UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Extracción de secuencias maximales de una colección de textos

Proyecto final de carrera de
Ingeniería Técnica en Informática de Gestión

Sandra García Blasco

Dirigido por:

Dr. Paolo Rosso
Dra. Roxana Danger Mercaderes

Diciembre 2009

A mis padres, a mi hermana Marta y a Fran,
por su cariño y apoyo, y por creer en mí.

A Santiago,
por su apoyo y ayuda incondicionales.

A Paolo y Roxana,
por introducirme en el mundo del Procesamiento del Lenguaje Natural
y por ser mi guía durante todas las etapas del proyecto.

A Alberto,
por sus ideas y su ayuda en la definición de los experimentos a realizar.

Resumen

El objetivo de este proyecto final de carrera es la definición e implementación de un método para comparar la similitud entre varios textos. Para ello, se ha implementado un algoritmo capaz de detectar las secuencias maximales que son frecuentes en una colección de textos dada.

Para conseguir nuestro objetivo hemos implementado un algoritmo incremental, basado en *pattern-growing* (crecimiento de patrones). Este algoritmo es capaz de encontrar las secuencias idénticas de palabras que se repiten β veces en la colección de documentos, siendo β un umbral de frecuencia determinado por el usuario. Además, le hemos añadido la posibilidad de definir un segundo umbral, de capacidad de salto, mediante el cual el usuario indica cuantas palabras podemos ignorar entre dos palabras que se tienen en cuenta para la secuencia. Esto es, dada la secuencia: A B C 1 2 D, podría encontrar la secuencia ABCD si el umbral para la capacidad de salto es mayor o igual a 2.

La característica de la capacidad de salto es interesante sobre todo si se desea utilizar el programa para la detección de plagio, puesto que es capaz de detectar frases iguales aún en el caso de que se hubiesen insertado palabras nuevas dentro de la frase para intentar despistar a otros sistemas comparadores menos precisos.

La aplicación de este programa es útil para el análisis de grandes colecciones de textos dado que es capaz de encontrar patrones que se repitan en varios de los documentos. También es muy interesante su aplicación para el análisis de similitud entre textos e incluso, para la atribución de autoría de un determinado texto.

Una vez construido el algoritmo, se realizaron dos experimentos con el mismo, para demostrar sus capacidades. En el primero de ellos, se estudió un corpus de revisiones de artículos de la Wikipedia, comparando de un mismo artículo, las diferentes revisiones que ha sufrido a lo largo del tiempo con la versión actual. Esto nos permite determinar qué cantidad de conocimiento incluido en cada una de las revisiones se conserva en el artículo actual. En el segundo experimento diseñado pusimos a prueba la capacidad de salto del algoritmo. A partir de un documento, se generaron una serie de documentos derivados del primero, incluyendo en cada uno de ellos diferentes palabras intercaladas. Al aplicar el algoritmo se demostró que es capaz de encontrar la similitud entre revisiones de un mismo texto, a pesar de las palabras intercaladas que fueron introducidas.

Índice

Tabla Ilustraciones

1. Introducción

Durante las últimas décadas, la mayoría de los organismos tanto públicos como privados, han pasado a administrar sus documentos y gestiones de forma electrónica. Cada vez más cantidad de datos se almacena digitalmente, tanto si son datos estructurados como si no lo son.

El volumen de datos almacenados crece muy rápidamente, casi de forma exponencial, debido a la informatización de los procesos, los avances en bases de datos y lo barato que es almacenar los datos. La extracción de información de estas bases de datos es imprescindible, de modo que sea posible analizarlos y explotarlos de forma eficaz para los objetivos de cualquier organización. Muchas veces estos datos almacenados pueden proporcionarnos más información de la que se esperaba de ellos. La era de la información nos ha dejado con un problema, y es el de la sobreinformación. Cuando la cantidad de información es tan grande, es imposible para los humanos poder deducir y extraer datos útiles y conocimiento a partir de ella.

1.1. Motivación

Este proyecto surge por la necesidad de comparar grandes cantidades de texto, para detectar la similitud entre los distintos textos. La similitud vendrá definida por la cantidad de frases o secuencias iguales que aparecen en ambos textos, así como la longitud de éstas.

También surge con la intención de proponer una base para un nuevo método para la detección de plagio. Se conoce como plagio el acto de usar las ideas de otras personas sin citarlas, atribuyéndose la autoría el plagiador (Barrón Cedeño & Rosso, 2008). Un método básico para la detección de plagio es buscar palabras o cadenas repetidas. Dicho método puede ser muy limitado en algunos casos, o muy costoso en otros.

La escasez de algoritmos que efectúen esta comparación y búsqueda, y la limitación de los existentes, nos ha llevado a la implementación de uno de ellos, aplicando mejoras. De este modo, nuestro algoritmo no solo será capaz de encontrar las secuencias maximales que se repiten con cierta frecuencia, sino que podremos añadirle cierta flexibilidad y permitir que deje huecos entre palabras de la secuencia, detectando no sólo secuencias exactas, sino también “parecidas”.

1.2 Estado del arte

Minería de datos

La minería de datos se define como el descubrimiento de conocimiento, a partir patrones observables de datos estructurados, en bases de datos relacionales. Se le denomina comúnmente Knowledge-Discovery in Databases (KDD).

KDD (Fayyad, 1996), se refiere al "proceso de descubrir conocimiento útil a partir de datos". El objetivo de KDD es extraer conocimiento de alto nivel a partir de datos de bajo nivel en el contexto de grandes colecciones de datos. El conocimiento de alto nivel adquirido suele ser más compacto, más abstracto y más útil. Se trata pues, del hallazgo en grandes bases de datos de información que en principio no es obvia.

KDD consiste en varios pasos: preparación de los datos, selección de los datos, limpieza de los datos, incorporación de conocimiento a priori, minería de datos y una correcta interpretación de los resultados obtenidos. La clave en el proceso de descubrimiento del conocimiento es el paso de data mining (minería de datos), que según (Fayyad, 1996): "consiste en aplicar ciertos algoritmos y análisis a los datos que, con un coste computacional aceptable, produzcan una enumeración particular de patrones sobre los datos". Así por ejemplo, si nuestra colección de datos es la de una librería, donde tenemos almacenados de cada usuario los libros que ha comprado, podríamos encontrar patrones del tipo: clientes que han comprado "Los pilares de la Tierra", han comprado también "La catedral del Mar" y "La sombra del viento". Estos patrones, desde luego, aportan información extra a los gestores de la librería. Se ha sacado información de los datos, que en un principio no era obvia, y ahora se pueden tomar medidas como colocar estos tres libros en la misma estantería o enviar e-mails recomendándolos a clientes que hayan comprado uno de los tres. Nótese que en este caso, no es necesario que los tres libros se compren en el mismo orden.

Las técnicas de minería de datos se aplican sobre información contenida en almacenes de datos. No obstante, actualmente está cobrando mayor importancia la información contenida en datos no estructurados, como colecciones de textos. Es aquí donde aparece la minería textual.

Minería de textos

La minería textual está orientada a la extracción de conocimiento a partir de datos en lenguaje natural no estructurados, almacenados en bases de datos textuales. Se identifica con el descubrimiento de conocimiento en los textos y se le denomina comúnmente Knowledge-Discovery in Text (KDT).

La minería textual es una herramienta de análisis que permite extraer información de una colección de textos. La información que se busca no aparece de forma explícita en ninguno de los textos, sino que es información que se obtiene al comparar y procesar los textos de manera conjunta. El procesamiento de grandes colecciones de texto no estructurado para extraer conocimiento requiere aplicar técnicas tales como la identificación y extracción de patrones, el análisis de *clustering* (no se conocen a priori las clases o categorías) o clasificación (clases definidas a priori), etc.

El proceso de la minería de texto consiste en dos etapas principales: una etapa de pre-procesamiento y una etapa de descubrimiento (Tan, 1999). En la primera etapa los textos se transforman a algún tipo de representación estructurada o semi-estructurada que facilite su posterior análisis. En la segunda etapa se analiza la representación obtenida en la primera etapa con el objetivo de descubrir en ellas patrones interesantes que aporten nuevos conocimientos.

Las estructuras intermedias encontradas en la primera fase del proceso deben ser lo suficientemente sencillas para que sea fácil su procesamiento, pero también lo suficientemente completas para que aporten toda la información necesaria para el descubrimiento de los patrones que nos interesan. Para la segunda etapa es importante hacer un especial hincapié en que el coste computacional de los algoritmos de descubrimiento debe ser aceptable, puesto que se aplican sobre grandes cantidades de información.

Los algoritmos de *Sequential Pattern Mining* afrontan el problema de descubrir las secuencias frecuentes en una colección de textos dada. Este tipo de algoritmos son relevantes cuando los datos que debemos minar tiene una naturaleza secuencial, por ejemplo cuando se trata de palabras que forman frases, es decir, que las secuencias a buscar son un conjunto ordenado de elementos.

Sea S una colección de documentos, donde cada documento consiste en una secuencia de palabras.

Para entender qué son las secuencias frecuentes maximales, debemos tener en cuenta las siguientes definiciones:

Definición: Dada una secuencia $p = a_1 \dots a_k$ y una secuencia q tal que todos los elementos a_i aparecen en q y lo hacen en el mismo orden en el que aparecen en p, decimos que p es sub-secuencia de q.

Definición: Una secuencia p es frecuente en S si p es una sub-secuencia de al menos β documentos en S. Donde β es un umbral dado. En ese caso decimos que p es una secuencia β-frecuente, o simplificamos diciendo que p es frecuente.

Sólo se contará una ocurrencia por documento. El hecho de que una secuencia aparezca varias veces en un mismo documento, no hará aumentar su frecuencia.

Definición: Una secuencia *p* es una secuencia frecuente maximal en *S* si *p* es frecuente en *S* y no existe ninguna secuencia *p'* en *S* tal que *p* es sub-secuencia de *p'* y *p'* es frecuente en *S*.

Durante la última década se han presentado varias técnicas y algoritmos de descubrimiento de patrones en colecciones de textos. Las dos principales aproximaciones son las basadas en métodos a priori y las basadas en crecimiento de patrones (*pattern-growth*). Los algoritmos más conocidos en éste ámbito son *Generalized Sequential Pattern algorithm* (GSP) y *PrefixSpan*, y representan las dos aproximaciones al problema: a priori y *pattern-growth*, respectivamente. En (Oliveira & Antunes, 2004) se analizan las dos vertientes, tal y como se explica a continuación.

A Priori

Los métodos a priori siguen la filosofía de generación de candidatos y testeo de los mismos. En general, se pueden ver como algoritmos de búsqueda por anchura, puesto que construyen todas las k-secuencias para un determinado k de forma simultánea.

GSP sigue esta filosofía. Empieza con el descubrimiento de secuencias frecuentes de longitud 1 (1-secuencias), y va generando el conjunto de (k+1)-secuencias potencialmente frecuentes a partir del conjunto de k-secuencias frecuentes (llamadas candidatos). Cada generación de k-secuencias potencialmente frecuentes (k candidatos), usa las (k-1)-secuencias frecuentes sacadas en el paso anterior. De este modo se reduce considerablemente el número de secuencias que se tendrán en cuenta en cada momento. Hay que tener presente que para decidir si una secuencia *s* es frecuente o no, se deberá escanear la base de datos entera, confirmando que *s* se repite al menos β veces, siendo β el umbral de frecuencia mínima fijado por el usuario.

Como se puede deducir fácilmente, el tiempo de proceso del algoritmo sería muy elevado, por lo que GSP adopta tres optimizaciones. En primer lugar, se mantienen todos los candidatos en un *hash-tree* para escanear la base de datos una sola vez por iteración. En segundo lugar, sólo se crea un nuevo k-candidato cuando existen dos (k-1)-secuencias tales que, el prefijo de la una sea igual al sufijo de la otra. En tercer lugar, se eliminan todos los candidatos que contienen alguna sub-secuencia que no sea frecuente.

Con estas estrategias, GSP reduce el tiempo gastado en recorrido de la base de datos, incrementando así su eficiencia.

Pattern-growth (crecimiento de patrones)

Los métodos de crecimiento de patrones son una propuesta más actual para los problemas de minería de patrones de secuencias. La idea principal es evitar la generación de candidatos, y restringir la búsqueda en una porción reducida de la base de datos inicial. En general, los métodos de crecimiento de patrones se pueden ver como un algoritmo de búsqueda en profundidad, ya que construyen cada patrón por separado y de un modo recursivo. *PrefixSpan* es el ejemplo más conocido de este tipo de algoritmos.

