TEMA 2. ÍNDICES INVERTIDOS. TÉRMINOS Y CONSULTAS

Contenidos basados en los materiales de otros cursos como los de Manning y Baeza

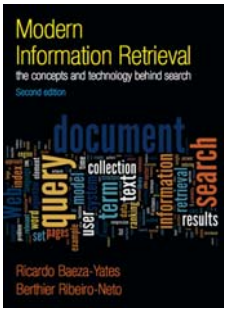
Contenidos

1. Introducción
2. Obtener el vocabulario de términos
   2.1. Documentos
   2.2. Tokenización
   2.3. Vocabulario de términos
3. Postings
   3.1. Skip list
   3.2. Postings posicionales y consultas de Sintagmas

Bibliografía

A Introduction to Information Retrieval:
Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze.
Cambridge University Press, ***2009***.
Capítulo 2

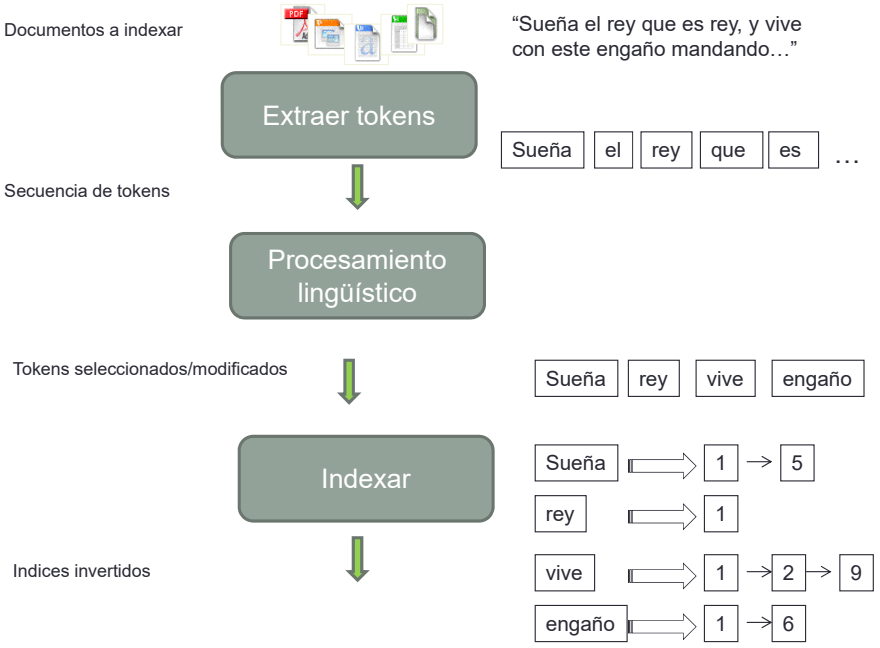
Modern Information Retrieval:
Ricardo Baeza-Yates and Berthier Ribeiro-Neto
Addison Wesley, First printed ***1999***
Capítulos 6-7



1. INTRODUCCIÓN

Recordemos …

Etapas de construcción de índices invertidos



Los pasos principales para su creación son:

1. En un primer paso se analiza cada documento de la colección y se extraen sus elementos, que llamaremos tokens, generalmente separados por blancos o signos de puntuación y se obtiene una secuencia de tokens por documento.

2. En un segundo paso se realiza un preprocesamiento lingüístico, produciendo una lista de tokens normalizados, que son los términos de indexación.

3. Finalmente se procede a realizar la indexación.

Objetivo del Tema: conocer una forma básica de construir un índice,

1) Preproceso para obtener los términos:
- Documentos
- Tokenización
- ¿Qué términos poner en el índice?

2) Postings Lists:
- Mejora de la operación “Intersección”: Skip lists
- Postings posicionales y consultas de sintagmas

2.OBTENER EL VOCABULARIO DE TÉRMINOS

2.1. Documentos

2.2. Tokenización

2.3. Vocabulario de términos

2.1. Documentos

Decisiones respecto al análisis de los documentos

- ¿De qué tipo de documento se trata (pdf, txt, html, avi..)?
- ¿En qué lengua(-s) está escrito?
- ¿Qué conjunto de caracteres usa?

Son tareas que pueden abordarse como un problema de clasificación, pero se suelen resolver de forma heurística o utilizando los metadatos.

Los documentos digitales, que son la entrada al proceso de indexación, son bytes en un fichero o en un servidor web.

Hay que:

1. Convertir esa secuencia de bytes en una secuencia de caracteres
2. Escoger una unidad de documento para la indexación, segmentar un documentos en varios, agrupar en uno solo varios ficheros
3. Interpretar la secuencia de caracteres a partir de diferentes formatos (doc, pdf, latex, html, xml, zip, …)

2.1. Documentos

Decisiones respecto al documento como unidad

- La colección de documentos puede incluir documentos en diferentes lenguas
  ⇨ Un índice simple debería incluir términos en diferentes lenguas.
- Hay documentos que contienen componentes en diferentes lenguas o formatos.
  ⇨ Un email en francés con un pdf adjunto en alemán.

¿Cuál es la unidad “Documento” (fichero o grupo de ficheros; e-mail; o e-mail y sus ficheros adjuntos,….)?

Un archivo de correo electrónico Unix tradicional (formato mbox) almacena una secuencia de mensajes de correo electrónico (una carpeta de correo electrónico) en un archivo, pero es posible que se desee considerar cada mensaje de correo electrónico como un documento separado. Muchos mensajes de correo electrónico contienen documentos adjuntos, por lo que es posible que se desee considerar el mensaje de correo electrónico y cada archivo adjunto como documentos separados. Si un mensaje de correo electrónico tiene un archivo zip adjunto, es posible que desee decodificar el archivo zip y considerar cada archivo que contiene como un documento separado.

Yendo en la dirección opuesta, varias piezas de software web (como latex2html) toman cosas que se podrían considerar como un solo documento (por ejemplo, un archivo de Powerpoint o un documento de LATEX) y las dividen en páginas HTML separadas para cada diapositiva o subsección, almacenados como archivos separados. En estos casos, es posible que desee combinar varios archivos en un solo documento.

Para una colección de libros, normalmente sería una mala idea indexar un libro completo como documento. Una búsqueda de juguetes chinos puede generar un libro que mencione a China en el primer capítulo y juguetes en el último capítulo, pero esto no lo hace relevante para la consulta. En cambio, es posible que deseemos indexar cada capítulo o párrafo como un mini documento. Es más probable que las coincidencias sean relevantes, y dado que los documentos son más pequeños, será mucho más fácil para el usuario encontrar los pasajes relevantes en el documento.

Formatos de documentos

Texto

- Representación de los códigos de caracteres en formato binario a través de los distintos “esquemas de codificación”:

EBCDIC (7 bits), ASCII (8 bits), UNICODE (16 bits)

- Recuperación desde diversos formatos de documentos de texto (doc, pdf, html, txt, rtf, ps,..)

El primer paso del procesamiento es convertir esta secuencia de bytes en una secuencia lineal de caracteres.

Para el caso de texto en inglés sin formato en codificación ASCII este proceso es trivial, sin embargo no siempre es así. La secuencia de caracteres puede estar codificada en alguno de los diferentes esquemas de codificación de un solo byte o multibyte, como Unicode UTF-8. Necesitamos determinar la codificación correcta. Esto puede considerarse como un problema de clasificación de aprendizaje automático, pero a menudo se maneja mediante métodos heurísticos, selección de usuario o mediante el uso de metadatos de documentos proporcionados.

Una vez que se determina la codificación, decodificamos la secuencia de bytes a una secuencia de caracteres. Podríamos guardar la opción de codificación porque brinda alguna evidencia sobre el idioma en el que está escrito el documento.

Es posible que los caracteres deban decodificarse a partir de alguna representación binaria, como archivos Microsoft Word DOC y / o un formato comprimido como archivos zip.

Nuevamente, debemos determinar el formato del documento y luego se debe utilizar un decodificador apropiado.

Texto…..cont

Otros formatos de intercambio se usan para codificar correos electrónicos y comprimir textos.

- **Multipurpose Internet Mail Exchange (MIME):** para codificar email
- **ZIP**: *gzip* en Unix/linux, *Winzip* en Windows para comprimir texto.

=> Los sistemas de RI tienen que disponer de filtros y conversores de formato para acceder a los tipos de documentos más habituales.

Lenguajes de marcado

Sirven para incorporar información al texto, tal como estructura de la información, acciones de formato, semántica, atributos…

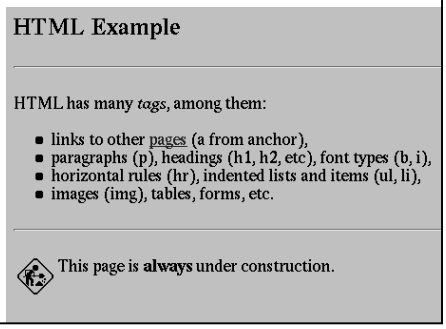
Algunos lenguajes conocidos son:

- SGML: Standard Generalized Markup Language
- XML: eXtensible Markup Language
- HTML: Hyper Text Markup Language

En documentos XML, HTML y otros, se hace necesaria la extracción de la parte textual que es la que se va a definir como documento a indexar.

Ejemplo de lenguajes de marcado. HTML

```
<html><head>
<title>HTML Example</title>
<meta name=rby content="Just an example">
</head>
<body>
<h1>HTML Example</h1>
<p><hr><p>HTML has many <i>tags</i>, among them:
<ul>
<li> links to other <a href=example.html>pages</a> (a from anchor),
<li> paragraphs (p), headings (h1, h2, etc), font types (b, i),
<li> horizontal rules (hr), indented lists and items (ul, li),
<li> images (img), tables, forms, etc.
</ul>
<p><hr><p><img align=left src="at_work.gif">
This page is <b>always</b> under construction.
</body></html>
```



Gran parte de los motores de búsqueda necesitan almacenar los documentos una vez convertidos a un formato común, para la preparación del índice.

También es posible que requiera un acceso rápido a los documentos, por ejemplo para construir los 'snippets'. Para ello se suelen usar bases de datos convencionales para almacenar los documentos.

2.2. Tokenización

Dada una secuencia de caracteres y una unidad de documento definida, la tokenización consiste en trocear la secuencia en piezas (tokens).

- Un token es una secuencia de caracteres
- Cada token es un candidato para ser una entrada del diccionario.
- ¿Son válidos todos los tokens?
- Ejemplo.

  Input: "***Friends, Romans, Countrymen***"
  Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***

14

Dada una secuencia de caracteres y una unidad de documento definida, la tokenización es la tarea de cortarlo en pedazos, llamados tokens, quizás al mismo tiempo desechando ciertos caracteres, como los signos de puntuación.

Llamaremos **token** a una instancia de una secuencia de caracteres en un documento.

Los agruparemos, eliminando repetidos e incluso definiendo equivalencias, y obtendremos los términos, que serán la unidad del diccionario.

