve en la Ecuación 2. Este cálculo me pareció válido ya que al ser un sistema que analiza un archivo tras otro y todos los archivos contienen la misma cantidad de líneas, el tiempo de procesamiento para cada archivo consideré que debía ser muy parecido.

Ecuación 1

$$\frac{25846\ [min]}{26\ [archivos]} = 994.076923077\left[\frac{min}{archivos}\right]$$

Ecuación 2

$$(1001\ [archivos])994.076923\left[\frac{min}{archivos}\right] = 995071\ [min]$$

El tiempo total estimado de procesamiento en segundos para todos los documentos del corpus con el sistema propuesto se muestra en el resultado de la Ecuación 2. Comparando este tiempo con el tiempo del sistema actual, obtuve una reducción del 99.9973%. En este punto decidí estimar el comportamiento del sistema para corpus aún más grandes. Fue por

esto que grafiqué los resultados de la Tabla 5 en la Figura 48; para ver de una manera más clara el comportamiento del sistema propuesto.

De la gráfica se puede ver un comportamiento lineal en el crecimiento del tiempo de indexado por parte de la propuesta de esta tesis. Sin embargo, es probable que la función encontrada en ésta gráfica no sea tan aplicable para todos los casos ya que está dada para una cantidad de archivos que dependen de la cantidad de líneas de cada uno. Esta dependencia se ve más clara en el CEMC. En la Tabla 6 se observan los resultados obtenidos para el indexado con distintos tamaños del CEMC utilizando el sistema propuesto.

Número de archivos	Tiempo [s]	Tiempo [min]
2	0.24	0.00
487	34.00	0.57

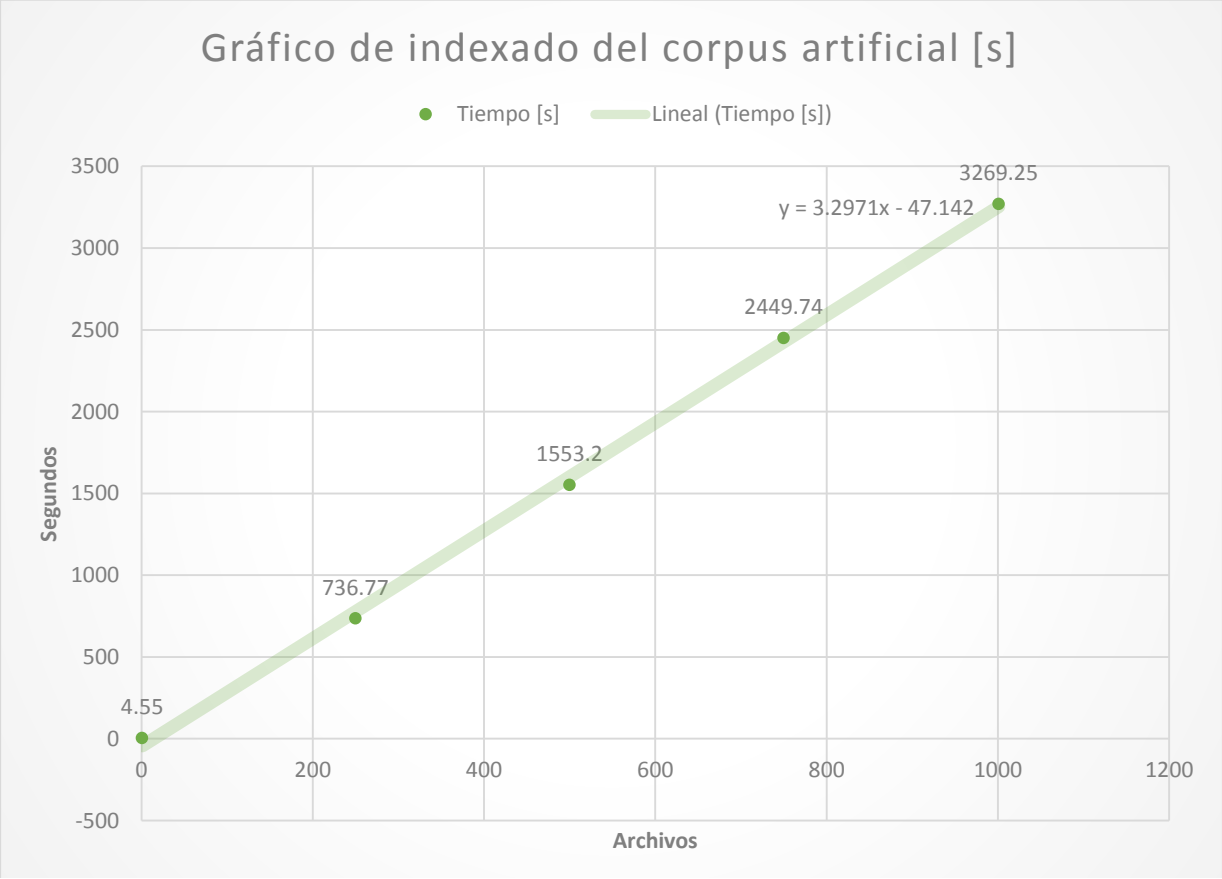


Figura 48. Gráfico de los resultados obtenidos en el indexado del corpus artificial con la nueva propuesta.

975	98.70	1.65
1462	142.14	2.37
1950	217.33	3.62

Tabla 6. Tiempo de indexado para distintos tamaños del CEMC.

De la Tabla 6 se observa entonces que el tiempo final de indexado para el 100% de los textos del corpus con el sistema propuesto es de 3.62 minutos. Cabe mencionar que con miras a una ejecución parecida a la del sistema actual, fue que decidí tomar estos tiempos sin el indexado de bigramas. En una prueba de indexado de unigramas y bigramas simultáneamente obtuve los siguientes valores: 4 minutos 1 segundo el indexado de bigramas y 3 minutos 30 segundos el indexado de unigramas, el tiempo total fue el tiempo de indexado de los bigramas.

Aunque la Tabla 6 también presenta un comportamiento lineal, como se ve en la gráfica de la Figura 49, el comportamiento en este caso es más errático que en el anterior. Por lo que