PrefixSpan va construyendo los patrones mientras que restringe la búsqueda a la base de datos proyectados. Una base de datos α-proyectada es el conjunto de sub-secuencias en la base de datos que son sufijos de las secuencias que tienen α como prefijo. En cada paso, el algoritmo busca la secuencias frecuentes con prefijo α, en la correspondiente base de datos proyectada. El mérito de *PrefixSpan* viene dado por la proyección que hace del conjunto original de datos en subconjuntos cada vez más pequeños, de donde los patrones se pueden ir sacando progresivamente.

A diferencia de los métodos a priori, los métodos basados en *pattern-growth* no necesitan generar candidatos y luego comprobar si son frecuentes en la base de datos, sino que hacen crecer los patrones a partir de ítems frecuentes, de modo que son más eficientes. No obstante, es común que los métodos basados en *pattern-growth* proyecten los datos en conjuntos que se solapan, es decir, es muy probable que estos métodos estudien la misma parte de los datos varias veces.

Los métodos a priori generan muchas secuencias que ni siquiera aparecerán en la base de datos. Este hecho contribuye también a que la eficiencia en los métodos basados en crecimiento de patrones sea mayor.

En este proyecto final de carrera, nos centraremos en la extracción de secuencias maximales frecuentes dada una colección de textos no estructurados. Así pues, buscaremos patrones o sub-secuencias que se repitan en β documentos, siendo β un umbral puesto por el usuario. Estas secuencias serán, por tanto β-frecuentes. Además, para que sean maximales, no serán sub-secuencia de ninguna otra secuencia frecuente. Puesto que tratamos con textos largos escritos en lenguaje natural y de temática diversa, la generación de candidatos se haría muy costosa, debido a la explosión combinatoria. Es por eso que utilizaremos un método basado en crecimiento de patrones (*pattern-growth*).

Secuencias frecuentes maximales

Las secuencias maximales pueden ser de gran utilidad, puesto que podían representar las partes más importantes de los textos. Dada una colección de textos, el hecho de que algunas secuencias se repitan en varios de ellos, nos señala la importancia de la información que contienen dichas secuencias. También hay que destacar que la aplicabilidad de la extracción de secuencias frecuentes maximales es muy amplia puesto que la técnica es independiente del dominio del problema y del lenguaje en el que estén escritos los textos.

Un ejemplo muy interesante de esta independencia del dominio del problema es la detección de similitudes en cadenas de ADN o proteínas. Las cadenas de ADN son representadas por secuencias de caracteres pertenecientes a un alfabeto compuesto por 4 letras: A, G, C y T. La comparación entre secuencias de ADN descubrirá fracciones maximales de estas secuencias que se repitan un determinado número de veces. De este modo, se puede utilizar para detección de enfermedades, o para comparar ADN de distintas especies y definir similitudes entre ellas que puedan dar pie a nuevas investigaciones biológicas. En (Becher, Deymonnaz, & Heiber, 2008) se propone el algoritmo *findpat*, como una herramienta útil para encontrar repeticiones dentro de secuencias de ADN.

Si estamos procesando colecciones de textos, las secuencias frecuentes maximales obtenidas se pueden utilizar como una descripción de los textos. De hecho, al estar escritas en lenguaje natural, estas secuencias maximales obtenidas se pueden utilizar para generar automáticamente resúmenes de los textos a partir de ellas (Ahonen-Myka, 2002). Las secuencias maximales largas suelen ser muy descriptivas, y al mismo tiempo son representaciones compactas del texto.

En (Ahonen-Myka, 2002) proponen un algoritmo de búsqueda de secuencias frecuentes maximales donde tienen en cuenta más información que la palabra en sí. En este caso particular, de cada palabra almacenan un vector de características, que serán lo que luego se comparará, haciendo la búsqueda más precisa. La información que almacenan en el vector de características de la palabra es, la palabra en sí, la raíz y la categoría gramatical *POS* (*Part-of-Speech*).

Las secuencias maximales también se han propuesto (Coyotl-Morales, 2006) como una herramienta para la atribución de autoría. Para atribuir la autoría de un determinado texto anónimo, nos basamos en que cada autor tiene su propio estilo y hábitos de escritura que distinguen su escritura de la de cualquier otro autor. En cierta manera esto no se cumple siempre puesto que el estilo del autor puede estar influido por el público al que se dirige e incluso por su estado de ánimo. Existen varios métodos para la atribución de autoría, métodos basados en estilo, como el anteriormente descrito, métodos basados en características sintácticas y métodos

basados en palabras. Dentro de este último grupo, entraría el método de usar secuencias frecuentes maximales. El hecho de no usar n-gramas, es decir, secuencias de una determinada longitud prefijada, da pie a que surjan expresiones que son utilizadas por el escritor normalmente, y que pueden determinar si es el autor de determinado texto.

1.3 Similitud entre textos

Las secuencias maximales también pueden utilizarse para determinar el grado de similitud entre dos textos. Esto puede ser útil a la hora de detectar plagio, puesto que podemos ver hasta qué punto dos documentos se asemejan.

Siguiendo a (Barrón Cedeño & Rosso, 2008), definimos la similitud entre dos textos como:

$$sim(d, d_q) = [0, 1]$$, siendo d y d_q textos.

Para $sim(d, d_q)=0$, diremos que los documentos no se parecen en nada. Con $sim(d,d_q)=1$, se demostrará que los textos son iguales.

Para calcular la similitud entre dos textos, tendremos en cuenta el número de secuencias maximales en común que tienen, así como la longitud de cada una de ellas. Por tanto, la similitud entre dos textos vendrá definida por:

$$sim(d, d_q) = \frac{\sum_{\alpha=2}^{\lambda} n_\alpha \cdot \alpha}{|d_q|}$$

Siendo λ la longitud de la secuencia maximal más larga encontrada que aparece simultáneamente en d y d_q, y n_α la cantidad de secuencias maximales de longitud longitud α.

Los resultados se normalizarán para que estén dentro del rango esperado. Puesto que dos documentos idénticos suponen una única secuencia maximal en común tan larga como el número de palabras del documento, se normalizará el resultado obtenido, dividiendo por el número de palabras de la revisión más actual, para obtener valores en el rango [0,1].

Será cosa del usuario determinar a partir de qué umbral de similitud se puede considerar que un texto es plagio de otro.

1.4 Organización de la memoria

La memoria está organizada como sigue. En el capítulo 2, se expone el método propuesto para la extracción de secuencias frecuentes maximales, incluyendo el parseado de los documentos y la construcción de la estructura, el algoritmo de búsqueda y el algoritmo de inserción con comparación. En el capítulo 3, se describen los dos experimentos realizados, así como el corpus utilizado para los mismos, y la discusión de los resultados obtenidos. El capítulo sección 4 describe las conclusiones y el trabajo futuro.

2. Método propuesto

2.1 Definición del problema

Una secuencia S es una lista ordenada de k elementos de la forma $<s_1,s_2,s_3,\ldots s_k>$. La longitud de una secuencia S es el número de elementos en S. Llamaremos k-secuencia a las secuencias de longitud k.

Una secuencia P será sub-secuencia de S si existe un entero $i \geq 1$, que cumpla:

$$p_1 = s_i,\ p_2 = s_{i+1},\ p_3 = s_{i+2},\ \ldots\ p_n = s_{i+(n-1)}$$

En este caso, diremos que: $P \subseteq S$

Consideramos un documento como una secuencia de palabras, expresado de la siguiente forma:

$$<w_1, w_2, w_3, \ldots w_n,>$$

La frecuencia de una secuencia, S_i, es el número de documentos dónde aparece dicha secuencia. Diremos que una secuencia es β-frecuente si aparece al menos en β documentos y no es sub-secuencia de ninguna otra secuencia P que aparezca con la misma frecuencia.

Denominaremos capacidad de salto, η, a la cantidad de espacios que se pueden dejar entre dos palabras para poder considerarlas parte de la misma secuencia. Así pues, las secuencias continuas serán aquellas que tengan capacidad de salto $\eta = 0$, es decir, desde una palabra, saltamos a la inmediatamente siguiente. De este modo, las secuencias con $\eta = 0$ serán del tipo: $<w_i, w_{i+1}, w_{i+2}, \ldots w_{i+n},>$. Cuando añadamos una $\eta > 0$, podremos obtener con el siguiente formato: $<w_i, w_{i+j}, w_{i+k}, \ldots>$, dónde $j,k \leq \eta$

2.2 Lenguaje de programación y plataforma

Para la realización de este proyecto se ha elegido Java como lenguaje de programación, puesto que es un lenguaje de programación orientado a objetos que nos permitirá modelar el sistema de una forma más simple, pudiendo tratar cada una de las partes (palabra, documento, secuencia maximal, etc.) como un objeto.

Al mismo tiempo, el hecho de programar en Java nos permite aprovechar las muchas herramientas y librerías que existen. En concreto, vamos a utilizar la librería de parseado de

XML, disponible en http://www.saxproject.org, que nos va a ser muy útil para cargar la colección de documentos que queremos analizar.

Como entorno de desarrollo, o IDE (i*ntegrated development environment*), hemos utilizado la plataforma IDE NetBeans. Hemos elegido IDE NetBeans porque es un proyecto de código abierto, libre y gratuito, sin restricciones de uso y escrito completamente en Java. Se encuentra disponible en http://netbeans.org.

2.3 Parseado de los documentos

Las colecciones de datos con las que trabajaremos vienen dadas en formato XML. Todos los documentos deben estar incluidos en un mismo fichero XML de tal manera que cada uno de ellos esté identificado con una etiqueta <doc>, y todos ellos estén agrupados con una etiqueta <documentos>. El documento XML debe de estar bien formado, es decir, puede contener otras etiquetas y propiedades, pero todas ellas tienen que estar cerradas.

El parseado de los documentos se hace mediante una librería SAX (*Simple API for XML*) para parseado de ficheros XML. Se barajaron dos posibilidades para hacer el parseado de los documentos. Éstas fueron DOM y SAX. Se eligió SAX, estándar "de facto", por ser más rápido y necesitar menos memoria. SAX está dirigido por eventos, recorre el documento de principio a fin y lanza un evento cada vez que encuentra una etiqueta que hayamos incluido en la lista de etiquetas a buscar. La mayor ventaja de SAX es que es muy rápido y que no ocupa mucho espacio en memoria, a diferencia de DOM, que almacena una copia en memoria de la información que contiene el documento estructurada jerárquicamente. El único inconveniente que presenta SAX es que no se puede recorrer "hacia atrás" es decir, que una vez hemos pasado por una etiqueta, no se puede volver a la anterior, a no ser que empecemos de nuevo por el principio. Esto no será un problema para nosotros, puesto que sólo queremos leer una vez el documento para crear nuestra propia estructura de datos.

2.4 Construcción de la estructura

Para la correcta ejecución del algoritmo que propondremos, necesitamos tener almacenada toda la información de los documentos que vamos a analizar. Necesitamos almacenar cada una de las palabras que aparecen en toda la colección de textos. Será necesario, al mismo tiempo, saber en qué documentos aparece cada una de las palabras encontradas y qué posición o posiciones ocupa dentro de cada uno de los documentos en los que aparece. Cada

una de las palabras deberá tener una referencia a la palabra que le siga, de tal forma que sea posible recorrer la estructura para leer el documento.

Se ha elegido una estructura de tipo tabla *hash* para almacenar la información relativa a cada palabra. A cada palabra se le asigna, mediante una función, un código *hash*, que será el índice de la tabla. Esto nos permitirá recuperar la información de cierta palabra accediendo a la tabla *hash* con la palabra en cuestión.

Denominamos al elemento básico de la estructura NodoPalabra (ver Figura 1). Cada NodoPalabra contiene la información relativa de una determinada palabra respecto a un determinado documento. Así, en cada NodoPalabra almacenamos la palabra a la que se refiere, representado por el idPalabra, y el número de documento en el que aparece. Por cada palabra distinta que aparece en un determinado documento se crea un solo nodo.