Este proceso hay que realizarlo sobre los documentos en el proceso de indexación, pero también sobre las consultas (on-line).

Cada lengua tiene sus propias características.

2.2. Tokenización

Decisiones respecto a tokens

- ***Finland's capital*** →
  Finland? Finlands? Finland's?
- ***Hewlett-Packard*** → ***Hewlett*** y ***Packard*** como dos tokens?
  - ***state-of-the-art***
  - ***co-education***
  - ***lowercase***, ***lower-case***, ***lower case*** ?
- ***San Francisco***: como dos tokens?
- **Números, fechas:** *3/12/91, Mar. 12, 1991, 12/3/91, 55 B.C., B-52, (800) 234-2333.*

15

2.2. Tokenización

Algunos problemas con las distintas lenguas

Francés, Catalán:

- la expresión ***L'ensemble: L*** ? ***L'*** ? ***Le*** ?

Idiomas aglutinantes: Palabras largas

- **Alemán:***Donaudampfschifffahrtselektrizitätenhauptbetriebswerkbauunterbeamten gesellschaft* («Sociedad de funcionarios subordinados de la construcción de la fábrica principal de la electricidad para la navegación de barcos de vapor en el Danubio»)
- **Turco:** *muvaffakiyetsizleştiricileştiriveremeyebileceklerimizdenmişsinizcesine*
- **Chino y japonés:** no tienen espacio entre palabras

莎拉波娃现在居住在美国东南部的佛罗里达。

16

2.2. Tokenización

Algunos problemas con las distintas lenguas….cont

Árabe: El árabe se escribe de derecha a izquierda, pero ciertos elementos como los números se escriben de izquierda a derecha.

استقالت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

Lenguas que utilizan múltiples alfabetos

フォーチュン500社は情報不足のため時間あた$500K(約6,000万円)

17

En el último ejemplo, la secuencia de caracteres contiene caracteres japoneses mezclados con números arábigos, letras en alfabeto latino, etc.

Stopwords (palabras "muy" frecuentes)

Conjunto de palabras con poco significado.

Métodos:

- Crear listas "fijas" de palabras para cada idioma:
  Inglés: a, cannot, into, our, thus, about, co, is, ours, to, above, could, it, ourselves, together, ...
  Castellano: él, éstos, última, últimas, aún, actualmente, adelante, además, ahí, ahora, de, ...

- Crear listas de categorías léxicas: (preposiciones, artículos, conjunciones, pronombres, adverbios).

- Crear listas a partir de la frecuencia de las palabras en la colección de documentos.

A veces, algunas palabras extremadamente comunes que parecen tener poco valor para ayudar a seleccionar documentos que coincidan con las necesidades del usuario se excluyen por completo del vocabulario. Estas palabras se denominan palabras vacías (stopwords).

La estrategia general para determinar una lista de stopwords es ordenar los términos por frecuencia en la colección (el número total de veces que cada término aparece en la colección de documentos) y luego tomar los términos más frecuentes, a menudo filtrados manualmente por su contenido semántico en relación con el dominio de los documentos que se indexan, como una lista de stopwords, cuyos elementos luego se descartan durante la indexación.

2.2. Tokenización

Problemas al eliminar las **Stopwords**:

- Pueden ser necesarias:

  "*Rey **de** Dinamarca*" ,

  "*Vitamina **A**",*

  *"¿Qué significa **ESA**?",*

  "***to be or not to be***",

  "***En un** lugar **de La** Mancha*",….

- Las buenas técnicas de compresión hacen que el espacio necesario para incluir stopwords no sea muy grande.

El uso de una lista de stopwords reduce significativamente el número de postings que un sistema tiene que almacenar.

Sin embargo, esto no es útil para las búsquedas de sintagmas.

La consulta "Presidente de los Estados Unidos", que contiene dos stopwords, es más precisa que "Presidente" AND "Estados Unidos".

Es probable que se pierda el significado de "los vuelos a Londres" si se suprime la palabra "a".

- Lista de stopwords(castellano):

él ésta éstas éste éstos última últimas último últimos a añadió aún actualmente adelante además afirmó agregó ahí ahora al algún algo alguna algunas alguno algunos alrededor ambos ante anterior antes apenas aproximadamente aquí así aseguró aunque ayer bajo bien buen buena buenas bueno buenos cómo cada casi cerca cierto cinco comentó como con conocer consideró considera contra cosas creo cual cuales cualquier cuando cuanto cuatro cuenta da dado dan dar de debe deben debido decir dejó del demás dentro desde después dice dicen dicho dieron diferente diferentes dijeron dijo dio donde dos durante e ejemplo el ella ellas ello ellos embargo en encuentra entonces entre era eran es esa esas ese eso esos está están esta estaba estaban estamos estar estará estas este esto estos estoy estuvo ex existe existen explicó expresó fin fue fuera fueron gran grandes ha había habían haber habrá hace hacen hacer hacerlo hacia haciendo han hasta hay haya he hecho hemos hicieron hizo hoy hubo igual incluso indicó informó junto la lado las le les llegó lleva llevar lo los luego lugar más manera manifestó mayor me mediante mejor mencionó menos mi mientras misma mismas mismo mismos momento mucha muchas mucho muchos muy nada nadie ni ningún ninguna ningunas ninguno ningunos no nos nosotras nosotros nuestra nuestras nuestro nuestros nueva nuevas nuevo nuevos nunca o ocho otra otras otro otros para parece parte partir pasada pasado pero pesar poca pocas poco pocos podemos podrá podrán podría podrían poner por porque posible próximo próximos primer primera primero primeros principalmente propia propias propio propios pudo pueda puede pueden pues qué que quedó queremos quién quien quienes quiere realizó realizado realizar respecto sí sólo se señaló sea sean según segunda segundo seis ser será serán sería si sido siempre siendo siete sigue siguiente sin sino sobre sola solamente solas solo solos son su sus tal también tampoco tan tanto tenía tendrá tendrán tenemos tener tenga tengo tenido tercera tiene tienen toda todas todavía todo todos total tras trata través tres tuvo un una unas uno unos usted va vamos van varias varios veces ver vez y ya yo

- Lista de stopwords (inglés):

a about above across actually after again against ago all almost along already also although always among an and another any anyone around as at b back bad because before behind best better between big biggest both but by c cent complete d day down during e each early eight enough entire ep etc even ever every everything f face fact far fell few finally first five for found four from g good got h he held her here him himself his hour hours how however i idea if in including instead into it its itself j k l lack last later least led less little long longer lot m man many matter may me men miles million moment month months more morning most much my n near nearly necessary never night no nor not note nothing now o of off often on once one only or other others our out outside over own p page part past per perhaps place point proved q qm question r really recent recently reported round s same sec second section sense seven she short should showed since single six small so some soon still such t ten text than that the their them themselves then there these they thing things third this those though thought thousands three through thus time tiny to today together too took toward two u under until up upon us v very w warning way we week weeks well went were what when where whether which while who whom whose why will with without word words would x y year years yet you your z

2.3. Vocabulario de términos

Normalización

Las palabras se deben normalizar:
- al procesar los documentos (off-line),
- al procesar las consultas (on-line).

Por ejemplo, deben considerarse iguales U.S.A. y USA?

Término: Un término es una "palabra" (normalizada) que será una entrada en el diccionario del sistema de RI.

Habiendo dividido nuestros documentos (y también nuestra consulta) en tokens, el caso fácil es si los tokens en la consulta coinciden con los tokens en la lista de tokens del documento. Sin embargo, hay muchos casos en los que dos secuencias de caracteres no son exactamente iguales pero se requiere que se produzca una coincidencia. Por ejemplo, si se busca "U.S.A.", es posible que se desee encontrar también documentos que contengan "USA".

La normalización de tokens es el proceso de canonizar tokens para que se produzcan coincidencias a pesar de diferencias superficiales en las secuencias de caracteres de los tokens. La forma más estándar de normalizar es crear implícitamente clases de equivalencia, que normalmente reciben el nombre de un elemento del conjunto.

Una alternativa a la creación de clases de equivalencia es mantener relaciones entre tokens no normalizados. Este método se puede extender a listas de sinónimos construidas a mano, como "coche" y "automóvil".

Estas relaciones de términos se pueden lograr de dos maneras.

Una forma es indexar tokens no normalizados y mantener una lista de expansión de consultas de múltiples entradas de vocabulario a considerar para un determinado término de consulta. Entonces, la respuesta a un término de consulta es la unión de varias listas de postings. La alternativa es realizar la expansión durante la construcción del índice. Cuando el documento contiene "automóvil", también lo indexamos para "coche".

2.3. Vocabulario de términos

Decisiones sobre normalización

(siempre las mismas en la indexación y en la consulta)

Ejemplo.-

- Acentos: *résumé, Tübingen, acción.*
- Eliminar puntos: U.S.A. como USA
- Eliminar guiones (hyphens):

  anti-discriminatorio como antidiscriminatorio
- Eliminar mayúsculas:

  Excepciones (nombres? Instituciones? Países? …)

  Ejemplo: *General Motors*

Acentos y signos diacríticos. En muchos idiomas, los acentos y los signos diacríticos son una parte regular del sistema de escritura y distinguen diferentes sonidos. Ocasionalmente, algunas palabras se distinguen solamente por sus acentos o signos diacríticos. Incluso en inglés, a veces se usan palabras originarias de otras lenguas en las que aparecen acentos o diacríticos como “cliché” o “naïve”.

Mayúsculas y minúsculas. Una estrategia común es hacer case-folding , reduciendo todas las letras a minúsculas.

Parece una buena idea, ya que permite reunir instancias de palabras a principio de oración con otras en otras posiciones.

Puede ayudar también en la búsqueda ya que frecuentemente en las consultas los términos se escriben en minúsculas, aunque sean marcas (ferrari).

Sin embargo, esta estrategia equipara palabras que sería mejor mantener separadas. Muchos nombres propios se derivan de sustantivos comunes y, por lo tanto, se distinguen solamente por las mayúsculas. Este es el caso de empresas como General Motors, The Associated Press, o nombres de personas (Bush, Black).

2.3. Vocabulario de términos

Lematización

Eliminar las variantes/inflexiones de las palabras, almacenando sólo el lema o forma base.

Ejemplo:

come, comía, comerá → *comer*
am, are, is → *be*
car, cars, car's, cars' → *car*
amarillo, amarilla, amarillos → *amarillo*

Stemming

Reducir los términos a sus "raíces", eliminando los sufijos.

Ejemplo:
automate(s), automatic, automation → *automat*

Stemming generalmente se refiere a un proceso heurístico que corta los extremos de las palabras con el objetivo de generar una representación canónica de la misma. A menudo, incluye la eliminación de afijos derivados.

Lemmatization generalmente se refiere a generar la representación canónica con el uso de un vocabulario y análisis morfológico de palabras, normalmente con el objetivo de eliminar solo las terminaciones flexivas y devolver la forma base de una palabra, lo que se conoce como lema.