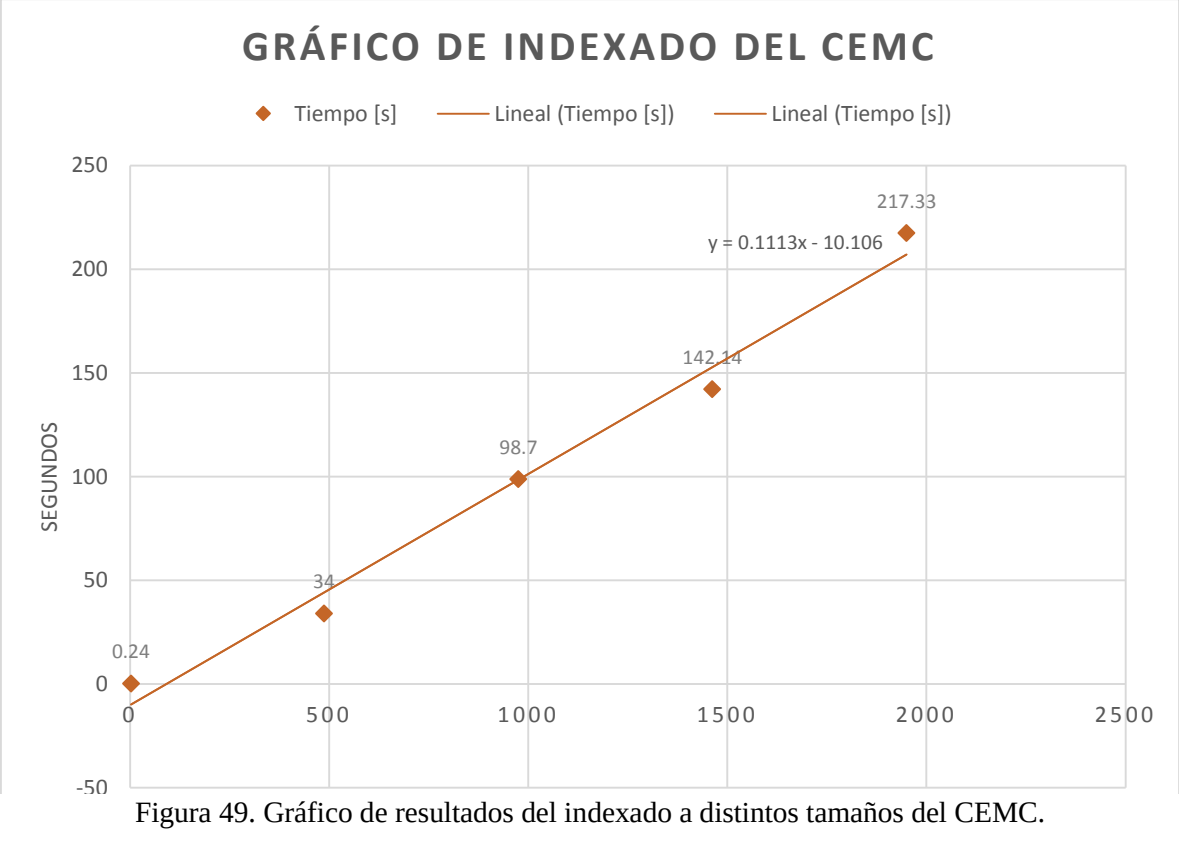


Figura 49. Gráfico de resultados del indexado a distintos tamaños del CEMC.

se hace más notorio que predecir el tiempo de indexado de esta propuesta va a estar dado por las líneas de texto que se vayan a analizar, no así por la cantidad de archivos del corpus.

Tomando en cuenta lo anterior, decidí realizar las mismas gráficas pero con la variante de la cantidad de líneas en lugar de la cantidad de archivos. Nótese que no estoy hablando de la cantidad de palabras, esto es porque debido al diseño del software también las líneas de estilo son consideradas dentro del tiempo de análisis del archivo.

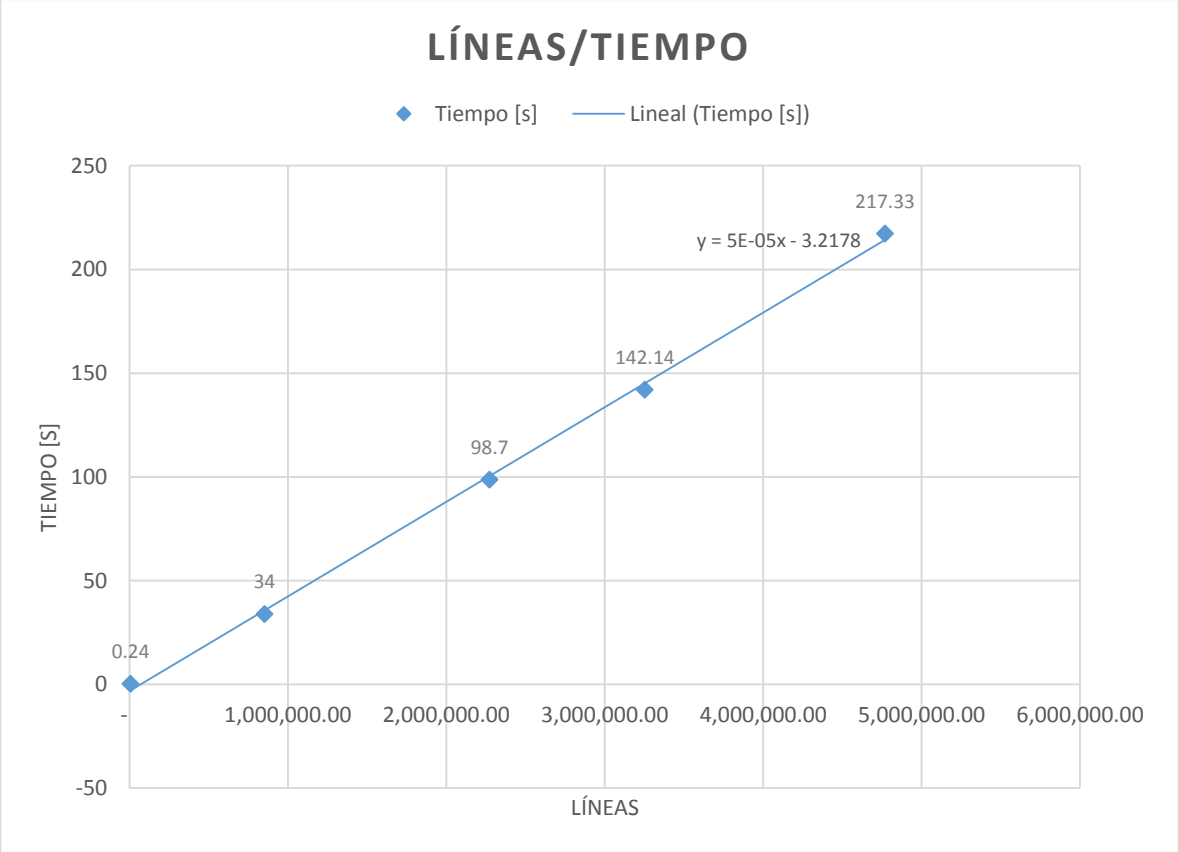


Figura 50. Gráfico de líneas/tiempo en el indexado del CEMC.

La Figura 50 muestra los tiempos obtenidos en la Tabla **6**, resultados dados para el indexado del CEMC con el sistema propuesto. Por su parte la Figura 51muestra los tiempos para el corpus artificial, también con la cantidad de aproximada de líneas en el eje X.