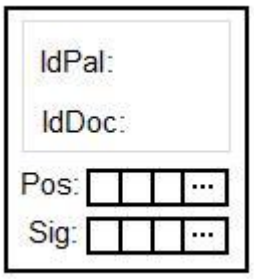

Figura 1 - NodoPalabra

Al recorrer los documentos, nos encontraremos con palabras que aparecen repetidas dentro del mismo. Por ejemplo, dado el siguiente documento: "*La agricultura es el arte de cultivar la tierra.*", la palabra "*la*" aparece dos veces, ocupando las posiciones 1 y 8. Además está seguida de "*agricultura*" y "*tierra*", respectivamente para cada una de las posiciones. En cambio, sólo queremos almacenar un NodoPalabra por cada palabra y documento en el que aparece, por lo que incluimos un vector con las posiciones que ocupa la palabra en cuestión en el documento y un enlace al NodoPalabra que sigue a cada una de las posiciones. En el NodoPalabra correspondiente a la palabra "*la"* del ejemplo anterior, añadiríamos un vector con las posiciones 1 y 8, y un vector de referencias a los NodoPalabra que ocupen las posiciones 2 y 9 en el documento. De este modo, desde la primera palabra de un documento, seremos capaces de recorrer toda la estructura y recuperar todas las palabras que aparecen en el documento en el orden correcto.

Para almacenar todos los NodoPalabra de todos los documentos, se utiliza un diccionario de palabras a listas enlazadas, de tal manera que de cada palabra *w* con id = i, se obtiene una lista L_i^w de las posiciones que ocupa dicha palabra en cada uno de los documentos en los que aparece, sabiendo en cada caso qué palabra le sigue. La longitud de L_i^w será, por tanto, la frecuencia de la palabra (w_f) en la colección de documentos. Esta lista será necesaria para la ejecución del algoritmo, como veremos más adelante.

Según lo descrito, si consideramos los documentos que aparecen en la Tabla 1, la estructura de datos para los mismos quedaría como la mostrada en la Figura 2

Doc. 1	En la casa roja de la esquina.
Doc. 2	En la casa de la plaza.
Doc. 3	En la puerta de la casa.

Tabla 1 – Documentos de ejemplo

En la Figura 2 podemos ver cómo, a través de una palabra de la tabla, podemos acceder a la información relativa a ella. La estructura permite ser recorrida tanto en horizontal, por palabras, como en vertical, por documentos. Para poder empezar la lectura de un documento desde el principio (lectura vertical) necesitaremos almacenar en una lista las referencias a las primeras palabras de cada documento, de modo que a partir de ellas podamos empezar la lectura del documento y reproducirlo en su totalidad siguiendo los enlaces verticales.

Esta lista de referencias a las primeras palabras de cada documento, se implementa usando un vector de punteros a NodoPalabra, tal que en la posición 1 del vector, exista una referencia al primer Nodo Palabra del documento 1, y así sucesivamente.

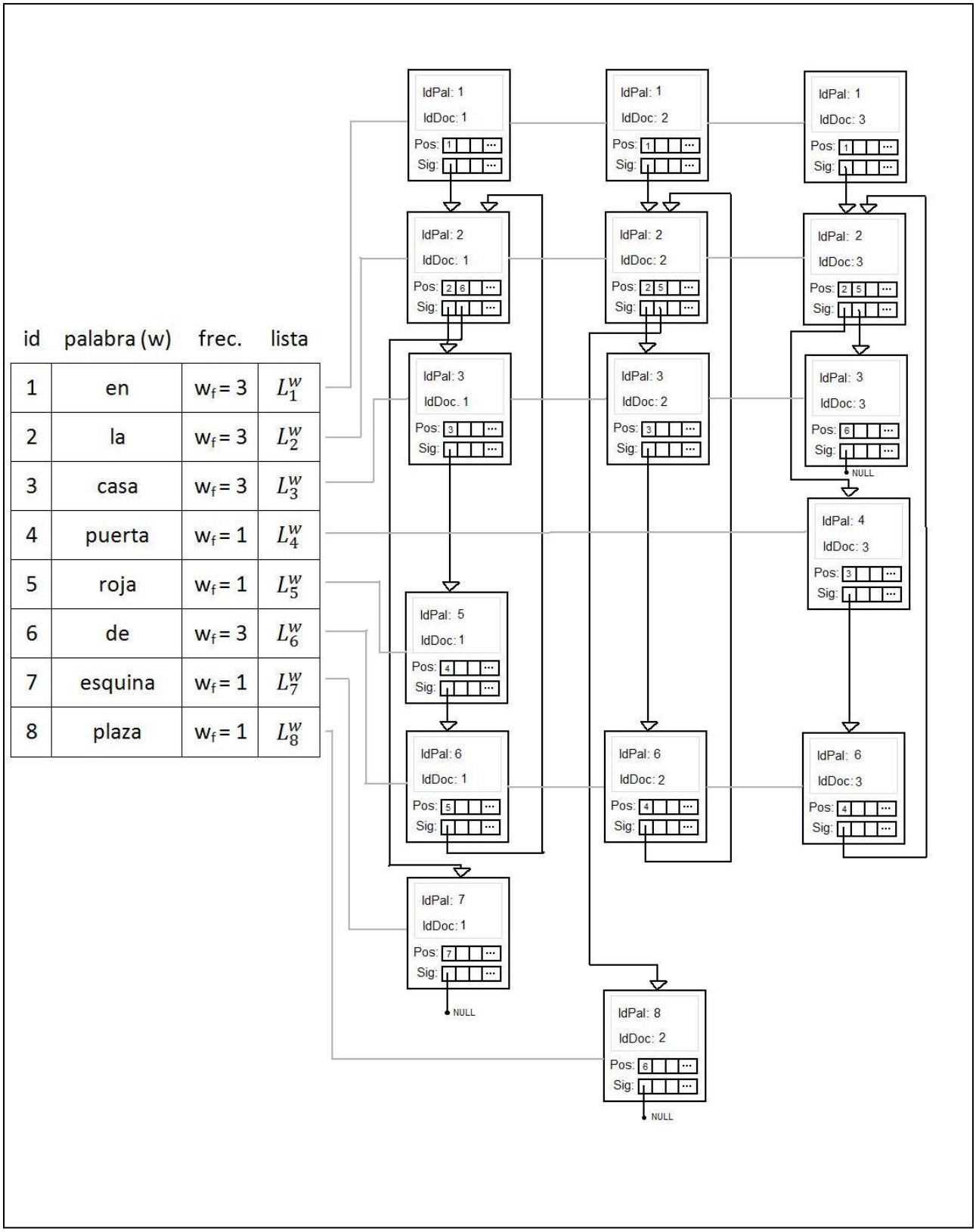

Figura 2 - Estructura de datos. Tabla *hash* de listas enlazadas para los documentos mostrados en la Tabla 1

2.5 Algoritmo de búsqueda de secuencias maximales

El objetivo de nuestro algoritmo es, dada la estructura anteriormente descrita, y una frecuencia, encontrar todas las secuencias que cumplan, como mínimo, la frecuencia solicitada y que además, sean lo más largas posible.

Este algoritmo está basado en el algoritmo DIMASP, presentado en (García-Hernández et al., 2006). Se trata de un algoritmo incremental basado en *pattern-growing* (crecimiento de patrones). Este algoritmo es capaz de encontrar las secuencias idénticas de palabras que se repiten β veces en la colección de documentos, siendo β la frecuencia indicada por el usuario.

Esta aproximación da muy buenos resultados cuando lo que buscamos son secuencias idénticas. El inconveniente que tiene este modelo para su uso en la detección de similitud entre textos y patrones de escritura es que el algoritmo sólo detectará la secuencia como β frecuente si se repite *exactamente igual* en β documentos. Esta característica limitará bastante la búsqueda puesto que es probable que las frases no sean exactamente iguales. Cabe la posibilidad de que se hayan cambiado ciertas palabras por sus sinónimos o se hayan añadido más palabras a la secuencia con el fin, precisamente, de evitar la detección del plagio. Por ejemplo, podríamos modificar la frase siguiente:

Doc. 0	*"Los escritores fueron aplaudidos por la audiencia".*
Doc. 1	*"Sólo los mejores escritores fueron tan aplaudidos por la gran audiencia."*

Tabla 2- Documentos de ejemplo

Si buscamos las secuencias maximales de estos dos documentos, con la primera aproximación del algoritmo sólo encontraría las secuencias que son exactamente iguales. Para este ejemplo concretamente, encontraría dos secuencias maximales, *"escritores fueron"* y "*aplaudidos por*" que no resultarían significativas, puesto que por sí solas no tienen mucho significado y además, tienen una longitud pequeña, por lo que serán menos consideradas que las de mayor longitud.

Para sortear este obstáculo y que no limite la calidad de nuestros resultados, hemos añadido una nueva característica al algoritmo. El usuario podrá definir un nuevo parámetro, que

llamaremos *capacidad de salto, o gap*. La *capacidad de salto* indicará el número máximo de espacios en blanco que se pueden dejar entre dos palabras que pertenecen a una secuencia. Se propone esta mejora con respecto al algoritmo propuesto en (García-Hernández et al., 2006)

Con esta modificación, si buscamos las secuencias maximales de los dos documentos incluyendo una capacidad de salto de una palabra, nos daría una sola secuencia maximal, "*Los escritores fueron aplaudidos por la audiencia*", que efectivamente se repite en los dos documentos y ésta sí será significativa. Tendrá una longitud adecuada y significado completo por sí sola, es decir, nos dará información más precisa para determinar que los documentos están relacionados.

La idea básica del algoritmo es, para cada uno de los documentos, ir recorriendo sus palabras y coger aquella que tenga una frecuencia de aparición mayor que el umbral definido por el usuario (β). Cuando encontramos una palabra que cumple esta condición, nos la guardamos y miramos qué palabra le sigue. Unimos la palabra siguiente a la actual, obteniendo así un bi-grama, y volvemos a comprobar si sigue superando la frecuencia mínima. Si la supera, agregamos la palabra que le sigue, y volvemos a hacer la comprobación. Seguiremos así hasta que deje de cumplirse, es decir, hasta que la siguiente palabra ya no sea frecuente. De ese modo garantizamos que la secuencia sea maximal, puesto que la hacemos crecer hasta que ya no cumpla la condición.

En la Figura 3 se puede ver, en pseudo-código, el algoritmo general con más detalle.

```
Sean n y β la capacidad de salto y la frecuencia respectivamente.
Sea Δ la colección de documentos.

para cada Documento D ∈ Δ
  para cada NodoPalabra actual ∈ D
        idW = actual.id
        w = actual.word
        mientras que frec_w ≥ β
            PSM += w    // Ampliamos la posible secuencia maximal (PSM) añadiendo la palabra
            ListaActual = L_w^idW   //Almacenamos la lista de apariciones de la palabra
            nextId = actual.siguientePalabra.id  //El id de la palabra que le sigue según D
            para cada NodoPalabra aux ∈ ListaActual
                posicionActual = aux.posicion
                para cada nodoSiguiente ∈ aux.siguientesNodos
                    si nodoSiguiente.id = nextId entonces
                        si nodoSiguiente.posiciones contiene p ≤ posicionActual + n entonces
                            //Si le sigue el mismo nodo teniendo en cuenta la capacidad de salto, lo
                        // añadimos a la lista de validos
                                ListaValidos ← ListaValidos + aux
                        fin si
                    fin si
                fin para cada
            fin para cada
            //En ListaValidos están incluidos todos los nodos que cumplen la propiedad:
            // "les sigue un nodo cuyo id es igual al id de la palabra que le sigue a actual"
            frec_w = ListaValidos.longitud
            actual = aux;
            w = actual.word;
            idW = actual.id;
            // Seguiremos hacienda crecer el PSM mientras se cumpla que la frecuencia es mayor que β
        fin mientras que
        si longitud(PSM) ≥ 2 entonces
            insertamos PSM en la tabla de Secuencias Maximales
        sino
            descartar
        fin si
    fin para cada
fin para cada
```

Figura 3 - Algoritmo principal de búsqueda, en pseudo-código. Localiza cada PSM (posible secuencia maximal)

2.6 Algoritmo de almacenamiento con comparación

El algoritmo anterior (Figura 3) va recorriendo cada uno de los documentos y al encontrar una secuencia maximal, la almacena. Sin embargo, hay casos en los que las secuencias maximales que encontramos no necesitan ser almacenadas porque, o bien ya están en la colección de secuencias maximales, o están incluidas en alguna de las secuencias maximales de mayor longitud existentes.

Por ejemplo, si el algoritmo encuentra la secuencia: "*cada diez años*", no deberá almacenarla si ya tenemos otra secuencia en la tabla como: "*La visita se repite cada diez años*".

Es muy probable que se encuentren secuencias maximales repetidas, o incluidas en otras, por lo que el número de comparaciones puede ser muy elevado. Para mejorar la eficiencia del algoritmo habrá que disminuir el número de comparaciones. Nos aprovecharemos de que cada palabra tiene asignado un id (*integer*) que la representa inconfundiblemente. Este id se añadió para almacenarla en la tabla *hash*, y nos será útil ahora. También añadimos un atributo *sumaIndex*, que es la suma (en valor absoluto) de todos los índices que componen la secuencia maximal. El almacenamiento se realiza según la longitud de las secuencias. Denominaremos *k-SM* a una secuencia maximal de longitud *k*.