Dado el token "*saw*", el stemming devolverá solamente "s", mientras que la lematización intentaría devolver "*see*" o "*saw*" dependiendo de si el uso del token es como un verbo o un sustantivo.

2.3. Vocabulario de términos

Stemmer para el inglés: ***Algoritmo de Porter.***

Aplica reglas para eliminar los sufijos de las palabras.

• Grupos de Reglas aplicadas al sufijo más largo:

sses → *ss*	(caresses →caress)
ies → *i*	(ponies →poni)
ss → *ss*	(caress →caress)
s →	(cats →cat)

• Reglas dependientes de la longitud de las palabras:

(m>1) EMENT:
replacement → replac
cement → cement

El algoritmo de Porter consta de 5 fases de reducciones de palabras, aplicadas secuencialmente.

Dentro de cada fase hay varias convenciones para seleccionar reglas, como seleccionar la regla de cada grupo de reglas que se aplica al sufijo más largo. En la primera fase, esta convención se usa con el primer grupo de reglas.

Muchas de las reglas posteriores usan un concepto de la medida de una palabra, que verifica libremente el número de sílabas para ver si una palabra es lo suficientemente larga como para que sea razonable considerar la parte correspondiente de una regla como un sufijo en lugar de como parte de la raíz de una palabra.

2.3. Vocabulario de términos

Stemmer para el inglés: ***Algoritmo de Porter.***

The project is a final examination exercise in the Informatics Engineering and in both Diplomas. Consequently, the student is required to show that he or she is able to apply the knowledge acquired during his/her studies to typical informatics engineering situations, allowing for the different specific course objectives, which are defined in the curriculum. For the Informatics Engineering these are to provide the student with a broad base and professional skills in software engineering and information systems for organisations, informatics engineering, or industrial information technology…

2.3. Vocabulario de términos

Stemmer para el inglés: ***Algoritmo de Porter.***

the project is a final examin exercis in the informat engin and in both diploma. consequ, the student is requir to show that he or she is abl to appli the knowledg acquir dure hi/her studi to typic informat engin situat, allow for the differ specif cours object, which ar defin in the curriculum. for the informat engin these ar to provid the student with a broad base and profession skill in softwar engin and inform system for organis, informat engin, or industri inform technolog…

Resultado de aplicar el algoritmo para el texto de la diapositiva anterior.

Stemming: pros and cons

- Pros:
  - Mejora la efectividad de la recuperación, en cierta medida
  - Casi tan efectivo como la lematización
  - Necesita menos conocimiento de la lengua que la lematización
  - Más rápido, fácil de describir y de implementar
- Cons:
  - La salida no es legible
  - Puede reducir a la misma raíz palabras diferentes
    - Relative (family)
    - Relativity (physics)
    - Relativism (philosophy)

2.3. Vocabulario de términos

Enriquecimiento (consulta)

- Sinónimos
  gun, arm, weapon -> weapon
- Palabras relacionadas
  laptop -> portable computer, light PC
- Categorias (semantic web)
  Ski, soccer, marathon -> sports
- Información adicional (part of speech)
- Herramientas de Procesamiento del Lenguaje Natural

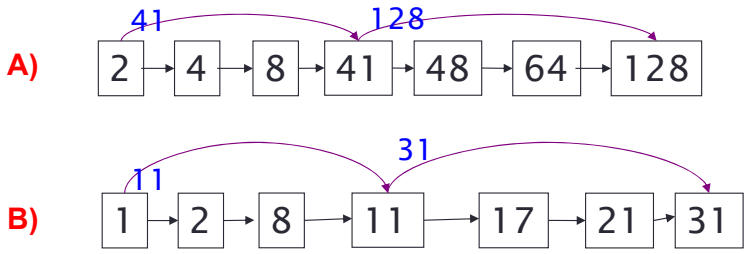
3. POSTINGS

3.1. Skip list

3.2. Posting posicionales y consultas de Sintagmas

3.1 Skip list

Modificar las posting list incluyendo "*skip pointers*".



- Supongamos que hemos explorado ambas listas hasta llegar a **8** en cada lista.
- Los siguientes elementos que tenemos son **41** y **11.**
- Como **11** es menor, tenemos que seguir en esa lista, pero podemos comparar con su *skip pointer* que vale 31 (y sigue siendo menor que 41) por lo que podemos saltar todo ese tramo de la lista y seguir por el 31.

Vamos a estudiar extensiones de las estructuras de datos para las postings lists y las formas de aumentar la eficiencia en su uso.

Revisemos la operación básica de intersección de postings lists: recorremos las dos listas simultáneamente, en tiempo lineal con el número total de entradas (postings).

Si las longitudes de la lista son 'm' y 'n', el coste en número de operaciones de la intersección es O (m + n).

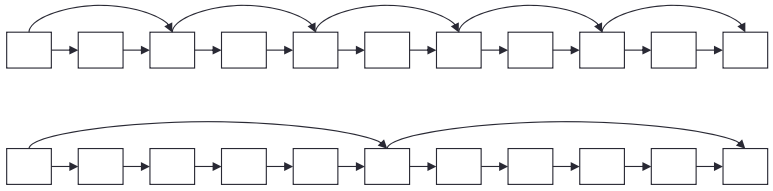
Una forma de hacerlo rápido es utilizar una *skip list* aumentando las postings lists con skip pointers (en el momento de la indexación), como se muestra en la Figura.

Los skip pointers son atajos que nos permiten evitar procesar partes de la postings list que no aparecerán en los resultados de búsqueda.

¿Dónde y cuántos skip pointers?

Equilibrio:

- Si hay muchos skip-pointers entonces es más fácil que se produzcan saltos, aunque serán cortos. Además hay un coste añadido en las comparaciones con los skip-pointers.
- Si hay pocos entonces hay menos comparaciones con los skip pointers y los saltos son mayores, pero es menos probable que se produzcan.



32

Un heurístico para la colocación skips, que ha demostrado funcionar bien en la práctica, es que para listas de longitud P, se usen raíz cuadrada de P skip pointers espaciados uniformemente.

Algoritmo rápido: operación "Intersección con skips" de posting lists

```
ALGORITMO INTERSECCION_CON_SKIPS (p1, p2)
respuesta ← {}
mientras No_FINAL( p1) AND No_FINAL( p2)
hacer si docID (p1) = docID (p2)
        entonces    Añadir (respuesta, docID (p1))
                    p1 ← Avanzar_Siguiente(p1)
                    p2 ← Avanzar_Siguiente(p2)
        sino si docID (p1) < docID (p2)
            entonces si Tiene_Skip(p1) AND (docID(skip(p1)) ≤ docID (p2))
                        entonces mientras Tiene_Skip(p1) AND (docID(skip(p1)) ≤ docID (p2))
                                    hacer p1 ← skip(p1)
                         sino p1 ← Avanzar_Siguiente(p1)
            sino si Tiene_Skip(p2) AND (docID(skip(p2)) ≤ docID (p1))
                        entonces mientras Tiene_Skip(p2) AND (docID(skip(p2)) ≤ docID (p1))
                                    hacer p2 ← skip(p2)
                         sino p2 ← Avanzar_Siguiente(p2)
```

En la diapositiva se muestra una versión del algoritmo de intersección de dos postings lists con skip pointers.

Ejercicio#1

Suponer una consulta de dos palabras (AND) que proporcionan las siguientes posting lists.

[4 6 10 12 14 16 18 20 22 32 47 81 120 122 157 180]

[47]

Cuántas comparaciones se realizan suponiendo:

- que no hay skip pointers
- que hay skip pointers de longitud 4

Ejercicio#2.

Los posting lists implementados con skip pointers, ¿son útiles para las operaciones?:

a) t1 and t2
b) t1 or t2
c) (t1) and (not t2)

siendo t1 y t2 los términos que se buscan.

3.2. Postings posicionales y consultas de Sintagmas

Supongamos que queremos acceder a un sintagma compuesto por varias palabras

- Sintagma Preposicional: "*En un lugar de la Mancha*"
- Sintagma Nominal: "*Universidad de Barcelona"* o "*Leo Messi*".

La aproximación de indexar cada palabra por separado y hacer operaciones AND entre las postings lists correspondientes puede devolver resultados no deseados:

- "Estas jornadas, celebradas en Barcelona recientemente, contaron con la presencia de investigadores de la universidad."

Hay conceptos complejos o técnicos, nombres de organizaciones, y nombres de productos que son multipalabra (secuencias de más de una palabra, sintagmas).

Los motores de búsqueda actualmente admiten una sintaxis de comillas dobles ("universidad de Stanford") para consultas de secuencias de palabras, que ha demostrado ser muy útil.

Para poder dar soporte a tales consultas, no es suficiente que las postings lists sean simplemente listas de documentos que contienen términos individuales.

Primera solución simple: Indexar pares de palabras consecutivas (biword = término)

"Universitat Politècnica València"

generaría las siguientes entradas en el diccionario:

Universidad Politècnica
Politècnica València

→ La consulta de Sintagmas de dos palabras se realiza de forma trivial.
→ La consulta de Sintagmas de más de dos palabras se realiza obteniendo secuencias de longitud dos.

Un enfoque para manejar sintagmas es considerar cada par de términos consecutivos en un documento como una unidad.

En este modelo, se trata cada uno de estos pares de palabras consecutivas como un término de vocabulario.

Ser capaz de procesar consultas de sintagmas de longitud dos es inmediato.

Los sintagmas más largos se pueden procesar segmentándolos en los biword que contiene y realizando una consulta booleana sobre los mismos.

Problemas con Biwords:

- La talla del diccionario se hace muy grande.
- La búsqueda de palabras sueltas requiere otro diccionario o realizar la unión de los resultados del diccionario de biwords.

Buscar sintagmas de más de dos palabras no es trivial. Calcular la intersección del resultado de la búsqueda de

"Universitat Politècnica" y "Politècnica València"

no garantiza que los documentos encontrados tengan

"Universitat Politècnica València"

Las búsquedas de un solo término no se manejan naturalmente en un índice de biwords (necesitaría escanear el diccionario en busca de todas las biword que contienen el término), por lo que también se necesita tener un índice de términos de una sola palabra.

Si bien siempre existe la posibilidad de coincidencias de falsos positivos, la posibilidad de una coincidencia de falso positivo en frases indexadas de longitud 3 o más se vuelve muy pequeña.

El concepto de biword index se puede extender a secuencias más largas de palabras.

Pero, almacenar sintagmas más largos implica una expansión enormemente el tamaño del vocabulario.

Indices posicionales (o full inverted index)

Para cada término se almacenan postings de la forma: identificador de documento y posiciones en que aparece el término:

<***term***, number of docs containing ***term***;
doc1: position1, position2 … ;
doc2: position1, position2 … ;
etc.>

Otra solución más usual es el uso de índices posicionales (full inverted index), introducidos en el tema 1.

