# TEMA 1. INTRODUCCIÓN A LA RECUPERACIÓN DE INFORMACIÓN

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

# Contenidos

# Bibliografía

***A Introduction to Information Retrieval***:
*Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze.*
*Cambridge University Press,* ***2009****.*
***Capítulo 1***

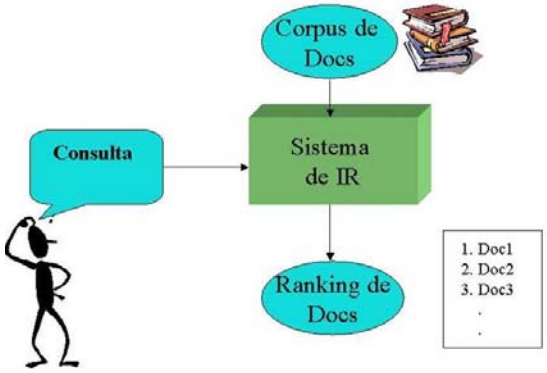

# 1. INTRODUCCIÓN

1.1. Recuperación de Información
1.2. Objetivos de la RI
1.3. Algo de historia
1.4. Relación con otras disciplinas
1.5. Recuperación de Datos vs RI

# 1.1. Recuperación de Información (RI)

Bajo este nombre se engloban las diversas arquitecturas, algoritmos, o sistemas cuyo objetivo es encontrar ciertos elementos entre una gran colección no estructurada de ellos, que satisfacen una información requerida.

Trata con la representación, almacenamiento, organización y acceso a la información.



# 1.2. Objetivos de la RI

- Objetivos originales de la Recuperación de Información: búsqueda e indexación de documentos.

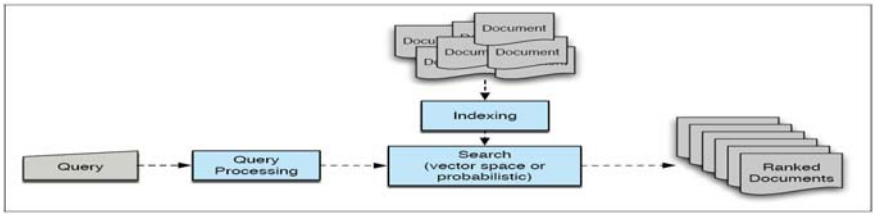
- Objetivos actuales de la Recuperación de Información:
  - búsqueda e indexación de documentos,
  - búsqueda en la web,
  - clasificación de textos,
  - arquitecturas de los sistemas,
  - interfaces de usuario,
  - visualización de los datos,
  - filtrado, multilingüismo …

# 1.3. Algo de historia

- Desde los orígenes de la escritura hace más de 5000 años, los hombres han organizado la información para posteriormente poder acceder fácilmente a sus distintos elementos.
- Para almacenar los libros, papiros o documentos se construyeron edificios específicos: bibliotecas.
- La biblioteca más antigua conocida existió en Elba, en la civilización de los sumerios, entre los años 3000 a 2500 aC. En el 300 aC Ptolomeo I creó la biblioteca de Alejandría.

# Algo de historia cont.

- El volumen de información propicia el desarrollo de estructuras para realizar búsquedas rápidas: los *índices*.
- Los índices fueron creados manualmente representando conjuntos de *categorías*, con etiquetas asociadas a éstas.
- Con la aparición de los computadores se pudieron empezar a generar índices de grandes volúmenes de datos de forma automática.




# 1.4. Relación con otras disciplinas

En Recuperación de Información intervienen tópicos y técnicas procedentes de otras disciplinas:

- Estructuras de Datos y Algoritmos
- Bases de datos
- Procesamiento de lenguaje natural
- Inteligencia Artificial
- Interfaces y visualización
- Minería de datos
- Machine learning

# 1.5. Recuperación de datos vs RI

| | Recuperación de datos | Recuperación de información |
|---|---|---|
| Según la forma de responder a la pregunta | se utilizan preguntas altamente formalizadas, cuya respuesta es directamente la información deseada | las preguntas resultan difíciles de trasladar a un lenguaje normalizado, y la respuesta es un conjunto de documentos que pueden contener, sólo probablemente, lo deseado, con un evidente factor de indeterminación |
| Según la relación entre el requerimiento al sistema y la satisfacción del usuario | la relación es determinística entre la pregunta y la satisfacción | la relación es probabilística, a causa del nivel de incertidumbre presente en la respuesta |
| Según el criterio de éxito | el criterio a emplear es la corrección y la exactitud | el único criterio de valor es la satisfacción del usuario, basada en un criterio personal de utilidad |
| Según la rapidez de respuesta | depende del soporte físico y de la perfección del algoritmo de búsqueda y de los índices | depende de las decisiones y acciones del usuario durante el proceso. |

## Ejemplo de Recuperación de Información:

Consulta: ¿Qué obras de Shakespeare contienen las palabras ***Brutus*** *y* ***Caesar*** pero no ***Calpurnia***?

Búsqueda lineal ("Mala" solución ):
Rastrear línea a línea todas las obras de Shakespeare para encontrar las que contienen ***Brutus*** y ***Caesar,*** y después eliminar aquéllas que contienen ***Calpurnia***

¿Por qué es una mala solución?