La estructura para el almacenamiento de las *k-SM* es la mostrada en la Figura 4. Se ha elegido una estructura de tabla *hash* de vectores, donde en un mismo vector se almacenan las secuencias de una misma longitud *k*. Dentro de un mismo vector, las palabras se almacenan por orden creciente de su *sumaIndex*. Nótese que para que dos secuencias maximales sean idénticas, su *sumaIndex* deberá ser la misma. Del mismo modo, para que exista la posibilidad de que una secuencia maximal esté incluida en otra, el valor de *sumaIndex* de la contenedora deberá ser estrictamente mayor que el valor de *sumaIndex* de la secuencia maximal contenida, puesto que incluye esta secuencia más algún otro fragmento que hará incrementar el valor de *sumaIndex*.

El algoritmo de almacenamiento de las secuencias maximales se divide en tres pasos. En primer lugar debemos comprobar que la secuencia encontrada no se encuentra ya entre las almacenadas con la misma longitud y que no es sub-secuencia de ninguna secuencia maximal de mayor longitud. En caso de que no se diese ninguno de los dos casos anteriores, procedemos a insertarla. Seguidamente, comprobamos que no existe ninguna secuencia ya insertada que esté incluida en la que acabamos de añadir. Para ello se comparan todas las *k-SM* con *k* menor que la longitud de la recién insertada.

El atributo de *sumaIndex* se utiliza para mejorar el tiempo de ejecución de la inserción. *sumaIndex* es la suma de los índices, en valor absoluto, de las palabras que contienen la secuencia. Así pues, *sumaIndex* será un valor positivo. Este atributo nos permite disminuir muy considerablemente el número de comparaciones. Si queremos comprobar si la secuencia A es sub-secuencia de la secuencia B, es decir, si A está contenida en B, comprobamos en primer lugar que *sumaIndex(B)* > *sumaIndex(A)*. Si no se cumple que la *sumaIndex* de B es estrictamente mayor que la *sumaIndex* de A, no es posible que estén contenidas. Lo mismo pasa al intentar comparar si dos cadenas son iguales. Para que lo sean, una condición necesaria, aunque no suficiente, es que tengan el mismo valor para *sumaIndex*.

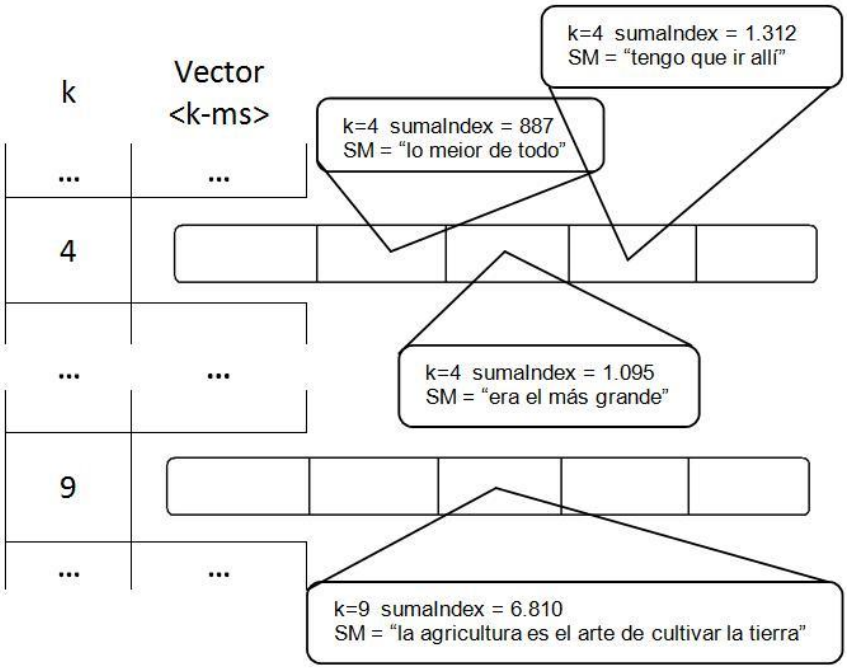

Figura 4 - Estructura de almacenamiento de las secuencias maximales (SM)

3. Experimentos

3.1 Descripción del corpus

El corpus[1] que vamos a utilizar para los experimentos es el presentado en (Barrón-Cedeño et al., 2009). Se trata de un corpus sacado de artículos de Wikipedia, y generado para el análisis de co-derivados, reutilización de texto y plagio. Está compuesto de más de 20.000 documentos de la Wikipedia en español, alemán, inglés e hindi (alrededor de 5.000 documentos por idioma). Para cada idioma, se han considerado algunos de los artículos que son consultados con más frecuencias en Wikipedia y se han recopilado 10 revisiones de cada uno de ellos, distribuidas uniformemente entre las últimas 500 revisiones. Estas revisiones de cada artículo componen los co-derivados. Se han descartado las revisiones que han sido rechazadas por los revisores por poder contener datos falsos.

Dado que el corpus es muy extenso, y debido a la limitación tanto temporal como de maquinaría, se han pre-seleccionado dos subconjuntos de 10 artículos cada uno, pertenecientes a la versión española de la Wikipedia y a la inglesa, respectivamente. Los artículos seleccionados, tienen un tamaño medio de 330Kb y entre 5.000 y 6.000 palabras por revisión

.

3.2 Similitud entre revisiones del mismo artículo de Wikipedia

El primero de los experimentos consistirá en comparar las similitudes entre textos co-derivados. Un co-derivado se define como una pareja de documentos que son revisiones o plagios el uno del otro (Hoad & Zobel, 2003).

En (Barrón-Cedeño et al., 2009) se comparan los resultados de calcular la similitud entre dos textos por 7 métodos distintos, estos son: Vector Space Models (Jaccard Coefficient, Cosine Similarity), Fingerprint Models (Winnowing Fingerprinting y SPEX Algorithm) y

[1] Co-Derivatives Corpus. A.Barrón-Cedeño. Natural Language Engineering Lab. Disponible en: http://users.dsic.upv.es/grupos/nle/downloads

Modelos probabilisticos(Machine translation y Kullback-Leibler Distance). Para más detalle sobre cada método, ver referencia.

En el experimento propuesto, compararemos los documentos a partir de las secuencias maximales obtenidas. Para ello, supondremos que la última revisión de cada artículo es la actual y compararemos cada una de las 9 revisiones anteriores con la actual, para poder ver la evolución del artículo. El resultado esperado es que cuanto más cercano en el tiempo esté la revisión en cuestión de la actual, más similitudes habrá entre ellas.

La comparación se hará con el algoritmo propuesto, es decir, a partir de las secuencias maximales que aparecen en común en los textos. Para ello, iremos comparando cada una de las 9 revisiones con la actual, buscando secuencias maximales que se repitan 2 veces, es decir, que aparezcan en ambos documentos.

Tal y como hemos explicado en el capítulo 1 sección 3, la similitud entre textos vendrá definida por:

$$sim(d, d_q) = \frac{\sum_{\alpha=2}^{\lambda} n_{\alpha} \cdot \alpha}{|d_q|}$$

Siendo λ la longitud de la secuencia maximal más larga encontrada que aparece simultáneamente en d y d_q, y n_α la cantidad de secuencias maximales de longitud longitud α.

Como se ha mencionado anteriormente, para este experimento se han seleccionado al azar 10 artículos del corpus pertenecientes a la versión española de la Wikipedia, y otros 10 pertenecientes a la versión inglesa.

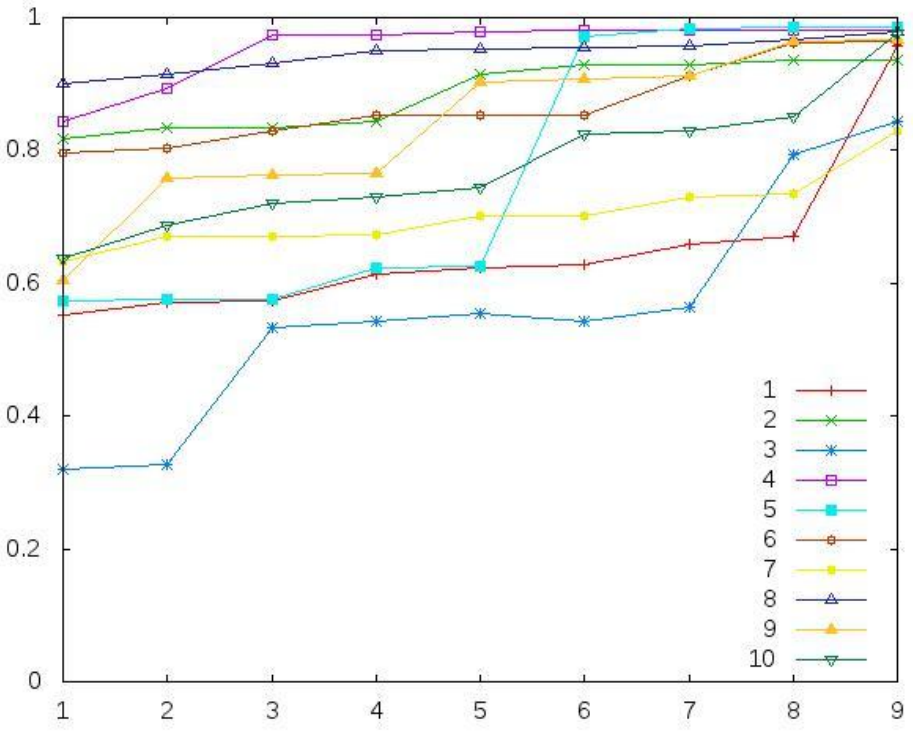

Figura 5 - Similitud entre las revisiones [1...9] de cada uno de los diez artículos de la Wikipedia en español y la versión actual del mismo.

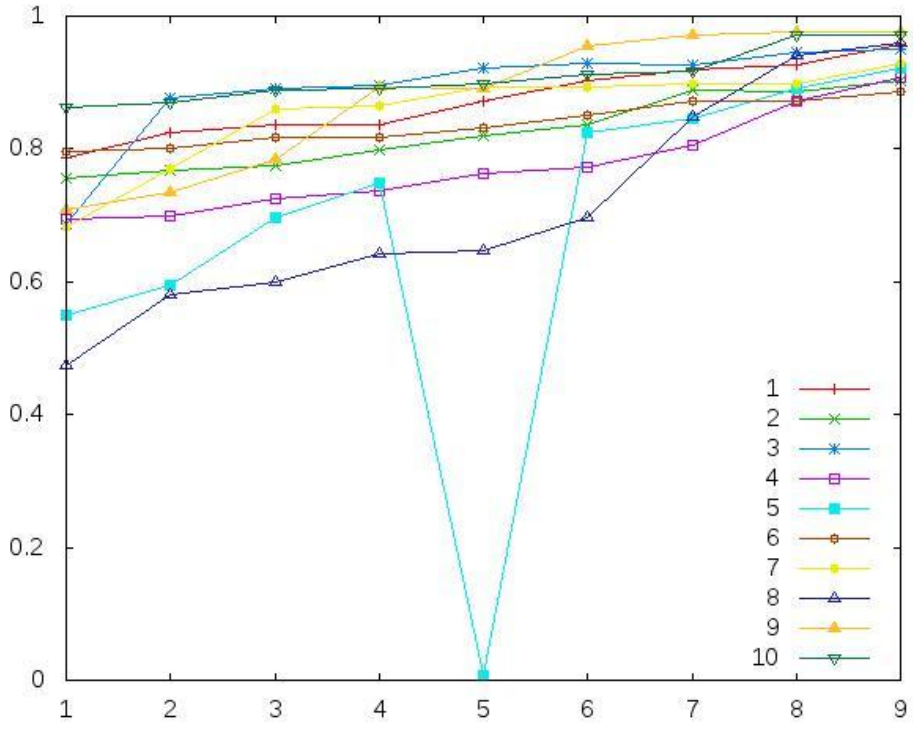

Figura 6 - Similitud entre las revisiones [1...9] de cada uno de los diez artículos de la Wikipedia en inglés y la versión actual del mismo. En la revisión 5 del artículo 5 se observa un blanqueo, acto de vandalismo en Wikipedia.