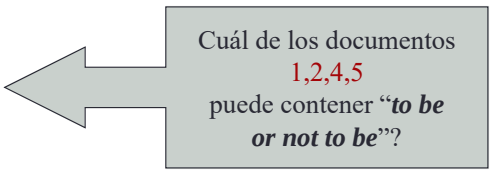
Para cada término del vocabulario, se almacenan postings de la forma docID: position-1, position-2, . . . position-i

Donde cada posición es la posición del token en el document.

Se suele registrar también la frecuencia del término en el documento.

Ejercicio#3:

<***be***: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>



Solucion: 4 y 5

Búsqueda de Sintagmas en índices posicionales

- Estraer las posting lists de cada término: ***to, be, or, not.***
- Hacer "Intersección" de las listas [*docID,tf:positions*] *para encontrar todas las posiciones en que aparece "**to be or not to be**".*

  ***to**,* 993427: [2,5:[1,17,74,222,551]; 4,5:[8,16,190,**429,433]**; 7,3:[13,23,191]; ...]

  be, 178239: [1,2:[17,19]; 4,5:[17,191,291,**430,434]**; 5,3:[14,19,101]; ...

  Requiere trabajar con los offsets entre las palabras y comprobar la compatibilidad con la consulta

Para procesar una consulta de sintagma, se necesita acceder a las entradas del índice invertido para cada término distinto.

Como antes, comenzaría con el período menos frecuente y luego trabajaría para restringir aún más la lista de posibles candidatos.

En la operación de merge, se usa la misma técnica general que antes, pero en lugar de simplemente verificar que ambos términos están en un documento, también debe verificar que sus posiciones de aparición en el documento sean compatibles con la consulta de sintagma que se evalúa. Esto requiere trabajar con los offsets entre las palabras.

Examinaremos la intersección de las postings lists para "to" y "be".

Primero buscamos documentos que contengan ambos términos. Luego, buscamos lugares en las listas donde hay una ocurrencia de "be" con un índice de token más alto que una posición de "to", y luego buscamos otra ocurrencia de cada palabra con un índice de token 4 más alto que la primera ocurrencia.

La talla del índice posicional

- Incluir índices posicionales aumenta la talla de las *postings lists* considerablemente
- Sin embargo lo normal es utilizarlos porque son muy útiles para las búsquedas de sintagmas.
- Un posting tendrá una entrada por cada ocurrencia de un término.
- La talla del índice depende de la talla promedio de los documentos.
  - Por ejemplo, en promedio, una página web tiene una longitud <1000 mientras que un libro puede tener una longitud de 100.000.

Las estrategias de los índices de biwords y los índices posicionales se pueden combinar de manera fructífera.

Si los usuarios suelen realizar consultas sobre sintagmas concretos, como 'Michael Jackson', es bastante ineficaz seguir fusionando postings lists posicionales.

Una estrategia de combinación utiliza un índice de biwords para determinadas consultas, y utiliza un índice posicional para otras consultas de sintagmas.

Si consideramos un término con frecuencia del 1 por 1000 (0.1%) de promedio.

Talla de los documentos	Postings	Positional postings
1000	1	1
100,000	1	100

En general, un índice posicional es de 2 a 4 veces la talla de uno simple y corresponde aproximadamente de 35–50% del volumen del texto original.

Examinemos las implicaciones espaciales de tener un índice posicional. Una publicación ahora necesita una entrada para cada aparición de un término.

Por tanto, el tamaño del índice depende del tamaño medio del documento.

La página web promedio tiene menos de 1000 términos, pero documentos como libros e incluso algunos poemas épicos alcanzan fácilmente los 100,000 términos.

Considere un término con una frecuencia de 1 en 1000 términos en promedio.

El resultado es que los documentos grandes provocan un aumento de dos órdenes de magnitud en el espacio requerido para almacenar las postings lists.

Ejercicio#4

Considerar los siguientes fragmentos de un índice invertido en el cual se ha almacenado la frecuencia de cada término en cada documento indicando la posición de cada ocurrencia. El primer número de cada línea antes de ':' identifica el documento.

a) ¿ Qué documentos satisfacen la consulta: la AND perla AND negra?
b) ¿ Cuántas veces podemos encontrar el sintagma "la perla negra" en cada documento?

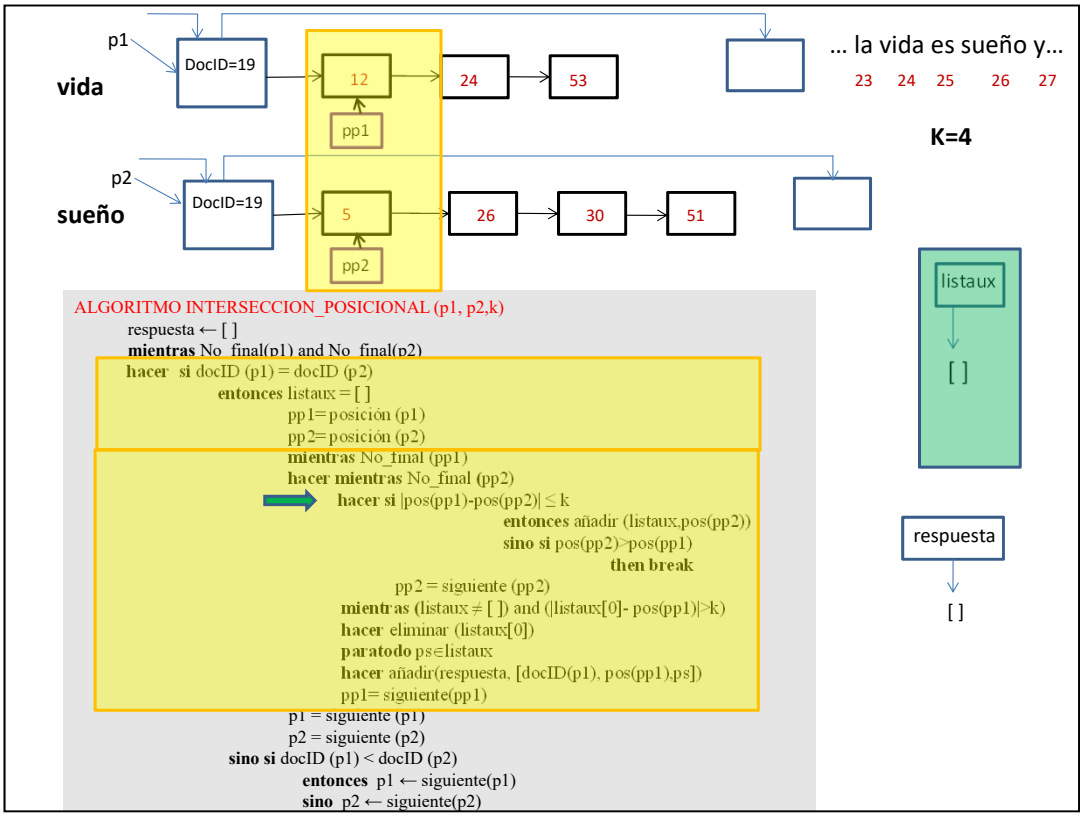
la	perla	negra
3:34,38,55; 5:12,16,25,44; 7:67,87,90,101; 10:33,39,45,62;	3:12,15,19; 5:3,5,17,41,45,96; 6:21,25,55,62; 7:4,68,70,85,110; 10:15,34,40,65,81;	3:22,26; 5:18,46,52,65; 7:5,69,91,105; 8:32,42,65,93; 10:32,44,75,83;

```
ALGORITMO INTERSECCION_POSICIONAL (p1, p2,k)
    respuesta ← [ ]
    mientras No_final(p1) and No_final(p2)
    hacer  si docID (p1) = docID (p2)
                entonces listaux = [ ]
                        pp1= posición (p1)
                        pp2= posición (p2)
                        mientras No_final (pp1)
                        hacer mientras No_final (pp2)
                                hacer si |pos(pp1)-pos(pp2)| ≤ k
                                                entonces añadir (listaux,pos(pp2))
                                                sino si pos(pp2)>pos(pp1)
                                                                then break
                                        pp2 = siguiente (pp2)
                                mientras (listaux ≠ [ ]) and (|listaux[0]- pos(pp1)|>k)
                                hacer eliminar (listaux[0])
                                paratodo ps∈listaux
                                hacer añadir(respuesta, [docID(p1), pos(pp1),ps])
                                pp1= siguiente(pp1)
                        p1 = siguiente (p1)
                        p2 = siguiente (p2)
                 sino si docID (p1) < docID (p2)
                          entonces  p1 ← siguiente(p1)
                          sino  p2 ← siguiente(p2)
    return respuesta
```

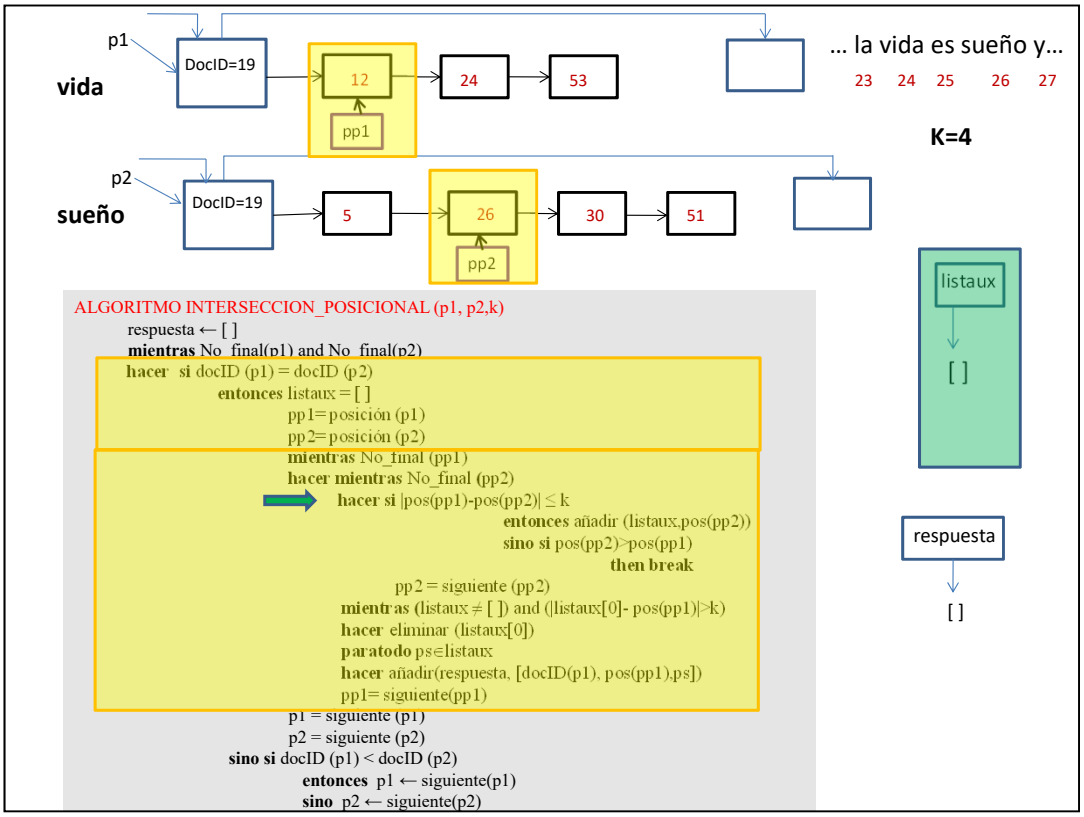
Se muestra un algoritmo para la intersección de las postings lists posicionales apuntadas por p1 y p2.