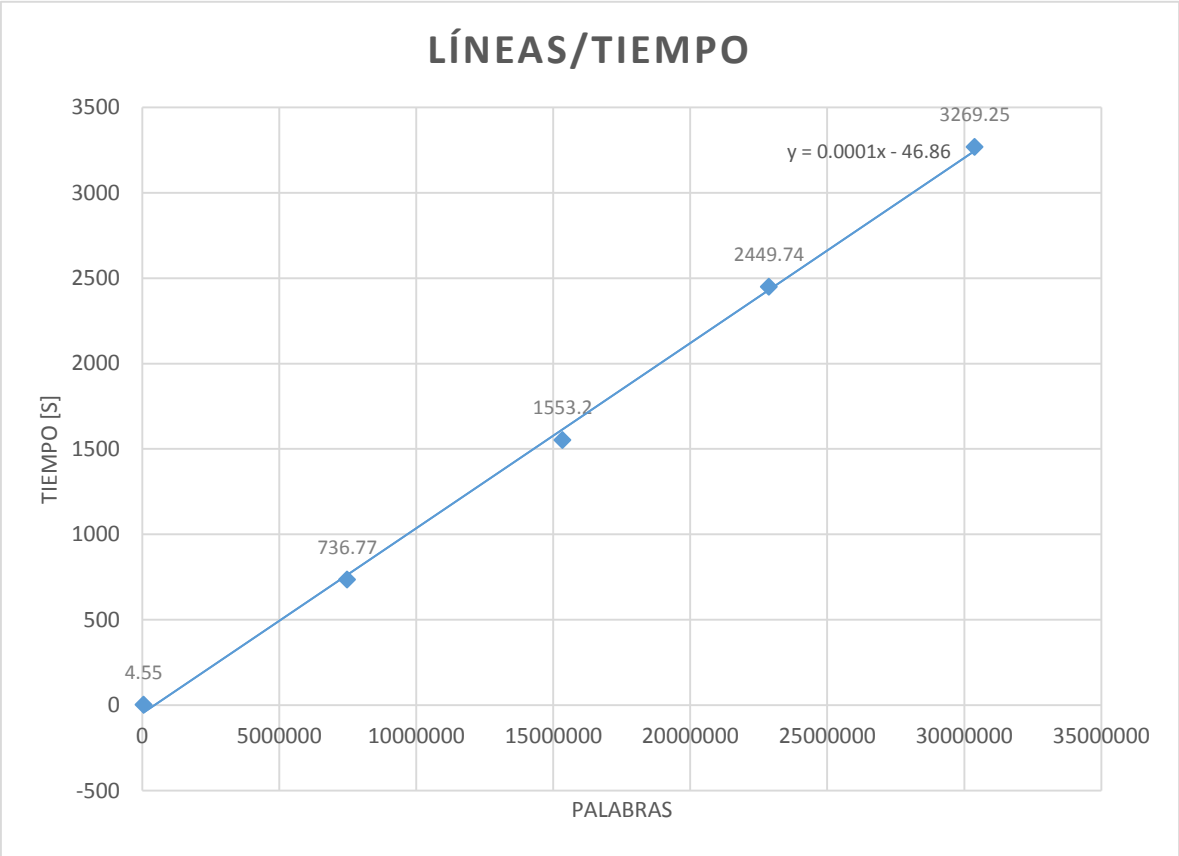


Figura 51. Gráfica de líneas/tiempo en el indexado del corpus artificial.

Con esto, podría aproximar un tiempo de indexado para un corpus de 100 millones de palabras en 165.885 minutos, o 2 horas y 45 minutos, de procesamiento continuo. Esto si se quisiera indexar esta cantidad de palabras en una sola exhibición. Con lo que el objetivo de tener un software capaz de soportar un corpus de gran tamaño, al menos para el indexado, se ha logrado.

Aunque no realicé el indexado del corpus artificial con la arquitectura actual, sí realicé un indexado de los distintos tamaños del CEMC. Este indexado, con la arquitectura actual, también resultó bastante interesante. Los resultados de estos experimentos los muestro en la gráfica de la Figura 52 y los valores de esta gráfica se pueden ver en la Tabla **7¡Error! No se encuentra el origen de la referencia.**.

Archivos	Tiempo [s]	Tiempo [min]
1	158	2.633
2	328	5.467
487	16059	267.650
975	15362	256.033
1462	15816	263.600
1950	26831	447.183

Tabla 7. Tiempos de indexado del CEMC con la arquitectura actual.

De la Figura 52 se puede observar claramente un comportamiento distinto al de la nueva propuesta. En el caso de la actual arquitectura el tiempo de indexado se estabiliza del 25% al 75%, aunque en el caso del 100% el tiempo se dispara a casi el doble de lo que venía mostrando. Estos datos se comprueban dentro de la Tabla **7¡Error! No se encuentra el origen de la referencia.**, en donde el tiempo de indexado entre el 25% del corpus (487 archivos) y el 75% varía por cuatro minutos. Siendo incluso el tiempo mayor para el 25% del corpus.

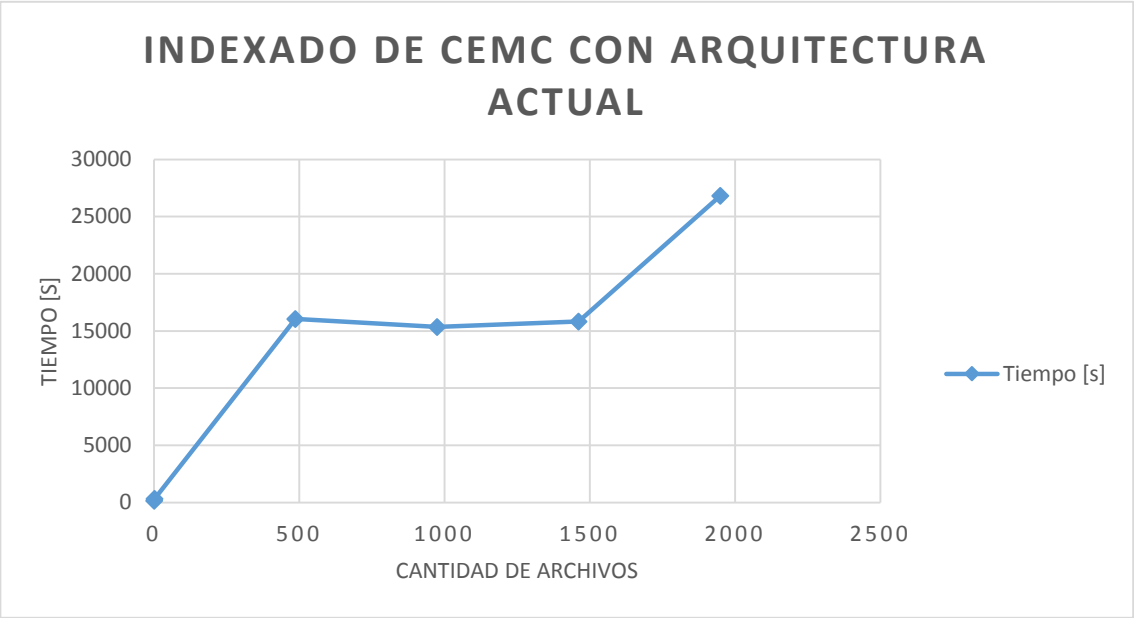


Figura 52. Gráfico de los tiempos de indexado del CEMC con la arquitectura actual.

A diferencia del indexado con la propuesta de esta tesis, el indexado con la arquitectura actual, no presenta un comportamiento lineal. Otra diferencia que noté entre ambos sistemas es el uso de los recursos del equipo anfitrión. La arquitectura actual no explota en ningún momento alguno de los núcleos del procesador ni ocupa una gran cantidad de espacio de la RAM.