Como podemos observar en las Figuras 5 y 6, la tendencia de las revisiones es a ser cada vez más similares a la actual. Esto es debido a la naturaleza colaborativa de la Wikipedia, que se edita entre distintas personas, aportando nueva información o modificando la información existente en cada una de las ediciones. Los artículos que presentan una gráfica con una mayor pendiente son los más nuevos, es decir, los que se han empezado a editar más recientemente, de modo que los cambios que están sufriendo son más pronunciados. En los casos en los que la inclinación de la recta es menos pronunciada, se puede deducir que el contenido del artículo está básicamente completo, y sólo se producen algunas modificaciones menores de una revisión a la siguiente.

En la Figura 6 se observa un comportamiento anormal en la revisión número 5 del artículo quinto. Se trata de un acto de vandalismo, habitual en Wikipedia (Potthast, Stein, & Gerling, 2008). Concretamente se trata de un "blanqueo", es decir, alguien ha editado el contenido y lo ha borrado todo. Afortunadamente, existen los bibliotecarios de Wikipedia, que pueden restaurar un artículo a cualquiera de sus revisiones anteriores y así defender la información contra este tipo de vandalismo.

Por lo demás, las gráficas presentan los resultados esperados: un incremento de la similitud entre los textos a medida que comparamos revisiones más cercanas en el tiempo a la actual. En unos casos, este incremento es más pronunciado que en otros, dependiendo de la novedad del artículo o la controversia del mismo.

A continuación se muestran las gráficas obtenidas para algunos de los artículos, que representan el número de secuencias frecuentes maximales encontradas, según sus longitudes. Para que se vea más claro, solamente se incluyen las revisiones impares de cada artículo. El eje horizontal representa la longitud de las secuencias encontradas, mientras que el vertical indica la cantidad de cada una de ellas. Se ha utilizado una escala logarítmica en el eje horizontal, para que se puedan leer mejor los datos.

En las Figuras 7-9, se muestran las secuencias maximales encontradas en 4 de los documentos pertenecientes a los seleccionados de la Wikipedia en español. Se corresponden con los artículos 1, 3, 5 y 10.

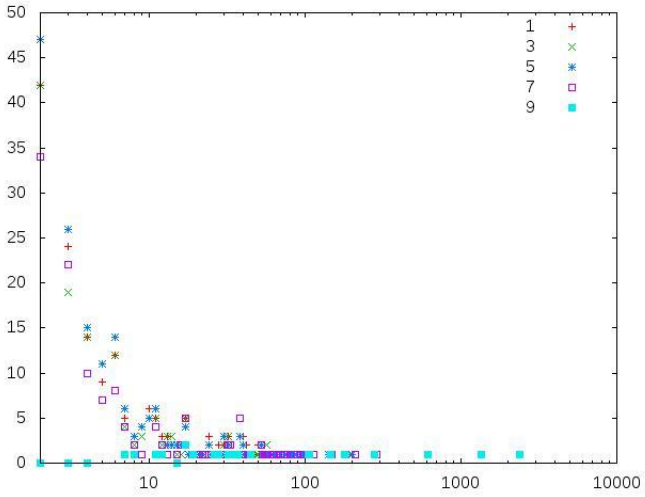

Figura 7 - Longitud de las secuencias maximales en función de la cantidad, para las revisiones impares del artículo 1 de la Wikipedia en español

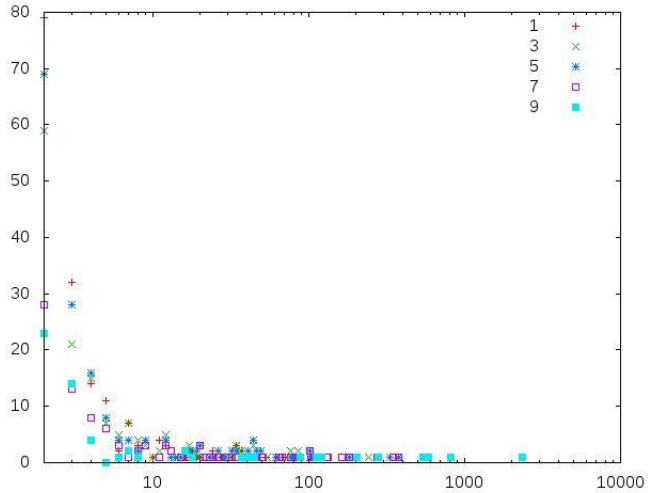

Figura 8 - Longitud de las secuencias maximales en función de la cantidad, para las revisiones impares del artículo 3 de la Wikipedia en español

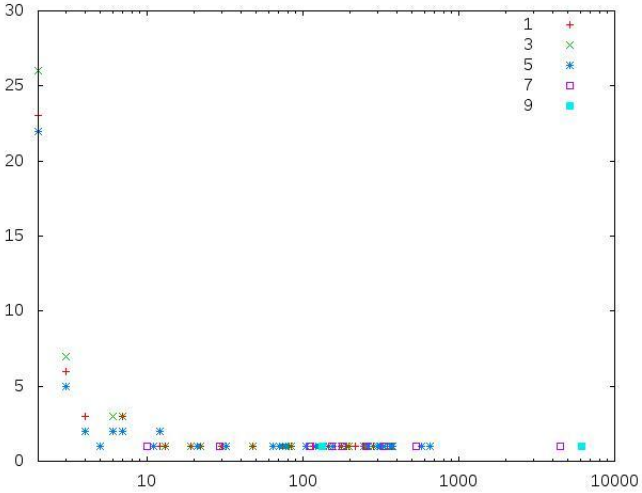

Figura 9 - Longitud de las secuencias maximales en funcion de la cantidad, para las revisiones impares del artículo 5 de la Wikipedia en español

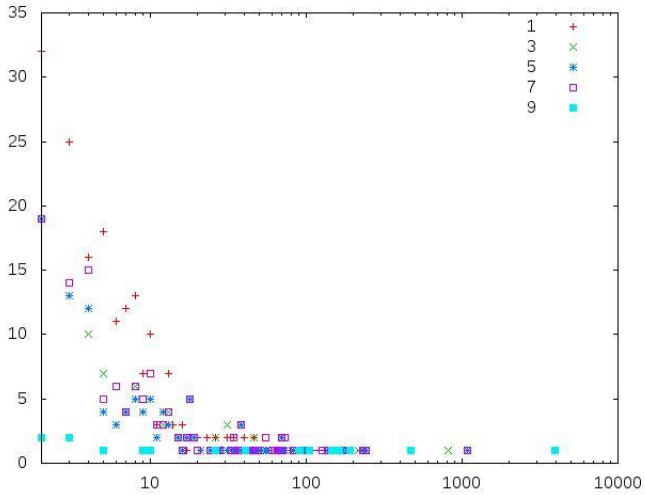

Figura 10 - Longitud de las secuencias maximales en función de la cantidad, para las revisiones impares del artículo 10 de la Wikipedia en español

A continuación se muestran las mismas gráficas para algunos de los artículos tomados de la versión inglesa. Podemos observar que en esto particularmente, no hay mucha diferencia entre la versión española y la inglesa.

En las Figuras 11- 14, se muestran las secuencias maximales encontradas en 4 de los documentos pertenecientes a los seleccionados de la Wikipedia en inglés. Se corresponden con los artículos 1, 4, 7 y 8.

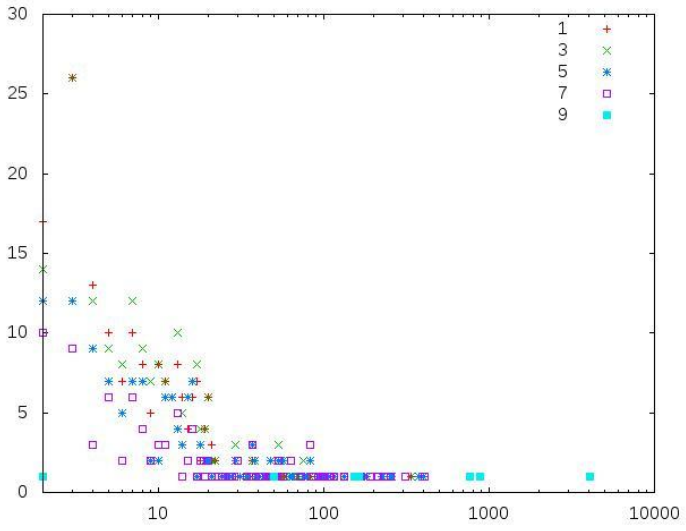

Figura 11 - Longitud de las secuencias maximales en función de la cantidad, para las revisiones impares del artículo 1 de la Wikipedia en inglés

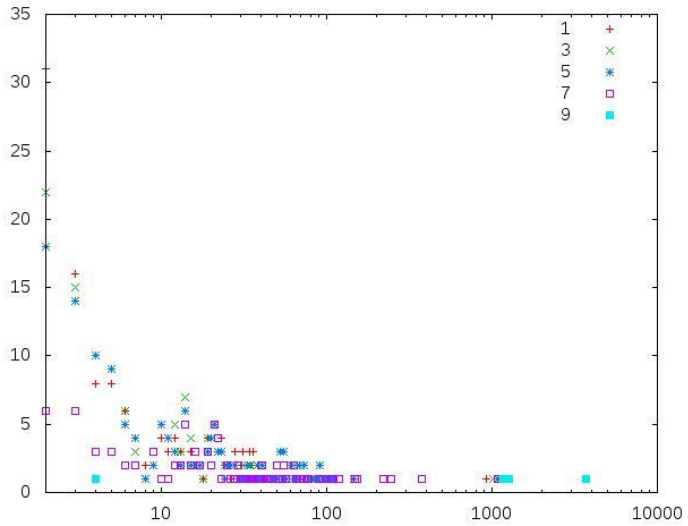

Figura 12 - Longitud de las secuencias maximales en función de la cantidad, para las revisiones impares del artículo 4 de la Wikipedia en inglés

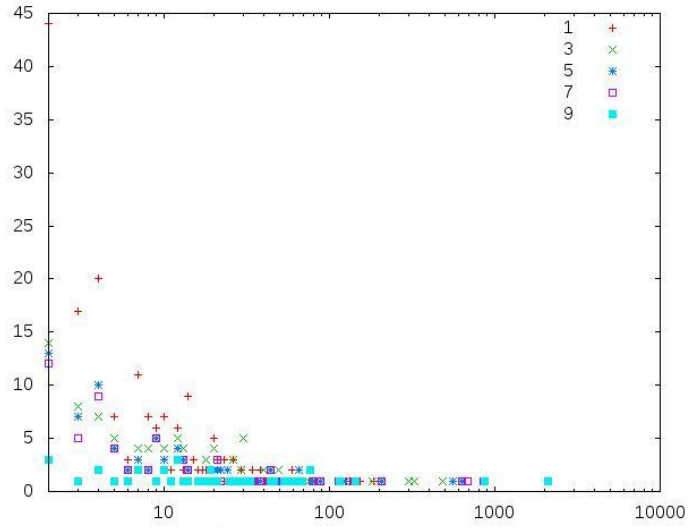

Figura 13 - Longitud de las secuencias maximales en función de la cantidad, para las revisiones impares del artículo 7 de la Wikipedia en inglés

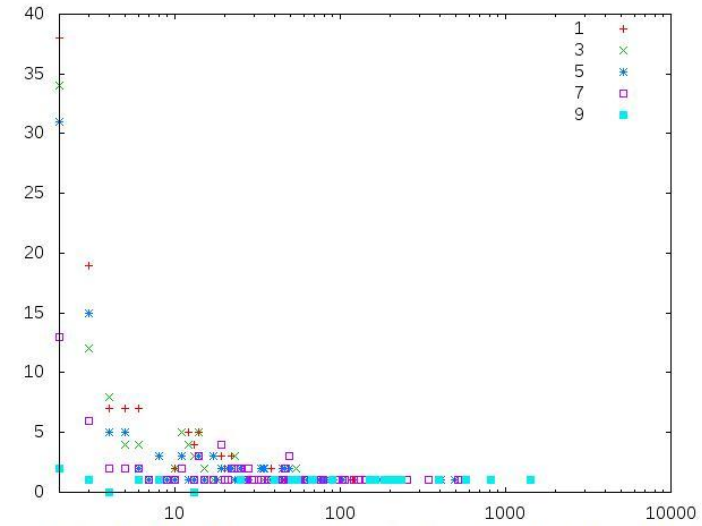

Figura 14 - Longitud de las secuencias maximales en función de la cantidad, para las revisiones impares del artículo 8 de la Wikipedia en inglés

Como se puede observar en las gráficas, tanto para el caso de los artículos en inglés como en el de los artículos en español, en las primeras revisiones se encuentra una mayor cantidad de secuencias maximales más cortas. A medida que nos vamos acercando a la revisión actual, las secuencias maximales cortas se unen para formar secuencias más largas. Por eso en las últimas revisiones podemos ver menor cantidad de secuencias maximales, pero de longitudes considerables. Por ejemplo, en la revisión 9 del artículo quinto de los españoles (Figura 9), se ha llegado a encontrar una secuencia maximal de una longitud de 6090 palabras.