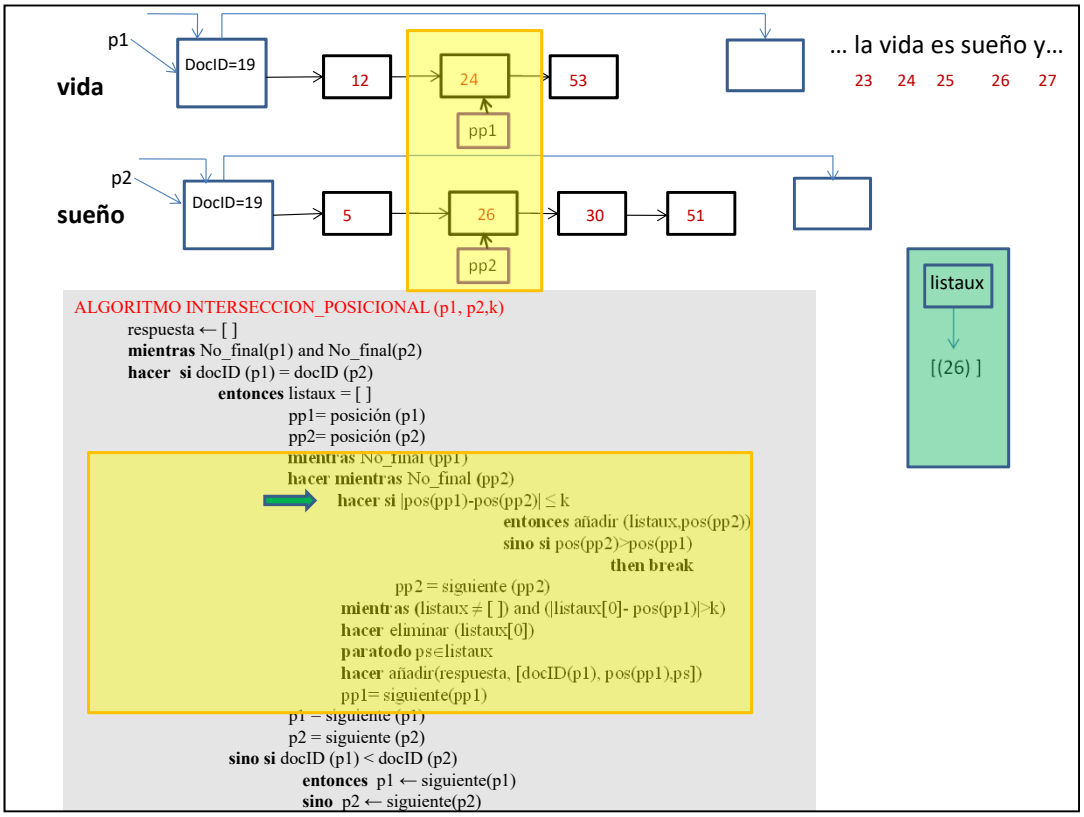
El algoritmo encuentra ocurrencias donde los dos términos aparecen con una distancia de como mucho k palabras entre sí.

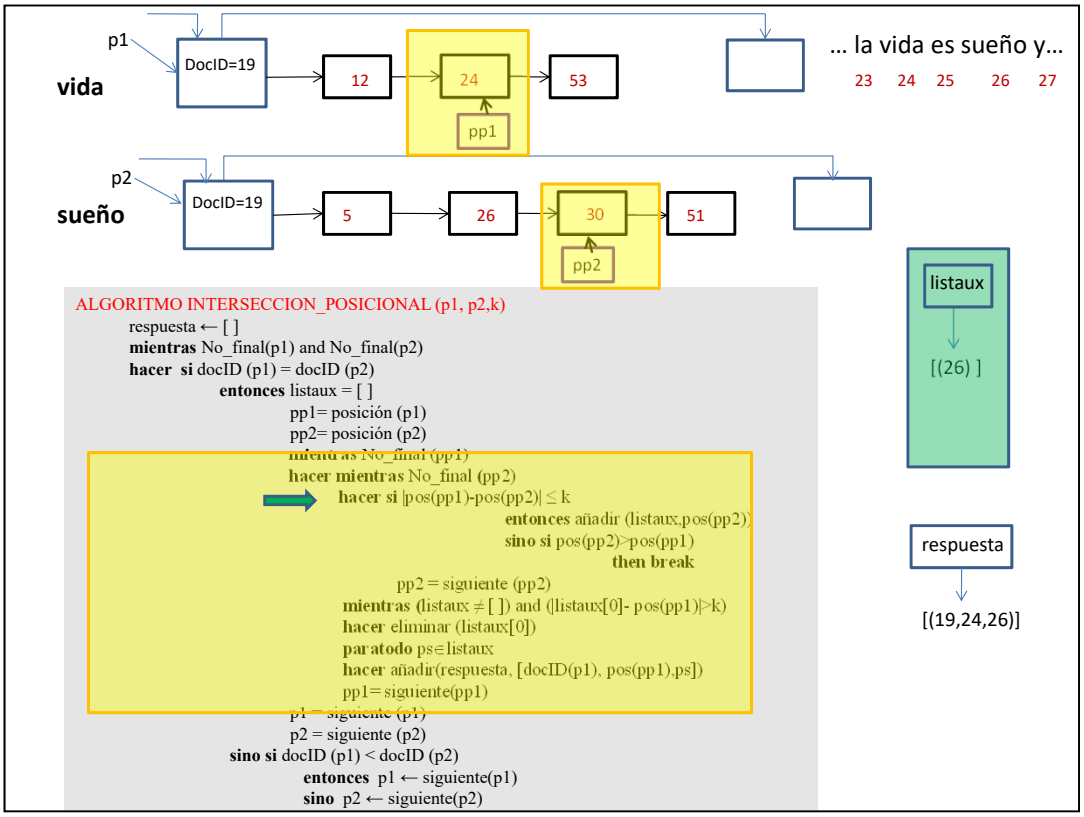
Devuelve una lista de tripletes que muestran el docID y la posición del término en p1 y p2.