En cuanto al indexado se refiere, la nueva propuesta mostró mejores resultados al indexar en un menor tiempo en todas las situaciones, incluso en el indexado de dos documentos. Aunque se crearon múltiples trazas de ejecución, los recursos del equipo no se vieron monopolizados o saturados por el sistema. Estos resultados fueron los obtenidos en el indexado, sin embargo, aún hace falta generar las concordancias. Este proceso es tan importante como el anterior, por lo que ambos sistemas fueron puestos a prueba.

En el caso de la arquitectura actual, los tiempos fueron tomados de un documento bitácora escrito por el mismo sistema. Por otra parte, el tiempo de la nueva propuesta fue tomado directamente del tiempo mostrado por el sistema en la terminal Linux en la que se estuvo

ejecutando. Ambos tiempos fueron tomados a partir del tiempo que el sistema recibe la consulta del usuario hasta que termina de imprimir los documentos XML. En estas pruebas se cuidó en todo momento que la consulta fuese la misma, esto implica no sólo a la palabra o lema buscado, sino también la ventana de palabras de la concordancia y el ordenamiento de los resultados.

En todos los casos en los que la nueva propuesta obtuvo resultados, el tiempo de respuesta fue menor que el de la arquitectura actual. Incluso en consultas con 8 concordancias y una ventana de 7 palabras, el tiempo de respuesta fue menor en la nueva propuesta. Todos estos tiempos se muestran en la Tabla **8¡Error! No se encuentra el origen de la referencia.**.

También en esta tabla se observan tiempos de respuesta para la nueva propuesta en 0 segundos, este tiempo es para los casos en los que la nueva propuesta no terminó el proceso de generación de concordancias correctamente. En estas consultas con más de cuatro mil resultados, el proceso se trunca en la obtención de las concordancias, esto en la nueva propuesta. Por su parte, la arquitectura actual sí termina la obtención del 100% de las concordancias, aunque no en un tiempo óptimo.

PALABRA	Resultados arquitectura actual	Resultados nueva propuesta	Tiempo arquitectura actual [s]	Tiempo nueva propuesta [s]	Diferencia en tiempos	Ganancia
dinero	615	614	15.321	7.3743	7.9467	51.87%
no	31705	31634	712.841	0	0	0.00%
bien	2616	2601	44.273	35.6164	8.6566	19.55%

profesor	128	127	6.11	2.69952	3.41048	55.82%
reloj	51	51	3.323	1.1872	2.1358	64.27%
[vivir][22]	1169	1167	26.429	17.15478	9.27422	35.09%
[ir][22]	13318	13287	171.047	0	0	0.00%
lápiz	25	0	2.304	0.000247	2.303753	99.99%
progreso	152	152	6.538	2.65547	3.88253	59.38%
[ver][22]	4992	4986	60.136	0	0	0.00%
diccionario	8	8	1.967	0.3818	1.5852	80.59%
Metros	219	219	5.749	4.20702	1.54198	26.82%

Tabla 8. Comparativo de tiempos de respuesta de la generación de concordancias.

En el caso de las consultas con una gran cantidad de resultados, el proceso se interrumpe en la generación de concordancias debido a una saturación de los recursos del equipo anfitrión. Esta saturación se da debido a la cantidad de hilos generados y que se encuentran en ejecución. Al estar todos en ese momento pidiendo tiempo del procesador se monopolizan y posteriormente se saturan los recursos del sistema. Por esto, es que el proceso se trunca y no se logra una recuperación total de las concordancias.

En el caso de la palabra *lápiz* la consulta no devolvió resultados debido a problemas con la codificación en que se está interpretando la consulta. Es por esto que no reconoce como consulta válida el uso de acentos y por tanto los resultados obtenidos son nulos en este caso. Otro problema generado por la codificación es la pérdida de concordancias. En la Tabla 8 se puede notar este problema en la cantidad de resultados de la nueva propuesta, siendo menor la cantidad de resultados obtenidos por la nueva propuesta que la arquitectura actual. Este problema se encuentra vinculado a la codificación de los textos en XML (UTF-8) y la codificación de Perl (ISO-8859-1), es por esto que no se están recuperando la totalidad de los registros dentro del CEMC.

[22] Recuérdese que los corchetes cuadrados [] indican que se busca el lema.

Finalmente, en la Figura 53 se observa el gráfico comparativo de los tiempos de respuesta entre los sistemas. En todo momento la línea del tiempo de la nueva propuesta se muestra por debajo de la arquitectura actual. Sin embargo, en los picos de la gráfica que son cuando las consultas fueron mayores a 4 mil resultados, la gráfica puede dar una falsa percepción de mejora. Esto porque, como ya lo mencioné previamente, en estas consultas la nueva propuesta no termino de escribir todos los archivos XML de salida. De modo que el proceso se truncó y el tiempo más bajo entonces le pertenece a la arquitectura actual.

Estos fueron todos los experimentos realizados a ambas arquitecturas, la nueva propuesta y la arquitectura actual. En el siguiente capítulo expondré las conclusiones de este trabajo, así como el trabajo a futuro, las ventajas y desventajas de la propuesta. También dentro de las conclusiones muestro un breve resumen de cada uno de los capítulos.