El tiempo de ejecución del programa para estos experimentos ha sido inferior a 3 minutos, por lo general, aunque se han dado casos en los que el tiempo se ha disparado al encontrar secuencias maximales de más de 5.000 palabras. En estos casos, ha tardado alrededor de 135 minutos. Según lo observado en los experimentos, podemos decir que el tiempo de ejecución es sensible a la longitud de las secuencias maximales encontradas, más que a la longitud del documento. Se disparan los tiempos de ejecución para el análisis de textos muy similares entre ellos, con similitud entre [0.8, 1].

3.3 Detección de secuencias maximales incluyendo huecos entre palabras

El objetivo de este experimento es poner a prueba la habilidad del programa para detectar secuencias maximales teniendo en cuenta un salto. Como se ha descrito anteriormente, el programa tiene un parámetro ajustable llamado *capacidad de salto* o *gap*. La *capacidad de salto* indicará el número máximo de espacios en blanco que se pueden dejar entre dos palabras que pertenecen a una secuencia.

Para la realización del experimento, se ha seleccionado un fragmento de un texto[2]. Este fragmento está compuesto de distintas frases distribuidas en 4 párrafos, con un total de 400 palabras aproximadamente. A partir de este texto, se han hecho 4 copias, insertando palabras nuevas en algunos de los párrafos, intentando camuflar el plagio. Así pues, obtenemos 5 documentos, siendo uno de ellos el original, y los otros 4, copias idénticas con algunas palabras insertadas en uno de los párrafos. En la Figura 15 se ve el texto y las palabras insertadas en cada uno de los documentos. Cada uno de los documentos copia, incluye el texto entero con las modificaciones en el correspondiente párrafo, tal y como se muestran en la figura.

El resultado esperado para este experimento es que, al ejecutar el programa sobre este mini-corpus que nos hemos creado (con los 5 documentos; el original y las 4 copias con palabras intercaladas), encuentre la similitud entre los 5 documentos.

Hemos ejecutado el algoritmo, cargando los 5 documentos y le hemos pedido que busque las secuencias maximales con una frecuencia mayor o igual a 5, es decir, secuencias maximales que se repitan en al menos 5 documentos distintos. Para que pueda detectar la copia a pesar de las palabras insertadas, se le establece una *capacidad de salto* o *gap* de 3. Esto es,

[2] Se trata de un fragmento de: “On the cruelty of really teaching computing science” by Edsger Dijkstra.

podrá dejar hasta tres huecos de palabra en la secuencia. El resultado obtenido es una secuencia maximal única, de una longitud de 403 palabras que incluye el documento original completo.

Tras la ejecución del experimento, que apenas dura 20 segundos, el programa produce el resultado esperado. A pesar de que habíamos intentado "esconder el plagio" añadiéndole palabras intercaladas en las copias del texto, el programa detecta la completa similitud entre los distintos documentos que le hemos pasado. Esto demuestra que el algoritmo es capaz de encontrar secuencias maximales en colecciones de textos aunque sean algo diferentes. Esta característica del programa es muy interesante si lo que se pretende es detectar plagio entre textos, puesto que es habitual el insertar palabras nuevas en medio de las frases para despistar.

En el Anexo B pueden consultarse las salidas proporcionadas por el programa: ResExperimento2.txt, GraficaResExperimento2.txt, y StatResExperimento2.txt.

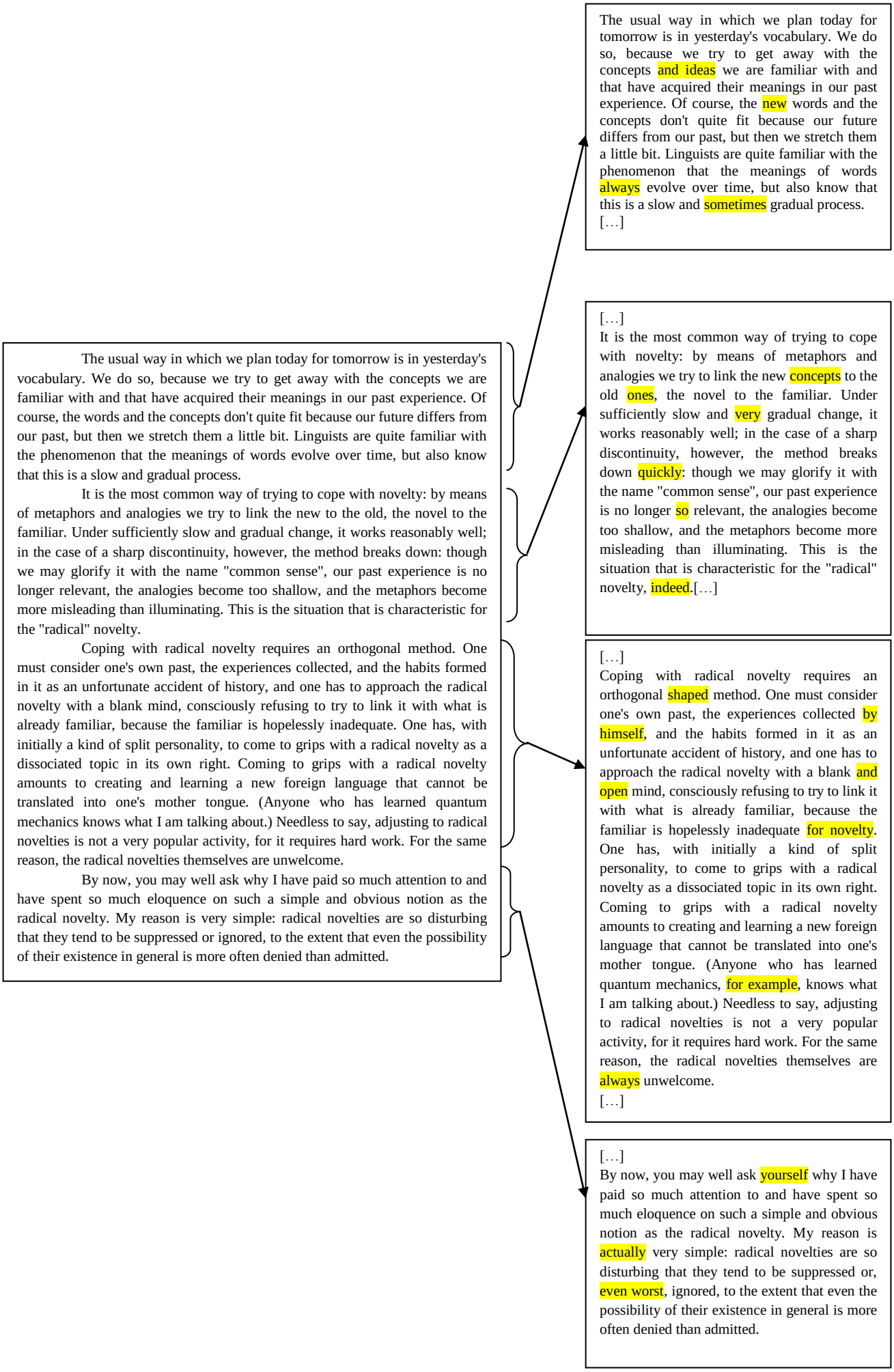

Figura 15 - Documento original y sus modificaciones para la búsqueda de secuencias maximales considerando un *gap* de separación entre palabras

4. Conclusiones y trabajo futuro

La extracción de secuencias frecuentes maximales de una colección de textos nos permite conocer en qué grado son similares los distintos textos de una colección. Esto puede ser útil para clasificar. La clasificación automática de textos, consiste en colocar un documento dentro de un grupo de clases previamente definidas. Usando esta información sería posible clasificar automáticamente los textos por temas, puesto que textos de la misma temática tendrán secuencias frecuentes maximales iguales. Probablemente se trate de secuencias frecuentes maximales de longitud corta. Sería similar a hacer la clasificación utilizando n-gramas, pero con la ventaja de que no tienen por qué tener una longitud pre-fijada, sino que puede ser variable para cada clase y para cada secuencia. Además, se puede hacer uso de la *capacida de salto* de modo que las secuencias no tienen que ser exactamente iguales, sino que pueden incluir palabras intercaladas.

Del mismo modo, se puede hacer una clasificación no-temática, que se haga según los estilos de redacción. De este modo, se pueden utilizar las secuencias maximales para determinar a qué autor pertenece un texto anónimo. Puesto que los sustantivos y términos que aparecen en un texto están muy relacionados con la temática, no nos aportarán mucha información sobre el estilo. Para el estilo, las secuencias maximales que mejor lo describirán serán las frases hechas, las maneras de conectar ideas, etc. por eso es interesante efectuar la búsqueda sobre el corpus completo, sin hacer limpieza de *stop words*.

Por otro lado, es muy interesante el uso de este algoritmo para la detección de plagio. Como hemos demostrado en los experimentos realizados, es sencillo establecer la similitud entre dos (o más) textos dados. Dependiendo de las características de los textos a analizar y del dominio del problema, se puede establecer un umbral de confianza, por debajo del cual no se considerará plagio. Cuando la similitud entre dos documentos sobrepase el umbral de confianza, se podrá afirmar que ha habido plagio. Es especialmente interesante su uso para la detección de plagio porque no exige a la igualdad exacta de las secuencias, sino que es flexible hasta un cierto punto puesto que incluye la capacidad de salto o gap explicada en la descripción del algoritmo. De este modo, se puede detectar plagio aún a pesar de que se haya intentado esconder insertando o eliminando palabras en medio de las secuencias.

A pesar de todo, este algoritmo tiene una limitación. Sólo encuentra una secuencia maximal si al menos en uno de los documentos aparece tal cual. Es decir, si se cambia una palabra por un sinónimo, no sería capaz de detectar la secuencia porque, pese a que sí que es capaz de dejar el gap entre las palabra, no hay un documento que contenga la secuencia tal cual. Esto se ver más claro con un ejemplo.

Si tenemos tres documentos tales que:

Doc. 1	*Si sigue así el tiempo, el niño pequeño no vendrá.*
Doc. 2	*Si sigue así el tiempo, el niño inglés no vendrá.*
Doc. 3	*Si sigue así el tiempo, el niño no vendrá.*

Tabla 3 - Documentos de ejemplo

Al ejecutar el algoritmo sobre los dos primeros documentos de la Tabla 3, se localizaría una secuencia 2-frecuente maximal: "*Si sigue así el tiempo, el niño"*.

Al insertar el tercer documento en la colección y ejecutar el algoritmo, con los tres documentos, y permitiendo un gap de 1 palabra, se localizaría una secuencia 3-frecuente maximal: "*Si sigue así el tiempo, el niño no vendrá"*, que incluye la frase entera. Es decir, el algoritmo es capaz de detectar una secuencia maximal, siempre y cuando aparezca "tal cual" en al menos uno de los documentos.

Como trabajo futuro, se pretende mejorar el algoritmo de modo que no exista esta limitación y pueda encontrar secuencias frecuentes maximales sin necesidad de que se repitan literalmente en al menos uno de los documentos. De este modo, en el ejemplo anterior, se detectaría la frase: "*Si sigue así el tiempo, el niño no vendrá"*, con los documentos 1 y 2, sin necesidad de que esté también el documento 3.

También se pretende aplicar el algoritmo para tareas tales como el análisis de textos de biomedicina para localizar secuencias frecuentes maximales pertenecientes a proteínas. Del mismo modo, se pretende aplicar este algoritmo para el análisis de vandalismo en la web, concretamente en las revisiones de Wikipedia, de modo que se puedan establecer técnicas para detectar cuando un artículo ha sido atacado con blanqueo, u otros tipos de vandalismo.

Trabajos citados

Ahonen-Myka, H. 2002. Discovery of Frequent Word Sequences in Text. En *Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery* (September 16 - 19, 2002). D. J. Hand, N. M. Adams, and R. J. Bolton, Eds. Lecture Notes In Computer Science, vol. 2447. Springer-Verlag, Londres, págs 180-189.

Antunes, C. & Oliveira, A. (2004). *Sequential Pattern Mining Algorithms: Trade-offs between Speed and Memory*. 2nd Workshop on Mining Graphs, Trees and Seq.