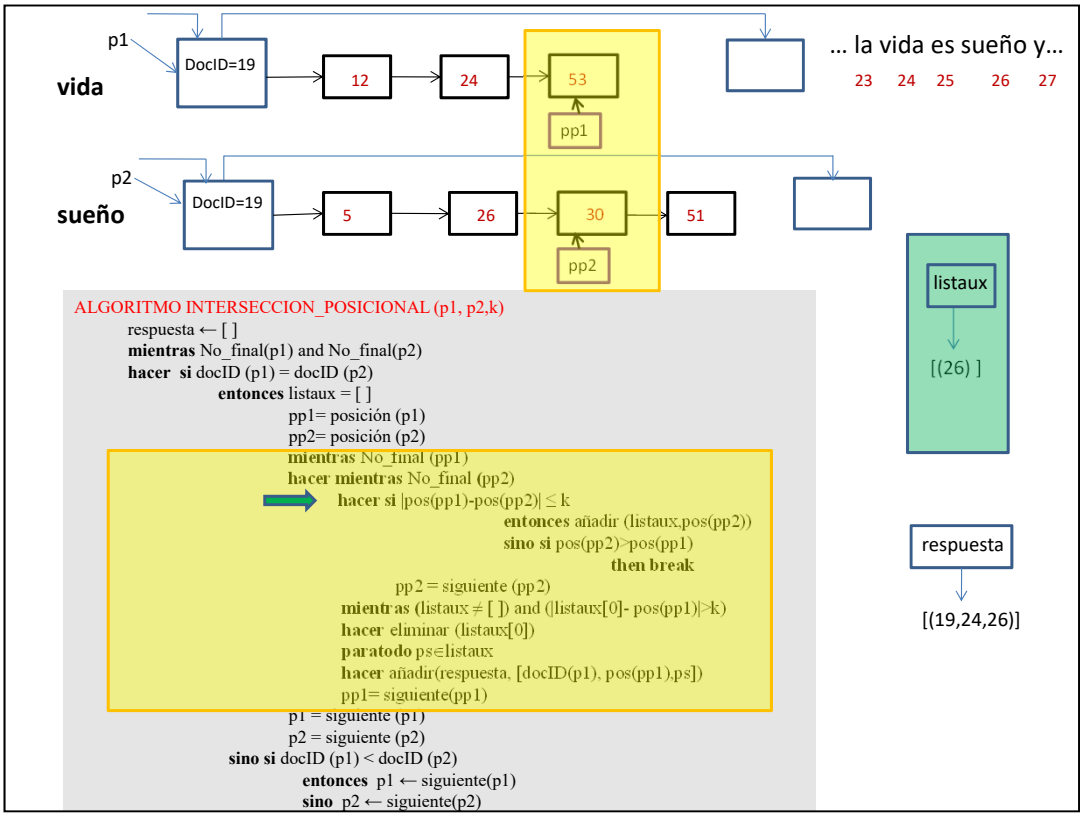


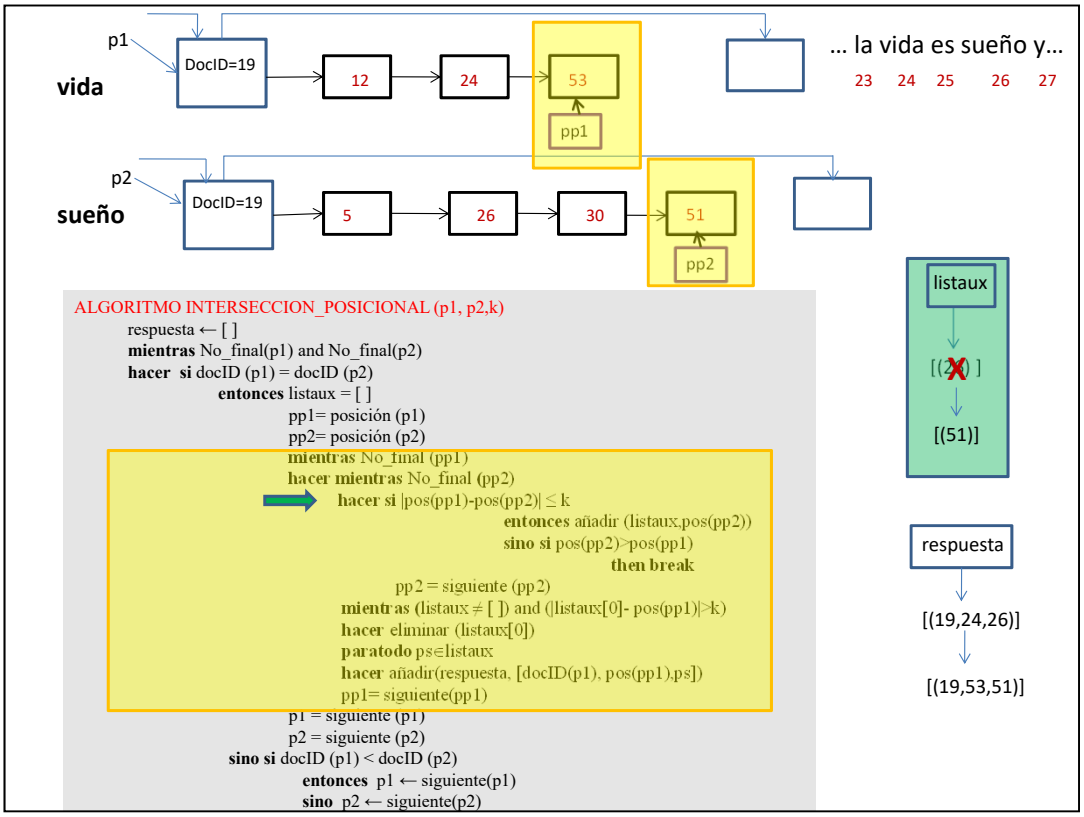
Veamos una traza del algoritmo para el ejemplo de las listas apuntadas por p1 y p2 que se muestran en la diapositiva y con k=4.

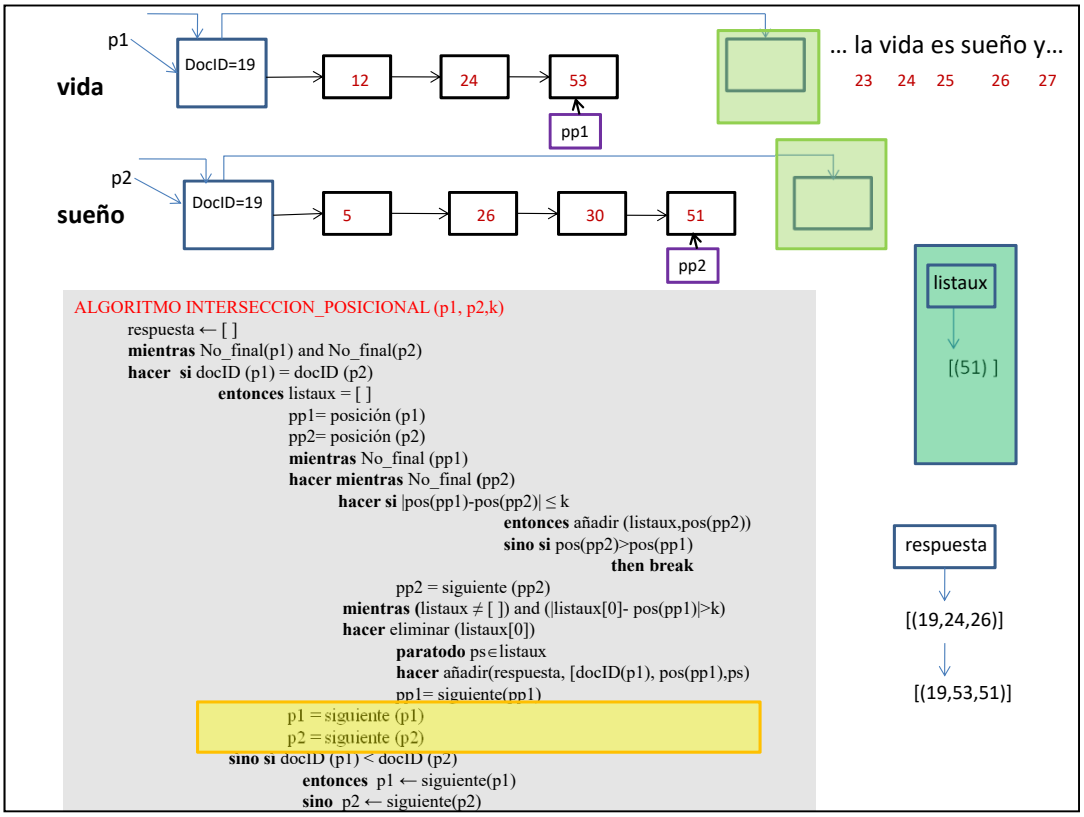

p1
vida
DocID=19
12
24
53
pp1
... la vida es sueño y...
23 24 25 26 27
K=4
p2
sueño
DocID=19
5
26
30
51
pp2
listaux
[ ]
respuesta
[ ]
ALGORITMO INTERSECCION_POSICIONAL (p1, p2,k)
respuesta ← [ ]
mientras No_final(p1) and No_final(p2)
hacer si docID (p1) = docID (p2)
entonces listaux = [ ]
pp1= posición (p1)
pp2= posición (p2)
mientras No_final (pp1)
hacer mientras No_final (pp2)
hacer si |pos(pp1)-pos(pp2)| ≤ k
entonces añadir (listaux,pos(pp2))
sino si pos(pp2)>pos(pp1)
then break
pp2 = siguiente (pp2)
mientras (listaux ≠ [ ]) and (|listaux[0]- pos(pp1)|>k)
hacer eliminar (listaux[0])
paratodo ps∈listaux
hacer añadir(respuesta, [docID(p1), pos(pp1),ps])
pp1= siguiente(pp1)
p1 = siguiente (p1)
p2 = siguiente (p2)
sino si docID (p1) < docID (p2)
entonces p1 ← siguiente(p1)
sino p2 ← siguiente(p2)


p1
vida
DocID=19
12
24
53
pp1
... la vida es sueño y...
23 24 25 26 27
p2
sueño
DocID=19
5
26
30
51
pp2
listaux
[(26)]
ALGORITMO INTERSECCION_POSICIONAL (p1, p2,k)
respuesta ← [ ]
mientras No_final(p1) and No_final(p2)
hacer si docID (p1) = docID (p2)
entonces listaux = [ ]
pp1= posición (p1)
pp2= posición (p2)
mientras No_final (pp1)
hacer mientras No_final (pp2)
hacer si |pos(pp1)-pos(pp2)| ≤ k
entonces añadir (listaux,pos(pp2))
sino si pos(pp2)>pos(pp1)
then break
pp2 = siguiente (pp2)
mientras (listaux ≠ [ ]) and (|listaux[0]- pos(pp1)|>k)
hacer eliminar (listaux[0])
paratodo ps∈listaux
hacer añadir(respuesta, [docID(p1), pos(pp1),ps])
pp1= siguiente(pp1)
p1 = siguiente (p1)
p2 = siguiente (p2)
sino si docID (p1) < docID (p2)
entonces p1 ← siguiente(p1)
sino p2 ← siguiente(p2)

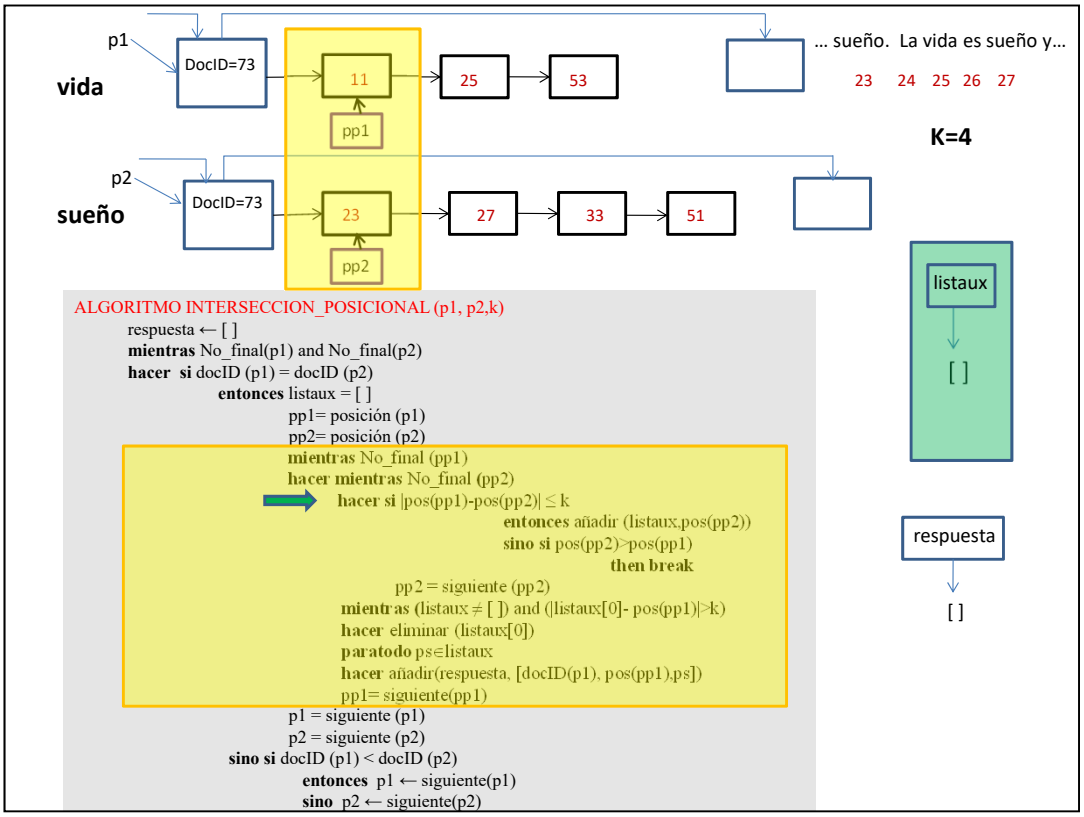

p1
vida
DocID=19
12
24
53
pp1
... la vida es sueño y...
23 24 25 26 27
p2
sueño
DocID=19
5
26
30
51
pp2
listaux
[(26)]
respuesta
[(19,24,26)]
ALGORITMO INTERSECCION_POSICIONAL (p1, p2,k)
respuesta ← [ ]
mientras No_final(p1) and No_final(p2)
hacer si docID (p1) = docID (p2)
entonces listaux = [ ]
pp1= posición (p1)
pp2= posición (p2)
hacer mientras No_final (pp2)
hacer si |pos(pp1)-pos(pp2)| ≤ k
entonces añadir (listaux,pos(pp2))
sino si pos(pp2)>pos(pp1)
then break
pp2 = siguiente (pp2)
mientras (listaux ≠ [ ]) and (|listaux[0]- pos(pp1)|>k)
hacer eliminar (listaux[0])
paratodo ps∈listaux
hacer añadir(respuesta, [docID(p1), pos(pp1),ps])
pp1= siguiente(pp1)
p2 = siguiente (p2)
sino si docID (p1) < docID (p2)
entonces p1 ← siguiente(p1)
sino p2 ← siguiente(p2)