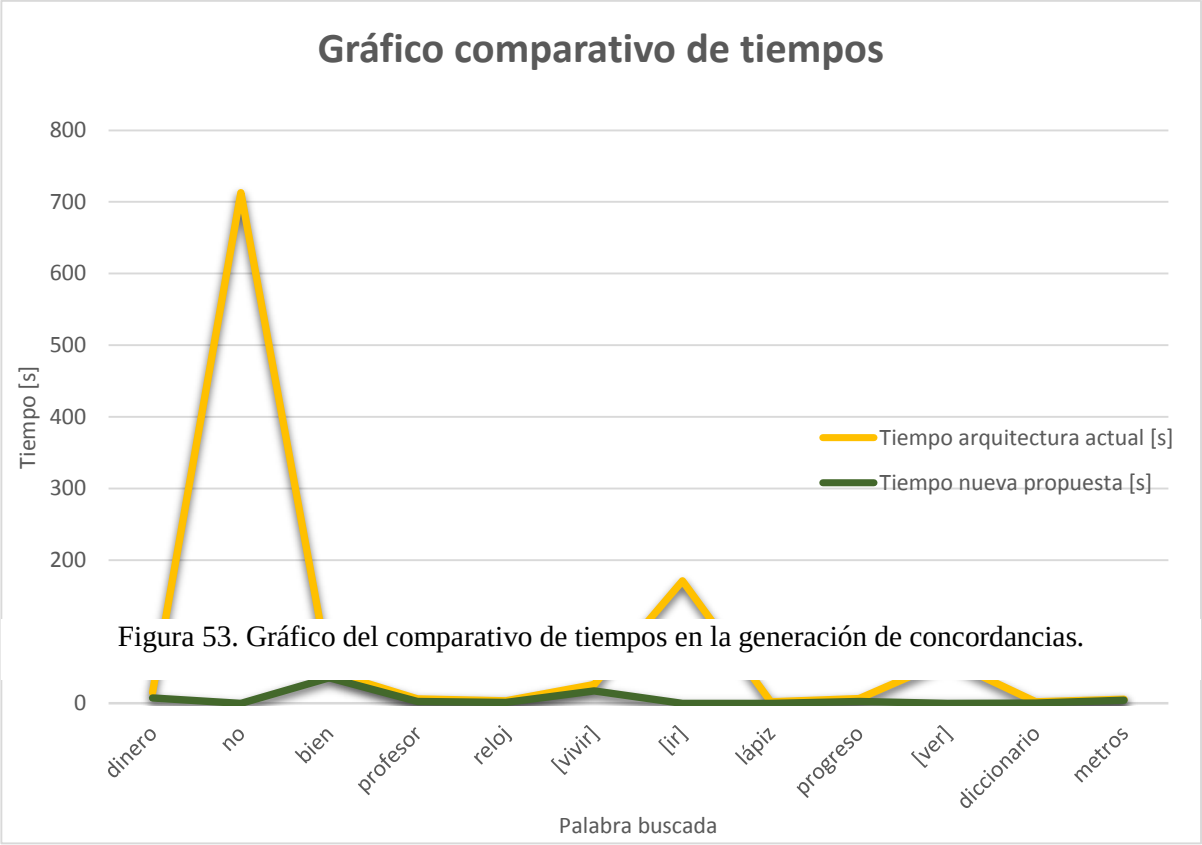


Figura 53. Gráfico del comparativo de tiempos en la generación de concordancias.

V. Conclusiones

A lo largo de esta tesis expuse, entre otras cosas, la investigación que realicé para obtener una propuesta para la generación de concordancias en los corpus del GIL, dando especial énfasis en la optimización de los procesos involucrados. Así, en la introducción expuse el planteamiento del problema, los objetivos perseguidos al inicio de este trabajo, la metodología que seguí y el alcance esperado de mi investigación.

Ya con una vista general del problema que dio origen a esta tesis, en el primer capítulo hablé sobre el marco teórico que consideré necesario para comprender de mejor manera mi trabajo. Dentro de este marco teórico expuse términos como el de procesamiento del lenguaje natural, corpus electrónico, concordancias, lema, etiquetado, entre otros.

Con estos y otros conceptos definidos, en el capítulo dos hablé sobre la arquitectura de los corpus electrónicos del GIL. Ya que dentro del alcance de esta tesis se encuentra la posibilidad de que la propuesta sea aplicada en más de uno de los corpus del grupo, en el capítulo dos también hablé sobre los corpus del GIL en los que podría ser aplicada esta propuesta. Hice énfasis en el CHEM y en el CEMC. El primero de estos dos corpus, tiene la arquitectura base sobre la que fue realizado el segundo. Es por esto que dentro del capítulo dos dediqué especial atención a este corpus.

Por otro lado, el CEMC es el corpus sobre el cuál se probaría mi propuesta una vez terminada. De modo que para realizar cualquier modificación, mejora o propuesta a la arquitectura de este corpus, había que conocer antes su arquitectura actual. Esta es la razón por la cual la mitad del segundo capítulo esté dedicado a describir esta arquitectura.

En el tercer capítulo expuse mi propuesta de optimización, su diseño, desarrollo y el porqué de mi elección. Dentro del diseño, describo el algoritmo que siguen las distintas partes

del motor de búsqueda propuesto. En el desarrollo explicó de manera más práctica cada uno de los componentes del motor de búsqueda, haciendo uso de fragmentos del código fuente para dejar más claras algunas de las funciones, métodos y subrutinas usadas para la generación de las concordancias. Antes del diseño, hay una sección de análisis preliminar en la cual discutí el porqué de algunas decisiones que tomé al realizar esta propuesta.

En el cuarto capítulo, presenté los resultados y la evaluación sobre los experimentos realizados para comprobar si la propuesta realizada era o no una opción viable para ser instalada en el CEMC o en cualquier otro corpus del GIL. Dentro de los análisis y pruebas, realicé experimentos sobre un corpus artificial del grupo, el cual contiene alrededor de 30 millones de palabras. De este experimento obtuve resultados que me ayudaron a ver mejor el comportamiento de la propuesta y su proyección para ser utilizada sobre corpus de gran tamaño.

Ahora expongo un resumen de los experimentos realizados. Para llevar a cabo los experimentos fue necesario buscar que estos fueran los más justos posibles con ambas arquitecturas, la actual y la propuesta. Terminado el desarrollo, el primer experimento consistió en indexar en su totalidad el CEMC con ambas arquitecturas. El resultado de este experimento fue una gran diferencia en tiempo. Por una parte, el indexador actual terminó el trabajo en poco menos de 7 horas y media, mientras que el indexador propuesto tuvo un tiempo de 3 minutos y 37 segundos. Una reducción en tiempo del 99.19%.

El tiempo no fue el único valor que evalué a partir del experimento anterior, también noté la sencillez y practicidad con la que cada uno de los sistemas se ejecuta. Si bien ambas propuestas (la actual y la nueva) pueden ser ejecutadas desde una terminal Linux, en la arquitectura actual se requiere de contar con una base de datos correctamente configurada y diseñada. Esto implica que el usuario invierta más tiempo en la configuración y puesta a punto del

sistema anfitrión para ejecutar el indexado que con la nueva propuesta. Con esta última, únicamente son necesarios los requerimientos señalados en la sección III.4 de esta tesis.

El indexar el corpus artificial del GIL como experimento tuvo mucho que ver con uno de los objetivos de este trabajo: que la arquitectura sea capaz de trabajar con corpus de gran tamaño. Fue por esto que la prueba que siguió al indexado del CEMC fue el indexado del corpus artificial.

De indexar el corpus artificial los resultados fueron:

- El indexado en sí mismo. Esto lo menciono porque con la arquitectura actual no se logró indexar correctamente y por completo este corpus dado su gran tamaño. En la nueva propuesta sí fue posible hacerlo.
- La reducción del tiempo de indexado. Como lo mencioné antes en esta tesis, el tiempo pronosticado de indexado para el corpus artificial con la actual arquitectura es de 1.8 años aproximadamente. Con la nueva propuesta, el tiempo real de indexado fue de 54 minutos.
- La facilidad de adaptación del indexador. Si bien es cierto que la estructura de los archivos XML de los documentos que conforman cada uno de los corpus es la misma, no todos los corpus usan las mismas etiquetas. Algunos corpus cuentan con más o menos etiquetas, tanto de información lingüística como de estilo. Por esta razón es necesario que el indexador sea capaz de adaptarse fácilmente a todos los corpus. Al indexar el corpus artificial verifiqué que debido al uso de expresiones regulares para las principales funciones de obtención de información, es que la adecuación de un corpus a otro no deberá representar ningún problema. En

otras palabras considero que es más fácil adaptar la nueva propuesta a otros corpus.