Barrón-Cedeño A., Eiselt A., & Rosso, P. 2009. Monolingual Text Similarity Measures: A comparison of Models over Wikipedia Articles Revisions. En *Proceedings of ICON-2009: 7th International Conference on Natural Language Processing*. India.

Barrón Cedeño, L. A. (2008). *Detección automática de plagio en texto*. Tesis del Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital. Universidad Politécnica de Valencia.

Becher, V., Deymonnaz, A., & Heiber, P. (2008). *Efficient computation of all perfect repeats in genomicsequences of up to half a gigabyte, with a case study on the Human genome*. Bioinformatics, vol. 25, págs. 1746-1753.

Coyotl-Morales, R. Villaseñor-Pineda, L., Montes-y-Gómez, M. & Rosso, P. 2006. Authorship attribution using word sequences. In *Proceedings of the Iberoamerican Congress on Pattern Recognition, CIARP*, Cancún, México, págs. 844-853.

Fayyad, G., Piatetsky-Shapiro, & Smyth, P. 1996. *From Data Mining to Knowledge Discovery in Databases*. American Association for Artificial Intelligence, vol. 17, págs 37-54.

García-Hernandez, R. A., Martínez-Trinidad, J., & Carrasco-Ochoa, J. A. 2006. A New Algorithm for Fast Discovery of Maximal Sequential Patterns in a Document Collecion. En *International Conference on Computational Linguistics and text Processing, CICLing-2006*. Lecture Notes in Computer Science, vol. 4225, Springer-Verlag, London, págs 514-523.

Hoad, T. C., & Zobel, J. 2003. Methods for identifying versioned and plagiarized documents. *Journal of the American Society for Information Science and Technology*, vol. 54, issue 3, John Wiley & Sons, Inc., New York, págs 203-215.

Potthast, M., Stein, B., & Gerling, R. (2008). *Automatic Vandalism Detection in Wikipedia.* C. Macdonald et al. (Eds.): ECIR 2008, LNCS 4956, pp. 663–668.

Tan, A. (1999). *Text Mining: The state of the art and the challenges*. Proc. of the Workshop Knowledge Discovery from advanced Databases PAKDDD–99.

Anexos

Anexo A: Manual de Uso

El programa[3] tiene una interfaz gráfica para facilitar su uso, pero también tiene la posibilidad de ejecutarse desde la línea de comandos. A continuación, vamos a explicar cómo proceder en ambos casos.

Interfaz gráfica de usuario

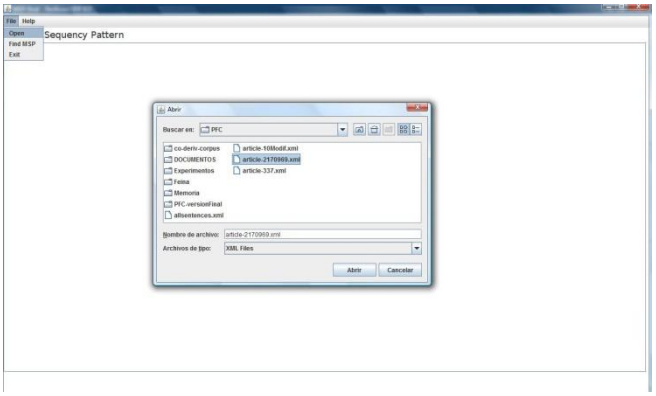

Figura 16 - Captura de la interfaz gráfica de usuario (1)

Paso 1: Cargar colección de textos

Seleccionar opción Open del menú File. Seleccionar el documento xml a cargar.

El documento xml tiene que contener todos los documentos en el siguiente formato. Cada documento estará delimitado por la etiqueta <document> y todos los documentos estarán agrupados con la etiqueta <documents>. Ambas con sus respectivos cierres. Si el documento contiene más etiquetas no será un problema, y no se tendrán en cuenta.

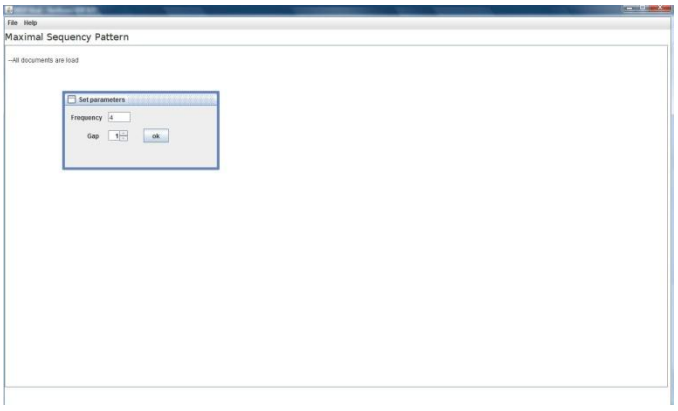

Figura 17 - Captura de la interfaz gráfica de usuario (2)

Paso 2: Selección de parámetros

Una vez nos aparece en pantalla el mensaje de que todos los documentos han sido cargados satisfactoriamente, procederemos a la selección de parámetros. Para ello seleccione la opción FindMSP del menú File. Indique los valores deseados para frecuencia y gap. Al pulsar OK, el algoritmo empezará su ejecución.

Línea de comandos

[3] El programa, así como esta memoria, se encuentran disponible en la web del Natural Language Engineering Lab. http://users.dsic.upv.es/grupos/nle/downloads

También es posible ejecutar el programa desde la línea de comandos. Para ello se proporciona la clase CLI. La instrucción será:

```
$java msp.CLI <nombrefichero.xml> <freq> <step>
```

Por ejemplo:

```
$java msp.CLI documentos.xml 7 2
```

Salida del programa

En ambos casos, tanto por línea de comandos como por interfaz gráfica, el programa almacena los resultados en ficheros de texto plano. Los nombres de los ficheros generados se establecen a partir del documento original, es decir, del nombre del documento xml que hemos cargado. Concretamente, se generan 3 ficheros:

Res<nombrefichero>.txt

Este primer fichero incluye todas las secuencias maximales encontradas, y para cada documento, la secuencia de posiciones de las palabras dentro del documento que darían lugar a esa secuencia.

StatRes<nombrefichero>.txt

Este fichero incluye las secuencias maximales encontradas en forma de estadística. Indica para cada longitud cuantas secuencias maximales se han encontrado, indicando las veces que aparece cada una de ellas y el texto de la secuencia maximal.

GraficaRes<nombrefichero>.txt

Este fichero pretende facilitar la tarea en caso de que se necesite dibujar gráficas con los datos obtenidos. Este fichero indica la cantidad de secuencias maximales encontradas para cada una de las longitudes. En sus dos últimas líneas, indica el tiempo de ejecución y el *total* correspondiente a la fórmula utilizada para el cálculo de la similitud (nótese que faltará normalizarlo dividiendo por el número total de palabras que contiene el documento más largo); así como se ha explicado en el apartado 1.3 del Proyecto Final de Carrera.

En el Anexo B se puede ver un ejemplo de los ficheros generados para el segundo de los experimentos propuestos.

Anexo B: Salida generada para el experimento de detección de secuencias maximales incluyendo huecos entre palabras.

GraficaResExperimento2.txt

```
Long: 403 Cant: 1
Total: 405
El tiempo de ejecución ha sido :0 min. aprox
```

StatResExperimento2.txt

Longitud	NumApariciones	Cadena
403	5	the usual way in which we plan today for tomorrow is in yesterday's vocabulary we do so because we try to get away with the concepts we are familiar with and that have acquired their meanings in our past experience of course the words and the concepts don't quite fit because our future differs from our past but then we stretch them a little bit linguists are quite familiar with the phenomenon that the meanings of words evolve over time but also know that this is a slow and gradual process it is the most common way of trying to cope with novelty by means of metaphors and analogies we try to link the new to the old the novel to the familiar under sufficiently slow and gradual change it works reasonably well in the case of a sharp discontinuity however the method breaks down though we may glorify it with the name common sense our past experience is no longer relevant the analogies become too shallow and the metaphors become more misleading than illuminating this is the situation that is characteristic for the radical novelty coping with radical novelty requires an orthogonal method one must consider one's own past the experiences collected and the habits formed in it as an unfortunate accident of history and one has to approach the radical novelty with a blank mind consciously refusing to try to link it with what is already familiar because the familiar is hopelessly inadequate one has with initially a kind of split personality to come to grips with a radical novelty as a dissociated topic in its own right coming to grips with a radical novelty amounts to creating and learning a new foreign language that cannot be translated into one's mother tongue anyone who has learned quantum mechanics knows what i am talking about needless to say adjusting to radical novelties is not a very popular activity for it requires hard work for the same reason the radical novelties themselves are unwelcome by now you may well ask why i have paid so much attention to and have spent so much eloquence on such a simple and obvious notion as the radical novelty my reason is very simple radical novelties are so disturbing that they tend to be suppressed or ignored to the extent that even the possibility of their existence in general is more often denied than admitted

ResExperimento2.txt

```
MaxSeqPattern 1 of Longitud: 403
Num. Apariciones: 5
CADENA: the usual way in which we plan today for tomorrow is in yesterday's
vocabulary we do so because we try to get away with the concepts we are
familiar with and that have acquired their meanings in our past experience of
course the words and the concepts don't quite fit because our future differs
from our past but then we stretch them a little bit linguists are quite
familiar with the phenomenon that the meanings of words evolve over time but
```

also know that this is a slow and gradual process it is the most common way of trying to cope with novelty by means of metaphors and analogies we try to link the new to the old the novel to the familiar under sufficiently slow and gradual change it works reasonably well in the case of a sharp discontinuity however the method breaks down though we may glorify it with the name common sense our past experience is no longer relevant the analogies become too shallow and the metaphors become more misleading than illuminating this is the situation that is characteristic for the radical novelty coping with radical novelty requires an orthogonal method one must consider one's own past the experiences collected and the habits formed in it as an unfortunate accident of history and one has to approach the radical novelty with a blank mind consciously refusing to try to link it with what is already familiar because the familiar is hopelessly inadequate one has with initially a kind of split personality to come to grips with a radical novelty as a dissociated topic in its own right coming to grips with a radical novelty amounts to creating and learning a new foreign language that cannot be translated into one's mother tongue anyone who has learned quantum mechanics knows what i am talking about needless to say adjusting to radical novelties is not a very popular activity for it requires hard work for the same reason the radical novelties themselves are unwelcome by now you may well ask why i have paid so much attention to and have spent so much eloquence on such a simple and obvious notion as the radical novelty my reason is very simple radical novelties are so disturbing that they tend to be suppressed or ignored to the extent that even the possibility of their existence in general is more often denied than admitted

Doc4: [[1, 25, 43, 46, 71, 74, 94, 114, 117, 119, 122, 135, 142, 152, 163,
169, 178, 184, 201, 205, 221, 241, 328, 331, 366, 391, 395], [2], [3], [4],
[5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18],
[19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31],
[32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44],
[45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57],
[58], [59], [60], [61], [62], [63], [64], [65], [66], [67], [68], [69], [70],
[71], [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82], [83],
[84], [85], [86], [87], [88], [89], [90], [91], [92], [93], [94], [95], [96],
[97], [98], [99], [100], [101], [102], [103], [104], [105], [106], [107],
[108], [109], [110], [111], [112], [113], [114], [115], [116], [117, 119],
[118], [119], [120], [121], [122], [123], [124], [125], [126], [127], [128],
[129], [130], [131], [132], [133], [134], [135], [136], [137], [138], [139],
[140], [141], [142], [143], [144], [145], [146], [147], [148], [149], [150],
[151], [152], [153], [154], [155], [156], [157], [158], [159], [160], [161],
[162], [163], [164], [165], [166], [167], [168], [169], [170], [171], [172],
[173], [174], [175], [176], [177], [178], [179], [180], [181], [182], [183],
[184], [185], [186], [187], [188], [189], [190], [191], [192], [193], [194],
[195], [196], [197], [198], [199], [200], [201], [202], [203], [204], [205],
[206], [207], [208], [209], [210], [211], [212], [213], [214], [215], [216],
[217], [218], [219], [220], [221], [222], [223], [224], [225], [226], [227],
[228], [229], [230, 232], [231], [232], [233], [234], [235], [236], [237],
[238], [239], [240], [241], [242], [243], [244], [245], [246], [247], [248],
[249], [250], [251], [252], [253], [254], [255, 257], [256], [257], [258],
[259], [260], [261], [262], [263], [264], [265], [266], [267], [268], [269],
[270], [271], [272], [273], [274], [275], [276], [277], [278], [279], [280],
[281], [282], [283], [284], [285], [286], [287], [288], [289], [290], [291],
[292], [293], [294], [295], [296], [297], [298], [299], [300], [301], [302],
[303], [304], [305], [306], [307], [308], [309], [310], [311], [312], [313],
[314], [315], [316], [317], [318], [319], [320], [321], [322], [323], [324],
[325], [326], [327], [328], [329], [330], [331], [332], [333], [334], [335],
[336], [337], [338], [339], [340], [341], [342], [344], [345], [346], [347],
[348], [349], [350], [351], [352], [353], [354], [355], [356], [357], [358],
[359], [360], [361], [362], [363], [364], [365], [366], [367], [368], [369],
[370], [371], [373], [374], [375], [376], [377], [378], [379], [380], [381],
[382], [383], [384], [385], [386], [389], [390], [391], [392], [393], [394],
[395], [396], [397], [398], [399], [400], [401], [402], [403], [404], [405],
[406], [407]]