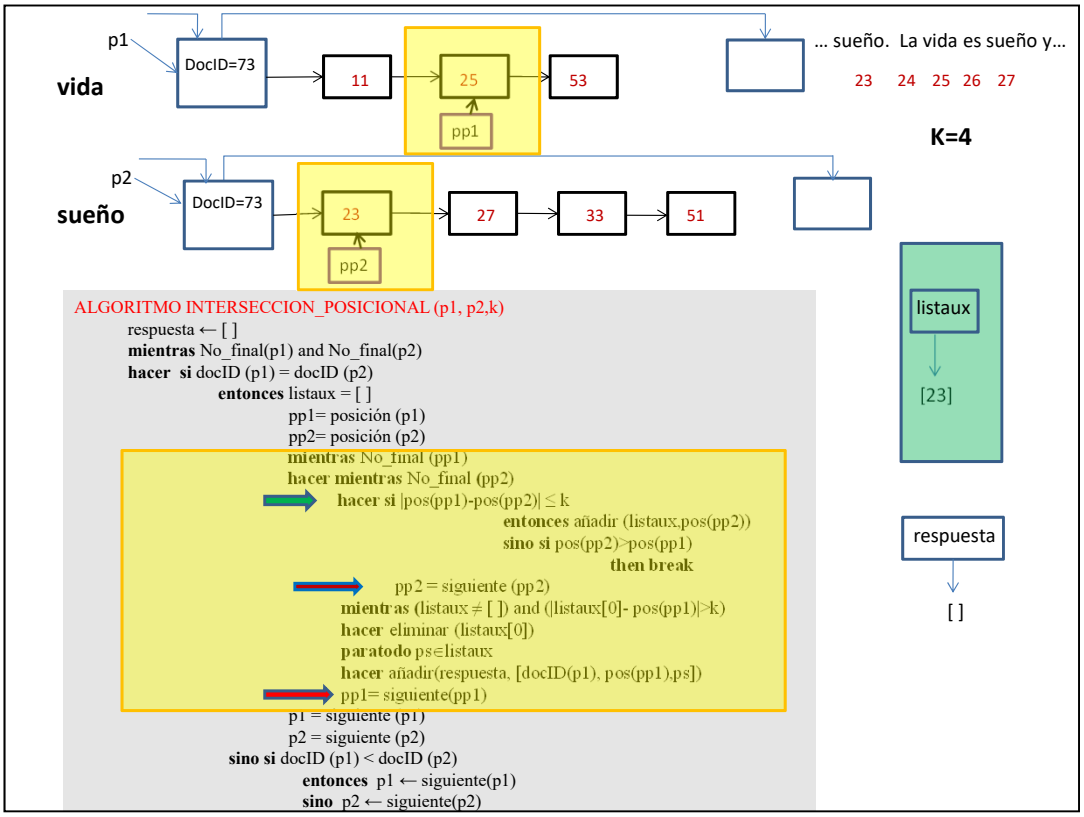

p1
vida
DocID=19
12
24
53
pp1
p2
sueño
DocID=19
5
26
30
51
pp2
… la vida es sueño y…
23 24 25 26 27
listaux
[(26)]
respuesta
[(19,24,26)]
ALGORITMO INTERSECCION_POSICIONAL (p1, p2,k)
respuesta ← [ ]
mientras No_final(p1) and No_final(p2)
hacer si docID (p1) = docID (p2)
entonces listaux = [ ]
pp1= posición (p1)
pp2= posición (p2)
mientras No_final (pp1)
hacer mientras No_final (pp2)
hacer si |pos(pp1)-pos(pp2)| ≤ k
entonces añadir (listaux,pos(pp2))
sino si pos(pp2)>pos(pp1)
then break
pp2 = siguiente (pp2)
mientras (listaux ≠ [ ]) and (|listaux[0]- pos(pp1)|>k)
hacer eliminar (listaux[0])
paratodo ps∈listaux
hacer añadir(respuesta, [docID(p1), pos(pp1),ps])
pp1= siguiente(pp1)
p1 = siguiente (p1)
p2 = siguiente (p2)
sino si docID (p1) < docID (p2)
entonces p1 ← siguiente(p1)
sino p2 ← siguiente(p2)

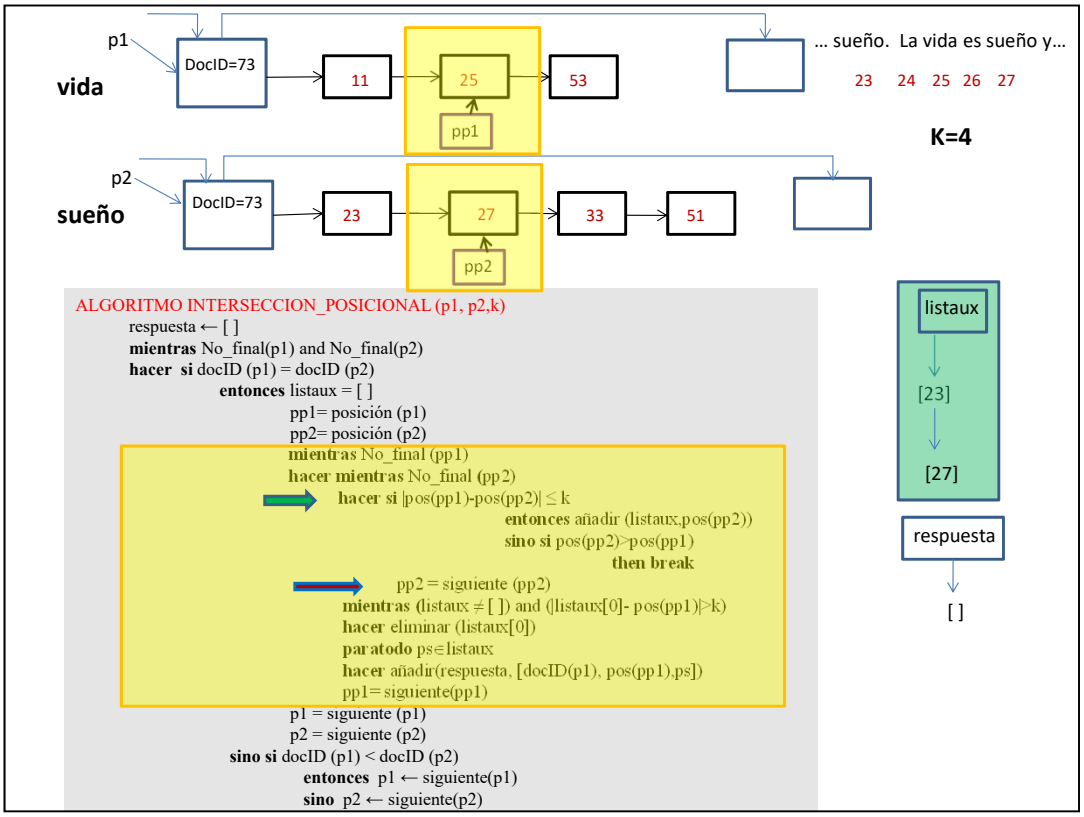

p1
vida
DocID=19
12
24
53
pp1
... la vida es sueño y...
23 24 25 26 27
p2
sueño
DocID=19
5
26
30
51
pp2
ALGORITMO INTERSECCION_POSICIONAL (p1, p2,k)
respuesta ← [ ]
mientras No_final(p1) and No_final(p2)
hacer si docID (p1) = docID (p2)
entonces listaux = [ ]
pp1= posición (p1)
pp2= posición (p2)
mientras No_final (pp1)
hacer mientras No_final (pp2)
hacer si |pos(pp1)-pos(pp2)| ≤ k
entonces añadir (listaux,pos(pp2))
sino si pos(pp2)>pos(pp1)
then break
pp2 = siguiente (pp2)
mientras (listaux ≠ [ ]) and (|listaux[0]- pos(pp1)|>k)
hacer eliminar (listaux[0])
paratodo ps∈listaux
hacer añadir(respuesta, [docID(p1), pos(pp1),ps])
pp1= siguiente(pp1)
p1 = siguiente (p1)
p2 = siguiente (p2)
sino si docID (p1) < docID (p2)
entonces p1 ← siguiente(p1)
sino p2 ← siguiente(p2)
listaux
[(51)]
respuesta
[(19,24,26)]
[(19,53,51)]