Las siguientes pruebas las realicé para buscar graficar el comportamiento de los indexadores de ambas propuestas en relación con el tiempo y el tamaño del corpus. Para graficar esto fue que indexé el CEMC con un 25%, 50%, 75% y 100% de su tamaño con ambos indexadores (actual y propuesto). De estas pruebas obtuve un registro del crecimiento del tiempo de indexado y una idea más clara del comportamiento de cada indexador. Además, de estos experimentos encontré que en el caso de la nueva propuesta, el crecimiento se dio de manera lineal y estuvo dado por la cantidad de líneas de un documento y no por la cantidad de documentos del corpus.

En el caso de la arquitectura actual, no pude identificar una función que encuadrara con los tiempos obtenidos. Con esto me refiero a que para indexar el 25% del CEMC, el programa se tardó más de lo que tardó en indexar el 50% y el 75%. Luego, para indexar el 100% del corpus el tiempo fue muy cercano a la suma del indexado del 75% más el 25%. De esta gráfica no logré inferir alguna función que me permitiera explicar el comportamiento del indexador, ya que incluso una aproximación polinomial distaba en mucho con los resultados reales.

La siguiente fase de los experimentos fue sobre la generación de las concordancias. De modo que realicé las mismas consultas con amabas arquitecturas, con las mismas variantes y con los mismos filtros. Para los resultados, tomé el tiempo de respuesta de la bitácora que se escribe de manera automática en el caso de la arquitectura actual. Para la nueva propuesta, tomé el tiempo que se tarda en terminar de generar los archivos XML de salida.

El tiempo de respuesta de la arquitectura actual lo tomé de la bitácora ya que ésta tiene implementada la generación de estadísticas dentro de la generación de las concordancias, así como la presentación en web de los resultados obtenidos. Estos dos procesos son adicionales a los que realiza la nueva propuesta. Sin embargo, comparar ambas propuestas resulta injusto pues varían en muchos aspectos, tales como:

- El lenguaje de programación utilizado. La propuesta nueva emplea Perl, mientras que la arquitectura actual usa Java.
- El paradigma de programación utilizado. La propuesta nueva usa una programación estructurada. Por su parte, la arquitectura actual usa una programación orientada a objetos.
- El enfoque para la organización de los datos. Por una parte, la arquitectura actual usa bases de datos relacionales y, por otra, la nueva propuesta usa archivos mediante índices invertidos.
- La cantidad de hilos. La nueva propuesta está pensada desde un principio en el uso de múltiples trazas de ejecución para agilizar la tarea. Esto es algo con lo que la actual arquitectura no cuenta.

Por estas y otras variantes entre las propuestas no me pareció válido decidir cuál de las dos es mejor; sin embargo, hay un indicador que sí puede ayudar a la selección de una de las dos como más conveniente: el tiempo de espera del usuario final. Y es que al final del día, es el usuario final quien decide si regresa a un sitio en línea de acuerdo al tiempo de respuesta, la precisión de los resultados y la gama de consultas posibles.

De realizar las consultas obtuve una mejora en el tiempo de respuesta en todos los casos. La menor ganancia fue de 1.5 segundos, obtenidos para una consulta con 8 concordancias y

una ventana de 7 palabras. Por otra parte, la mayor ganancia fue de 9 segundos para una consulta de dos mil concordancias y una ventana de 10 palabras. Sin embargo, la nueva propuesta presentó problemas para entregar resultados en consultas con cuatro mil o más concordancias. Este problema es debido a la cantidad de hilos generados por el programa, deduje esto al monitorear el estado del equipo anfitrión y notar una saturación de los recursos del equipo debida a la cantidad de hilos generados y que buscaban tiempo del procesador.

Mas en el caso de las consultas con comodines, la arquitectura actual se mostró superior en la sencillez con la que el sistema acepta este tipo de consultas. Con esto me refiero a que Lucy, para hacer el tratamiento de comodines, necesita de una programación más compleja que la que requiere hacer la consulta a PostgreSQL. Es por esto que la nueva propuesta aún no tiene funcional esta opción. Así, la nueva propuesta no resulta útil en este tipo de consultas.

En cuanto a los objetivos que se plantearon en un principio para esta tesis, se tenían:

- Tener una arquitectura capaz de indexar y trabajar sobre corpus de gran tamaño. En este caso, se logró indexar un corpus de 30 millones de palabras en una hora. Este corpus puede ser ya considerado como de gran tamaño, por lo que puedo concluir que la propuesta actual es capaz de indexar corpus de gran tamaño. Sin embargo, dada la composición del corpus, no se realizó ninguna consulta que generara concordancias. Así que si bien en teoría la nueva propuesta debe ser capaz de trabajar con este corpus, esto no se demostró en la práctica en la parte de la consulta.

- Tener un indexador incremental. Este objetivo se cumplió en un 100%. El sistema, gracias a Lucy, agrega sin problema alguno las nuevas entradas de los archivos que se deseen agregar a un indexado previo.
- Mejorar la velocidad de respuesta del generador de concordancias. Pese a que el objetivo se consiguió en algunos casos, la respuesta del sistema es deficiente y no podría decir que mejoró a la arquitectura actual. Esto es porque, si bien en todas las consultas en las que ambos sistemas obtuvieron resultados los tiempos de la nueva propuesta fueron mejores, también hubo consultas para las que esta propuesta no consiguió obtener resultados. Esto se debió a la cantidad de hilos lanzados para la generación de las concordancias pues arriba de las 4 mil concordancias, la nueva propuesta monopolizó los recursos y en consecuencia no consiguió generar concordancias. Esto contrastó con la arquitectura actual, en la que para el 100% de las consultas el sistema consiguió extraer las concordancias.
- Conseguir un mejor uso del hardware disponible. Con la implementación de múltiples trazas de ejecución dentro del indexador y del generador de concordancias, el sistema utiliza más recursos que el anterior. Con esto logra cumplir con su tarea en un menor tiempo, aunque para esto utiliza todos los núcleos del procesador y una mayor cantidad de memoria RAM. Si es o no un mejor uso del hardware, me parece una pregunta con una solución bastante subjetiva y que depende de la definición de un mejor uso. Lo que es cierto, es que el sistema actual utiliza una mayor cantidad de recursos y entrega, en muchos de los casos, resultados en un menor tiempo.

Por último, resta revisar el objetivo central de esta tesis: proponer una arquitectura distinta a la actual para los corpus lingüísticos electrónicos del GIL. Este objetivo se cumplió en un 100% ya que la arquitectura actual varía con respecto de la nueva propuesta desde cosas tan básicas como el lenguaje de programación hasta el paradigma de programación utilizado. Otro aspecto es el método de indexado, teniendo por un lado un enfoque de bases de datos relacionales en la arquitectura actual y por otro lado un enfoque de índices invertidos en la propuesta de esta tesis.

Claro que esta nueva propuesta tiene tanto desventajas como ventajas y a continuación haré un listado de ambas. Por una parte, el sistema presenta ciertas desventajas que listo a continuación.