Doc3: [[1, 25, 43, 46, 71, 74, 94, 114, 117, 119, 122, 135, 142, 152, 163,
169, 178, 184, 202, 208, 224, 246, 337, 340, 375, 397, 401], [2], [3], [4],
[5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18],
[19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31],

[32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44],
[45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57],
[58], [59], [60], [61], [62], [63], [64], [65], [66], [67], [68], [69], [70],
[71], [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82], [83],
[84], [85], [86], [87], [88], [89], [90], [91], [92], [93], [94], [95], [96],
[97], [98], [99], [100], [101], [102], [103], [104], [105], [106], [107],
[108], [109], [110], [111], [112], [113], [114], [115], [116], [117, 119],
[118], [119], [120], [121], [122], [123], [124], [125], [126], [127], [128],
[129], [130], [131], [132], [133], [134], [135], [136], [137], [138], [139],
[140], [141], [142], [143], [144], [145], [146], [147], [148], [149], [150],
[151], [152], [153], [154], [155], [156], [157], [158], [159], [160], [161],
[162], [163], [164], [165], [166], [167], [168], [169], [170], [171], [172],
[173], [174], [175], [176], [177], [178], [179], [180], [181], [182], [183],
[184], [185], [186], [187], [188], [189], [190], [191], [192], [193], [195],
[196], [197], [198], [199], [200], [201], [202], [203], [204], [207], [208],
[209], [210], [211], [212], [213], [214], [215], [216], [217], [218], [219],
[220], [221], [222], [223], [224], [225], [226], [227], [228], [229], [232],
[233], [234], [235, 237], [236], [237], [238], [239], [240], [241], [242],
[243], [244], [245], [246], [247], [248], [249], [250], [253], [254], [255],
[256], [257], [258], [259], [260], [261], [262, 264], [263], [264], [265],
[266], [267], [268], [269], [270], [271], [272], [273], [274], [275], [276],
[277], [278], [279], [280], [281], [282], [283], [284], [285], [286], [287],
[288], [289], [290], [291], [292], [293], [294], [295], [296], [297], [298],
[299], [300], [301], [302], [303], [304], [305], [306], [307], [308], [309],
[312], [313], [314], [315], [316], [317], [318], [319], [320], [321], [322],
[323], [324], [325], [326], [327], [328], [329], [330], [331], [332], [333],
[334], [335], [336], [337], [338], [339], [340], [341], [342], [343], [344],
[346], [347], [348], [349], [350], [351], [352], [353], [354], [355], [356],
[357], [358], [359], [360], [361], [362], [363], [364], [365], [366], [367],
[368], [369], [370], [371], [372], [373], [374], [375], [376], [377], [378],
[379], [380], [381], [382], [383], [384], [385], [386], [387], [388], [389],
[390], [391], [392], [393], [394], [395], [396], [397], [398], [399], [400],
[401], [402], [403], [404], [405], [406], [407], [408], [409], [410], [411],
[412], [413]]

Doc2: [[1, 25, 45, 49, 74, 77, 99, 119, 122, 124, 127, 140, 147, 157, 168,
174, 183, 189, 206, 210, 226, 246, 333, 336, 370, 392, 396], [2], [3], [4],
[5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18],
[19], [20], [21], [22], [23], [24], [25], [26], [29], [30], [31], [32], [33],
[34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [47],
[48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58], [59], [60],
[61], [62], [63], [64], [65], [66], [67], [68], [69], [70], [71], [72], [73],
[74], [75], [76], [77], [78], [79], [80], [82], [83], [84], [85], [86], [87],
[88], [89], [90], [91], [92], [93], [95], [96], [97], [98], [99], [100],
[101], [102], [103], [104], [105], [106], [107], [108], [109], [110], [111],
[112], [113], [114], [115], [116], [117], [118], [119], [120], [121], [122,
124], [123], [124], [125], [126], [127], [128], [129], [130], [131], [132],
[133], [134], [135], [136], [137], [138], [139], [140], [141], [142], [143],
[144], [145], [146], [147], [148], [149], [150], [151], [152], [153], [154],
[155], [156], [157], [158], [159], [160], [161], [162], [163], [164], [165],
[166], [167], [168], [169], [170], [171], [172], [173], [174], [175], [176],
[177], [178], [179], [180], [181], [182], [183], [184], [185], [186], [187],
[188], [189], [190], [191], [192], [193], [194], [195], [196], [197], [198],
[199], [200], [201], [202], [203], [204], [205], [206], [207], [208], [209],
[210], [211], [212], [213], [214], [215], [216], [217], [218], [219], [220],
[221], [222], [223], [224], [225], [226], [227], [228], [229], [230], [231],
[232], [233], [234], [235, 237], [236], [237], [238], [239], [240], [241],
[242], [243], [244], [245], [246], [247], [248], [249], [250], [251], [252],
[253], [254], [255], [256], [257], [258], [259], [260, 262], [261], [262],
[263], [264], [265], [266], [267], [268], [269], [270], [271], [272], [273],
[274], [275], [276], [277], [278], [279], [280], [281], [282], [283], [284],
[285], [286], [287], [288], [289], [290], [291], [292], [293], [294], [295],
[296], [297], [298], [299], [300], [301], [302], [303], [304], [305], [306],
[307], [308], [309], [310], [311], [312], [313], [314], [315], [316], [317],
[318], [319], [320], [321], [322], [323], [324], [325], [326], [327], [328],
[329], [330], [331], [332], [333], [334], [335], [336], [337], [338], [339],
[340], [341], [342], [343], [344], [345], [346], [347], [348], [349], [350],
[351], [352], [353], [354], [355], [356], [357], [358], [359], [360], [361],

[362], [363], [364], [365], [366], [367], [368], [369], [370], [371], [372],
[373], [374], [375], [376], [377], [378], [379], [380], [381], [382], [383],
[384], [385], [386], [387], [388], [389], [390], [391], [392], [393], [394],
[395], [396], [397], [398], [399], [400], [401], [402], [403], [404], [405],
[406], [407], [408]]

Doc1: [[1, 25, 43, 46, 71, 74, 94, 114, 118, 121, 124, 138, 145, 156, 168,
174, 183, 189, 207, 211, 227, 247, 334, 337, 371, 393, 397], [2], [3], [4],
[5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18],
[19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31],
[32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44],
[45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57],
[58], [59], [60], [61], [62], [63], [64], [65], [66], [67], [68], [69], [70],
[71], [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82], [83],
[84], [85], [86], [87], [88], [89], [90], [91], [92], [93], [94], [95], [96],
[97], [98], [99], [100], [101], [102], [103], [104], [105], [106], [107],
[108], [109], [110], [111], [112], [113], [114], [115], [117], [118], [119],
[121], [122], [123], [124], [125], [126], [127], [128], [129], [131], [132],
[133], [134], [135], [136], [137], [138], [139], [140], [141], [142], [143],
[144], [145], [146], [147], [148], [150], [151], [152], [153], [154], [155],
[156], [157], [158], [159], [160], [161], [162], [163], [164], [165], [167],
[168], [169], [170], [171], [172], [173], [174], [175], [176], [177], [178],
[179], [180], [181], [182], [183], [184], [185], [186], [187], [188], [189],
[190], [191], [193], [194], [195], [196], [197], [198], [199], [200], [201],
[202], [203], [204], [205], [206], [207], [208], [209], [210], [211], [212],
[213], [214], [215], [216], [217], [218], [219], [220], [221], [222], [223],
[224], [225], [226], [227], [228], [229], [230], [231], [232], [233], [234],
[235], [236, 238], [237], [238], [239], [240], [241], [242], [243], [244],
[245], [246], [247], [248], [249], [250], [251], [252], [253], [254], [255],
[256], [257], [258], [259], [260], [261, 263], [262], [263], [264], [265],
[266], [267], [268], [269], [270], [271], [272], [273], [274], [275], [276],
[277], [278], [279], [280], [281], [282], [283], [284], [285], [286], [287],
[288], [289], [290], [291], [292], [293], [294], [295], [296], [297], [298],
[299], [300], [301], [302], [303], [304], [305], [306], [307], [308], [309],
[310], [311], [312], [313], [314], [315], [316], [317], [318], [319], [320],
[321], [322], [323], [324], [325], [326], [327], [328], [329], [330], [331],
[332], [333], [334], [335], [336], [337], [338], [339], [340], [341], [342],
[343], [344], [345], [346], [347], [348], [349], [350], [351], [352], [353],
[354], [355], [356], [357], [358], [359], [360], [361], [362], [363], [364],
[365], [366], [367], [368], [369], [370], [371], [372], [373], [374], [375],
[376], [377], [378], [379], [380], [381], [382], [383], [384], [385], [386],
[387], [388], [389], [390], [391], [392], [393], [394], [395], [396], [397],
[398], [399], [400], [401], [402], [403], [404], [405], [406], [407], [408],
[409]]

Doc0: [[1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13],
[14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26],
[27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39],
[40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52],
[53], [54], [55], [56], [57], [58], [59], [60], [61], [62], [63], [64], [65],
[66], [67], [68], [69], [70], [71], [72], [73], [74], [75], [76], [77], [78],
[79], [80], [81], [82], [83], [84], [85], [86], [87], [88], [89], [90], [91],
[92], [93], [94], [95], [96], [97], [98], [99], [100], [101], [102], [103],
[104], [105], [106], [107], [108], [109], [110], [111], [112], [113], [114],
[115], [116], [117], [118], [119], [120], [121], [122], [123], [124], [125],
[126], [127], [128], [129], [130], [131], [132], [133], [134], [135], [136],
[137], [138], [139], [140], [141], [142], [143], [144], [145], [146], [147],
[148], [149], [150], [151], [152], [153], [154], [155], [156], [157], [158],
[159], [160], [161], [162], [163], [164], [165], [166], [167], [168], [169],
[170], [171], [172], [173], [174], [175], [176], [177], [178], [179], [180],
[181], [182], [183], [184], [185], [186], [187], [188], [189], [190], [191],
[192], [193], [194], [195], [196], [197], [198], [199], [200], [201], [202],
[203], [204], [205], [206], [207], [208], [209], [210], [211], [212], [213],
[214], [215], [216], [217], [218], [219], [220], [221], [222], [223], [224],
[225], [226], [227], [228], [229], [230], [231], [232], [233], [234], [235],
[236], [237], [238], [239], [240], [241], [242], [243], [244], [245], [246],
[247], [248], [249], [250], [251], [252], [253], [254], [255], [256], [257],
[258], [259], [260], [261], [262], [263], [264], [265], [266], [267], [268],

[269], [270], [271], [272], [273], [274], [275], [276], [277], [278], [279],
[280], [281], [282], [283], [284], [285], [286], [287], [288], [289], [290],
[291], [292], [293], [294], [295], [296], [297], [298], [299], [300], [301],
[302], [303], [304], [305], [306], [307], [308], [309], [310], [311], [312],
[313], [314], [315], [316], [317], [318], [319], [320], [321], [322], [323],
[324], [325], [326], [327], [328], [329], [330], [331], [332], [333], [334],
[335], [336], [337], [338], [339], [340], [341], [342], [343], [344], [345],
[346], [347], [348], [349], [350], [351], [352], [353], [354], [355], [356],
[357], [358], [359], [360], [361], [362], [363], [364], [365], [366], [367],
[368], [369], [370], [371], [372], [373], [374], [375], [376], [377], [378],
[379], [380], [381], [382], [383], [384], [385], [386], [387], [388], [389],
[390], [391], [392], [393], [394], [395], [396], [397], [398], [399], [400],
[401], [402], [403]]