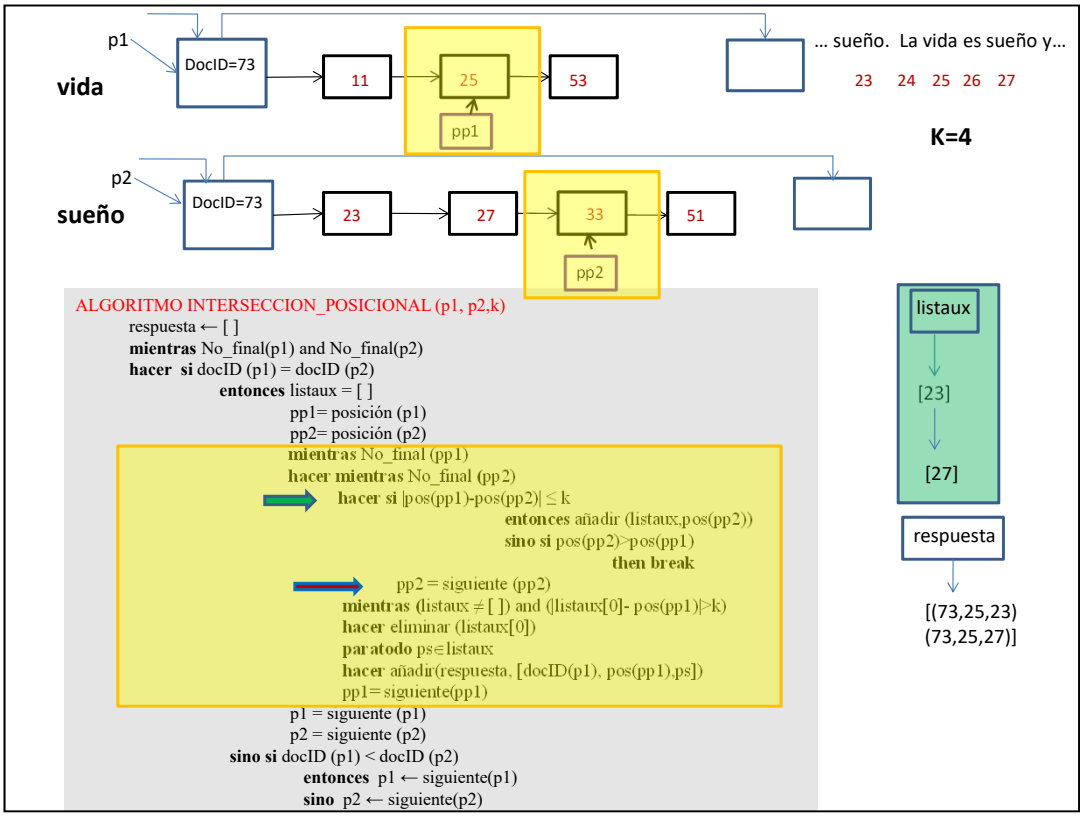

p1
vida
DocID=19
12
24
53
pp1
... la vida es sueño y...
23 24 25 26 27
p2
sueño
DocID=19
5
26
30
51
pp2
listaux
[(51)]
respuesta
[(19,24,26)]
[(19,53,51)]
ALGORITMO INTERSECCION_POSICIONAL (p1, p2,k)
respuesta ← [ ]
mientras No_final(p1) and No_final(p2)
hacer si docID (p1) = docID (p2)
entonces listaux = [ ]
pp1= posición (p1)
pp2= posición (p2)
mientras No_final (pp1)
hacer mientras No_final (pp2)
hacer si |pos(pp1)-pos(pp2)| ≤ k
entonces añadir (listaux,pos(pp2))
sino si pos(pp2)>pos(pp1)
then break
pp2 = siguiente (pp2)
mientras (listaux ≠ [ ]) and (|listaux[0]- pos(pp1)|>k)
hacer eliminar (listaux[0])
paratodo ps∈listaux
hacer añadir(respuesta, [docID(p1), pos(pp1),ps)
pp1= siguiente(pp1)
p1 = siguiente (p1)
p2 = siguiente (p2)
sino si docID (p1) < docID (p2)
entonces p1 ← siguiente(p1)
sino p2 ← siguiente(p2)

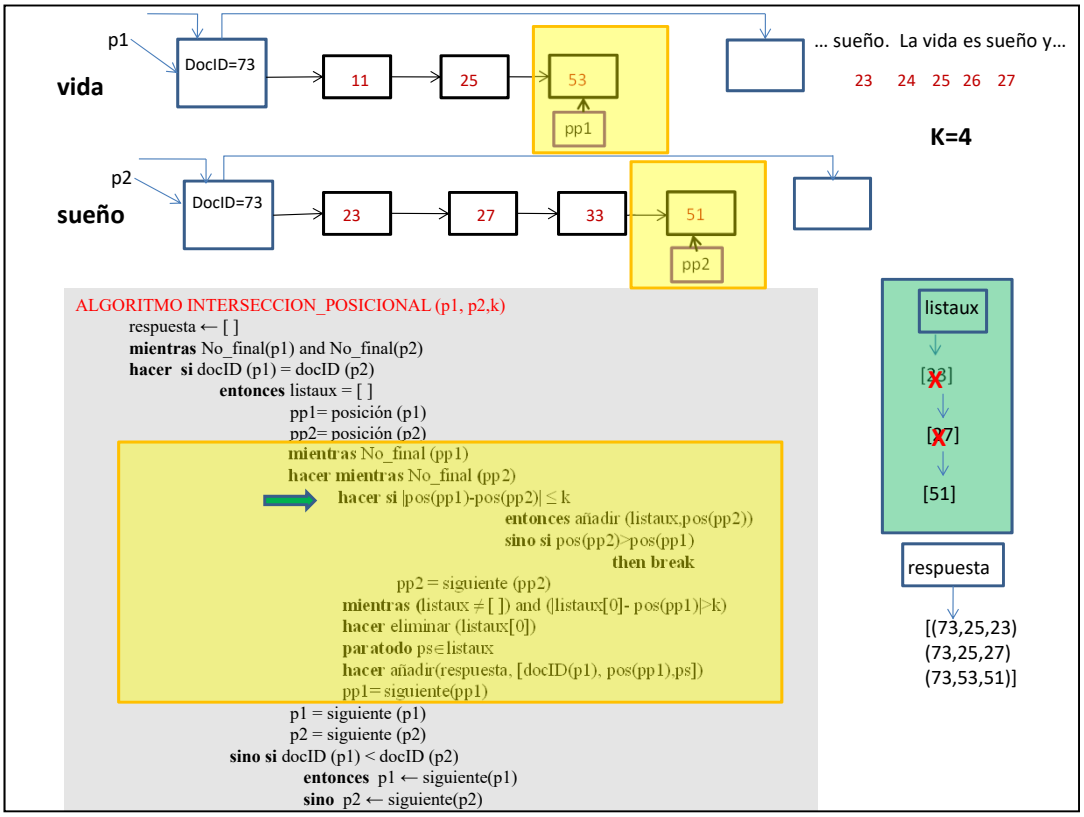
una situación más compleja…


p1
vida
DocID=73
11
25
53
pp1
p2
sueño
DocID=73
23
27
33
51
pp2
... sueño. La vida es sueño y...
23 24 25 26 27
K=4
listaux
[ ]
respuesta
[ ]
ALGORITMO INTERSECCION_POSICIONAL (p1, p2,k)
respuesta ← [ ]
mientras No_final(p1) and No_final(p2)
hacer si docID (p1) = docID (p2)
entonces listaux = [ ]
pp1= posición (p1)
pp2= posición (p2)
mientras No_final (pp1)
hacer mientras No_final (pp2)
hacer si |pos(pp1)-pos(pp2)| ≤ k
entonces añadir (listaux,pos(pp2))
sino si pos(pp2)>pos(pp1)
then break
pp2 = siguiente (pp2)
mientras (listaux ≠ [ ]) and (|listaux[0]- pos(pp1)|>k)
hacer eliminar (listaux[0])
paratodo ps∈listaux
hacer añadir(respuesta, [docID(p1), pos(pp1),ps])
pp1= siguiente(pp1)
p1 = siguiente (p1)
p2 = siguiente (p2)
sino si docID (p1) < docID (p2)
entonces p1 ← siguiente(p1)
sino p2 ← siguiente(p2)


p1
vida
DocID=73
11
25
53
pp1
... sueño. La vida es sueño y...
23 24 25 26 27
K=4
p2
sueño
DocID=73
23
27
33
51
pp2
listaux
[23]
respuesta
[ ]
ALGORITMO INTERSECCION_POSICIONAL (p1, p2,k)
respuesta ← [ ]
mientras No_final(p1) and No_final(p2)
hacer si docID (p1) = docID (p2)
entonces listaux = [ ]
pp1= posición (p1)
pp2= posición (p2)
mientras No_final (pp1)
hacer mientras No_final (pp2)
hacer si |pos(pp1)-pos(pp2)| ≤ k
entonces añadir (listaux,pos(pp2))
sino si pos(pp2)>pos(pp1)
then break
pp2 = siguiente (pp2)
mientras (listaux ≠ [ ]) and (|listaux[0]- pos(pp1)|>k)
hacer eliminar (listaux[0])
paratodo ps∈listaux
hacer añadir(respuesta, [docID(p1), pos(pp1),ps])
pp1= siguiente(pp1)
p1 = siguiente (p1)
p2 = siguiente (p2)
sino si docID (p1) < docID (p2)
entonces p1 ← siguiente(p1)
sino p2 ← siguiente(p2)


p1
vida
DocID=73
11
25
53
pp1
... sueño. La vida es sueño y...
23 24 25 26 27
K=4
p2
sueño
DocID=73
23
27
33
51
pp2
ALGORITMO INTERSECCION_POSICIONAL (p1, p2,k)
respuesta ← [ ]
mientras No_final(p1) and No_final(p2)
hacer si docID (p1) = docID (p2)
entonces listaux = [ ]
pp1= posición (p1)
pp2= posición (p2)
mientras No_final (pp1)
hacer mientras No_final (pp2)
hacer si |pos(pp1)-pos(pp2)| ≤ k
entonces añadir (listaux,pos(pp2))
sino si pos(pp2)>pos(pp1)
then break
pp2 = siguiente (pp2)
mientras (listaux ≠ [ ]) and (|listaux[0]- pos(pp1)|>k)
hacer eliminar (listaux[0])
paratodo ps∈listaux
hacer añadir(respuesta, [docID(p1), pos(pp1),ps])
pp1= siguiente(pp1)
p1 = siguiente (p1)
p2 = siguiente (p2)
sino si docID (p1) < docID (p2)
entonces p1 ← siguiente(p1)
sino p2 ← siguiente(p2)
listaux
[23]
[27]
respuesta
[ ]


p1
vida
DocID=73
11
25
53
pp1
... sueño. La vida es sueño y...
23 24 25 26 27
K=4
p2
sueño
DocID=73
23
27
33
51
pp2
ALGORITMO INTERSECCION_POSICIONAL (p1, p2,k)
respuesta ← [ ]
mientras No_final(p1) and No_final(p2)
hacer si docID (p1) = docID (p2)
entonces listaux = [ ]
pp1= posición (p1)
pp2= posición (p2)
mientras No_final (pp1)
hacer mientras No_final (pp2)
hacer si |pos(pp1)-pos(pp2)| ≤ k
entonces añadir (listaux,pos(pp2))
sino si pos(pp2)>pos(pp1)
then break
pp2 = siguiente (pp2)
mientras (listaux ≠ [ ]) and (|listaux[0]- pos(pp1)|>k)
hacer eliminar (listaux[0])
paratodo ps∈listaux
hacer añadir(respuesta, [docID(p1), pos(pp1),ps])
pp1= siguiente(pp1)
p1 = siguiente (p1)
p2 = siguiente (p2)
sino si docID (p1) < docID (p2)
entonces p1 ← siguiente(p1)
sino p2 ← siguiente(p2)
listaux
[23]
[27]
respuesta
[(73,25,23)
(73,25,27)]


p1
vida
DocID=73
11
25
53
pp1
... sueño. La vida es sueño y...
23 24 25 26 27
K=4
p2
sueño
DocID=73
23
27
33
51
pp2
ALGORITMO INTERSECCION_POSICIONAL (p1, p2,k)
respuesta ← [ ]
mientras No_final(p1) and No_final(p2)
hacer si docID (p1) = docID (p2)
entonces listaux = [ ]
pp1= posición (p1)
pp2= posición (p2)
mientras No_final (pp1)
hacer mientras No_final (pp2)
hacer si |pos(pp1)-pos(pp2)| ≤ k
entonces añadir (listaux,pos(pp2))
sino si pos(pp2)>pos(pp1)
then break
pp2 = siguiente (pp2)
mientras (listaux ≠ [ ]) and (|listaux[0]- pos(pp1)|>k)
hacer eliminar (listaux[0])
paratodo ps∈listaux
hacer añadir(respuesta, [docID(p1), pos(pp1),ps])
pp1= siguiente(pp1)
p1 = siguiente (p1)
p2 = siguiente (p2)
sino si docID (p1) < docID (p2)
entonces p1 ← siguiente(p1)
sino p2 ← siguiente(p2)
listaux
[33]
[27]
[51]
respuesta
[(73,25,23)
(73,25,27)
(73,53,51)]