- Poca compatibilidad con otros sistemas operativos. Pese a que Perl es un lenguaje multiplataforma como Java, instalar Lucy no es una tarea sencilla fuera de un sistema operativo Linux.
- Difícil implementación de la búsqueda con comodines. El tratamiento de las consultas con comodines es una opción no implementada por Lucy, por lo que es necesario un procesamiento extra para implementar este tipo de búsqueda. A diferencia de la facilidad con que PostgreSQL realiza este tipo de búsquedas sobre cualquiera de sus campos marcados como cadenas de texto.
- El enfoque de Lucy de recuperación de la información que obliga a realizar la recuperación de las concordancias a partir de los archivos y no de la carga de los mismos en memoria temporal. Esto provoca que para las concordancias con resultados arriba de los 4 mil resultados, el sistema se tan demandante de recursos que llega a bloquear el equipo anfitrión.

- El uso de todos los recursos del sistema. Debido a la planeación de crecimiento de los hilos, la nueva propuesta utiliza la mayor cantidad de recursos posible del equipo anfitrión. Aunque esto en algunas ocasiones ayuda y resulta conveniente para la extracción de resultados, en otras resulta un problema pues monopoliza el procesador e impide al equipo anfitrión realizar algún otro proceso.
- El procesamiento de caracteres especiales, como son: letras con acentos y diéresis, entre otros, son un punto débil dentro de esta propuesta. En el caso de los archivos del corpus en XML, su codificación es UTF-8 y para leer este tipo de archivos fue necesario hacer la transformación de las cadenas obtenidas. De lo contrario, el sistema tomaba las cadenas con una codificación ISO-8859-1, ocasionando una pérdida de datos y un indexado incorrecto. Pese a que el indexado se realiza correctamente, la recuperación de las concordancias no tiene los mismos resultados, impidiendo así la recuperación de concordancias de palabras como *lápiz* y *pingüino*.

Por otra parte, listo las ventajas que presenta la nueva propuesta.

- Reducción del tiempo de indexado. Como quedó consignado arriba, el tiempo de indexado de la nueva propuesta disminuyó considerablemente en relación a la arquitectura actual.
- Menor cantidad de fuentes de errores. En el caso del indexador actual, los errores en el indexado pueden venir de distintas fuentes, tales como la lectura de los archivos, la falta de paquetería requerida, la conexión con la base de datos, la interacción con la base de datos y los parámetros de entrada para ejecutar el indexador. En el otro caso, la arquitectura propuesta elimina la base de datos como fuente de

errores, dejando la lectura de los archivos y la falta de paquetería requerida como fuentes de errores.

- Simplicidad en la puesta a punto del sistema anfitrión. Si bien la nueva propuesta no requiere de una instalación como normalmente la conocemos, en la que se extraen archivos, se notifica de la nueva instalación al sistema operativo, etc., sí requiere de ciertos archivos que permitan su ejecución. Estos archivos son pocos y están todos concentrados en permitir la ejecución de un programa en Perl y de un programa en Lucy, sin necesidad de bases de datos o de un servidor dedicado como Apache Tomcat, como es el caso de la actual propuesta.
- Menor cantidad de parámetros de entrada para la ejecución del indexador. El único parámetro de entrada para el indexador es el directorio donde se encuentra el corpus a procesar.
- Facilidad de adecuación para distintos corpus. Esto lo comprobé al indexar el corpus artificial. Pues para realizar el indexado únicamente tuve que ajustar algunas expresiones regulares para acoplar el sistema a las etiquetas presentes dentro del corpus. El esquema también se puede modificar, siempre y cuando el corpus así lo requiera.
- Capacidad de integrarse completo o por partes con el sistema actual. Pese a que la salida global de la arquitectura son los archivos XML con las concordancias, también se puede dividir el sistema para integrar únicamente el indexador al sistema actual. Gracias a Lucy la salida del indexador son archivos JSON. Estos archivos se pueden leer por el equivalente de Lucy en java Lucene y así mantener gran parte del generador de concordancias actual. Desarrollado también en JAVA.

Sin lugar a dudas, aún queda mucho trabajo por delante, muchos puntos que pueden ser afinados para que esta propuesta pueda llegar a ser un hecho dentro del GIL. Como parte de este trabajo a futuro se encuentra una mejor planeación dentro de la generación de hilos para la recuperación de las concordancias. El problema de la monopolización de los recursos es sin lugar a dudas un punto débil muy importante dentro de esta propuesta, que se puede mejorar mediante un control de natalidad sobre la cantidad de hilos que son ejecutados por el sistema.

La inclusión de todas las posibilidades de consulta actuales es otro punto que dentro de este desarrollo quedó simplemente expresado como posible. Pero que no se implementó y ha quedado como trabajo a futuro. Sin embargo, es el desempeño del sistema para consultas de más de 4 mil resultados lo que resulta indispensable solucionar antes de implementar más opciones de consulta. Así también, el correcto tratamiento de caracteres especiales es un trabajo que se deberá realizar antes de poner en línea esta propuesta.

La unión de la propuesta actual con la interfaz web es otro punto que aún queda pendiente, ya que aunque desde un principio hablé sobre el alcance de este trabajo y no incluí en él esta implementación, esto no quiere decir que no se planee realizar este paso en un futuro para los corpus del GIL. De hecho, el GIL planea desarrollar un nuevo corpus para el *Seminario Universitario de Estudios del Discurso Forense*, especializado en documentos sobre *trata de personas*, haciendo uso de la propuesta de esta tesis.

Finalmente, quisiera resaltar que no hay que perder de vista en ningún momento que esta propuesta es tan solo una opción más que puede ser utilizada por el GIL, mas no es la única

opción. Así también, todo el código y el algoritmo propuesto son susceptibles de ser mejorados. Sin embargo la propuesta aquí encontrada dio muy buenos resultados al ser instalada y aplicada dentro del CEMC.

Referencias

About the Collins Corpus and the Bank of English. (2012). Recuperado el 18 de septiembre de 2012, de http://www.mycobuild.com/about-collins-corpus.aspx

Allen, J. (2003). Natural Language Processing [Versión Electrónica]. En *Encyclopedia of Computer Science 4th ed*. (pp. 1218-1222). Reino Unido: John Wiley and Sons Ltd. Chichester.

Barceló, M. (2001). *Inteligencia Artificial*. Editorial UOC.

Cabré, T. y Bach, C. (2004). El corpus tècnic del IULA: corpus textual especializado plurilingüe [Versión electrónica]. En *Panace@*, V, 173-176.

Cachero C., Ponce de León, P. y Saquete E. (2006). Introducción a la programación orientada a objetos [Versión electrónica]. Publicaciones de la Universidad de Alicante.

Da Cunha, I., Torres, J., Sierra, G- (2011). On the Development of the RST Spanish Treebank. En *Proceedings of the 5th Linguistic Annotation Workshop. 49th Annual Meeting of the Association for Computational Linguistics* (ACL). Portland, Oregon, USA.

Davies, M. (2002-). *Corpus del Español: 100 million words, 1300s-1900s*. Recuperado el 20 de septiembre de 2012, de http://www.corpusdelespanol.org/x.asp

Davies, M. (2002). Un corpus anotado de 100.000.000 palabras del español histórico y moderno [Versión Electrónica]. En *Procesamiento del Lenguaje Natural*, 29, 21-27.

Davies, M. (2008-). *The Corpus of Contemporary American English: 450 million words, 1990-present*. Recuperado el 19 de septiembre de 2012, de http://corpus.byu.edu/coca/

Davies, M. y Ferreira M. (2006-). *Corpus do Português: 45 million words, 1300s-1900s*. Recuperado el 24 de septiembre de 2012, de http://www.corpusdoportugues.org/x.asp

Davies, M. (s.f.). *Relational n-gram databases as a basis for unlimited annotation on large corpora*. Illinois State University.

Diccionario del Español de México (DEM), El Colegio de México, A.C., Recuperado el 23 de septiembre de 2013 de http://dem.colmex.mx

Evert, S. (s.f.). *Inside the IMS Corpus Workbench.* Institute of Cognitive Science, University of Osnabrück.

Fitzgerald, M. (2012). *Introducing Regular Expressions Oreilly and Associate Series*.O'Reilly Media, Inc.

Gelbukh, A. (2010). *Procesamiento del lenguaje natural.* Recuperado el 16 de abril de 2013, de http://www.gelbukh.com/CV/Publications/2010/Procesamiento%20de%20lenguaje%20natural%20-%20BORRADOR.pdf

Gómez, R. (2005). *La lematización en español: una aplicación para la recuperación de*

información. Ediciones Trea.

Hilera, J., Bengochea, L., Sánchez, R., Gutiérrez, J. y Martínez, J. (2005). Aplicación de técnicas de Ingeniería Lingüística en sistemas de e-learning basados en objetos de aprendizaje. *II Simposio Pluridisciplinar sobre Diseño, Evaluación y Descripción de Contenidos Educativos Reutilizables (SPDECE)*, Barcelona, España.

Ingeniería lingüística. (s.f.). Recuperado el 9 de noviembre de 2013 de http://sunsite.dcc.uchile.cl/~abassi/WWW/Lengua/ingenieria.html#t

Joshi, A. (1991). Natural language processing. *Science*, *Volumen* (253), 1242-1249. DOI: 10.1126/science.253.5025.1242

Lara, L. (2006). *Curso de lexicología*. Distrito Federal: El Colegio de México.

Lenguaje: ¿Qué es Lenguaje? Tipos de Lenguaje, Características, Definición, Concepto (s.f.). Recuperado el 21 de julio de 2014, de http://www.queeslenguaje.info

Liddy, E. (2001). Natural Language Processing. En *Encyclopedioa of Library and Information Science 2nd ed*. Estados Unidos: Marcel Decker, Inc.

MacCartney, B. (s. f.). *The Stanford Parser: A statistical parser*. Recuperado el 20 de junio de 2013, de http://nlp.stanford.edu/software/lex-parser.shtml

McEnery, T. y Wilson, C. (2001). *Corpus Linguistics: An Introduction*. Reino Unido: Edinburgh University Press.

Medina, A. y Méndez, C. (2006). Arquitectura del Corpus Histórico del Español en México (CHEM). En Hernández A. y Zenchinelli J. (Eds.). *Avances en la ciencia de la computación* (pp. 248-253). México: Sociedad Mexicana de Ciencia de la Computación.

Méndez, C. (2009). *Identificación automática de categorías gramaticales en español del siglo XVI*. Trabajo de grado presentado como requisito parcial para optar al Título de Maestro en Lingüística Hispánica. Universidad Nacional Autónoma de México. Ciudad Universitaria, México.

Mitkov, R. (2004). *The Oxford Handbook of Computational Linguistics* [Versión electrónica]. Oxford University Press.

Monz, C. y de Rijke, M. (2002). Inverted Index Construction. Introducction to Information Retrieval. Recuperado el 28 de abril de 2014, de http://www.n3labs.com/pdf/make-inverted.pdf

Moreno, A. (2009). Panorama actual de la ingeniería lingüística. En A. Alcina, E. Valero & E. Rambla (Eds.), *Terminología y sociedad del conocimiento* (pp. 99-116). Alemania: Peter Lang.

Oller, J. (2003). Elementos teórico-prácticos útiles para comprender el uso de los motores de búsqueda en Internet. *ACIMED* [versión electrónica]. *Volumen 11*, pp. 0-0.

Procházková, P. (2006). *Fundamentos de la lingüística de corpus. Concepción de los corpus y métodos de investigación con corpus* (T. Ramírez, Trads.). Recuperado el 19 de julio de 2012, de http://www.prochazkova.de/fundamentos_de_la_ling%C3%BC%C3%ADstica_de_corpus.pdf

REAL ACADEMIA ESPAÑOLA: Banco de datos (CREA) [en línea]. *Corpus de referencia del*

español actual. Recuperado el 24 de septiembre de 2012, de http://corpus.rae.es/ayuda_c.htm

Rojo, G. (2002). *Sobre la Lingüística basada en el análisis de corpus*. Recuperado el 21 de agosto de 2012, de http://www.uzei.com/Modulos/UsuariosFtp/Conexion/archivos54A.pdf

Santillán, M. (2012). *Ingeniería lingüística: un área innovadora*. Recuperado el 26 de septiembre de 2013, de http://ciencia.unam.mx/contenido/texto/leer/86#Trabajo mexicano en ingeniería lingüística

Sierra, G., Reyes, A., Antonio, C., Morett, S., Fonseca, C. y Baca, I. (2004, noviembre). Plan de desarrollo del grupo de ingeniería lingüística 1999-2004. México, D.F.: Universidad Nacional Autónoma de México, Grupo de Ingeniería Lingüística.

Sierra, G. (2008). Diseño de corpus textuales para fines lingüísticos. *IX Encuentro Internacional de Lingüística en el Noroeste*, Tomo 2, pp 445-462.

Sinclair, J. (1996) Preliminary Recommendations on Corpus Typology. EAGLES Document EAG-TCWG-CTYP/P, Mayo 1996. Recuperado el 19 de septiembre de 2013, de http://www.ilc.cnr.it/EAGLES96/corpustyp/corpustyp.html

Spyns, P. (1996). Natural Language Processing. *Methods of information in medicine*, 35(4), 285-301.

The Lancaster Stemming Algorithm (s.f.). Recuperado el 12 de junio de 2013, de http://www.comp.lancs.ac.uk/computing/research/stemming/index.htm

Universidad Nacional Autónoma de México. (2010). *Corpus de Contextos Definitorios (CORCODE)*. http://www.iling.unam.mx/corcode, Recuperado el 2 de diciembre de 2013.

Uszkoreit, H. (s. f.). Language technology a first overview. Recuperado el 17 de junio de 2013, de http://www.dfki.de/~hansu/LT.pdf

Vargas, J. (2013). *Optimización del generador de concordancias del Corpus Histórico del Español en México (CHEM) mediante una comparación entre manejadores de bases de datos relacionales y administración de desempeño de bases de datos*. Tesis de licenciatura. México: Universidad Nacional Autónoma de México.

What is a search engine? (2001). Recuperado el 12 de mayo de 2014, de http://designhammer.com/services/seo-guide/search-engines

What is the BNC? (2009). Recuperado el 19 de septiembre de 2012, de http://www.natcorp.ox.ac.uk/corpus/index.xml

Zobel, J. y Moffat, A. (2006). Inverted Files for Text Search Engines [Versión electrónica]. *ACM Comput. Surv., Vol. 38*, 1-56.