- Lento (para grandes colecciones de documentos)
- No es trivial procesar *NOT* ***Calpurnia***
- No permite ordenación de lo recuperado por relevancia (*ranking*)

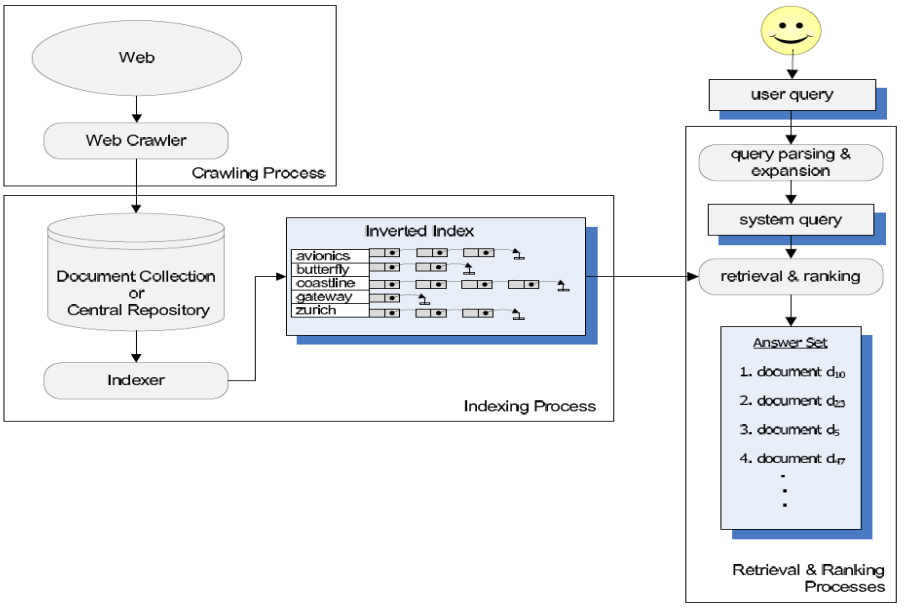
Construcción de un índice

Sea una colección de documentos formada por las Obras completas de Shakespeare.
La forma de evitar escanear linealmente los textos para cada consulta es indexar los documentos por adelantado.

# 2. ARQUITECTURA

2.1. Arquitectura de un sistema de RI
2.2. Proceso de indexación, recuperación y ranking

# 2.1. Arquitectura de un sistema de RI



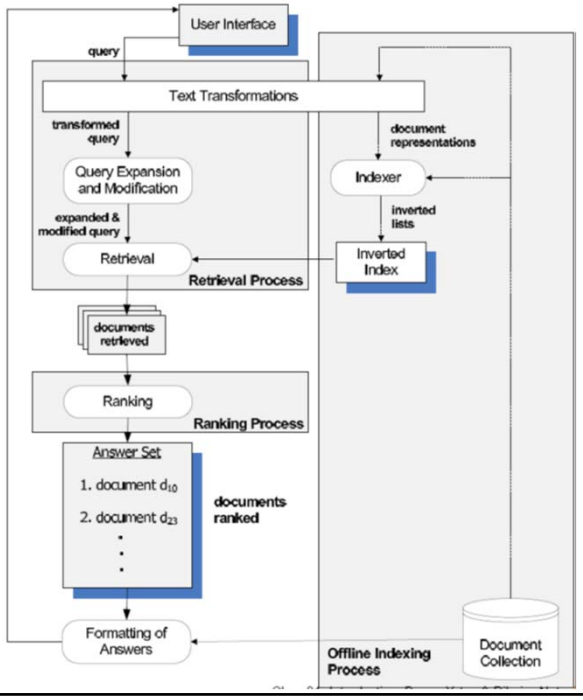

Arquitectura software de alto nivel de un sistema de RI

La consulta de usuario es procesada y expandida, por ejemplo con variantes de las palabras, generando la llamada “consulta de sistema”.

Esta consulta es procesada en el índice y se recupera un subconjunto de la colección de documentos completa.

Este subconjunto es “rankeado” y al usuario se le devuelven los primeros documentos del ranking.

# 2.2. Proceso de indexación, recuperación y ranking




El proceso de indexación, recuperación y ordenación de los documentos.

Indexación:

Dada la colección de documentos, se aplican operaciones sobre texto tales como eliminación de stopwords, stemming, y se selecciona el subconjunto de términos (unidades) que definirán el vocabulario.

Estos términos se usan para componer la representación de los documentos, que puede/suele ser más pequeña que la colección.

Dada la representación de los documentos, hay que construir el índice del texto. La más usual es el índice invertido. Este es el proceso de indexación, que hay que realizar antes.

Una vez indexada la colección, el proceso de recuperación puede empezar.

El usuario especifica una consulta, que sufre un proceso de transformación similar a la de los documentos; corrección ortográfica y eliminación de términos como las stopwords.

La consulta es expandida y modificada, esta modificación es confirmada por el sistema con el usuario.

Recuperación:

Se procesa la consulta con la ayuda del índice y se devuelve el resultado, los documentos recuperados.

Ordenación:

A continuación los documentos son ordenados atendiendo a la relevancia con respecto a la consulta del usuario.

Los documentos más relevantes son formateados para su presentación al usuario; recuperación del título, generación de snippets (segmentos de texto que contienen la consulta).

# 3. MODELO DE RI BOOLEANO

3.1. Consultas y respuestas

3.2. Matriz de incidencia binaria

# 3.1. Consultas y respuestas

Las consultas (query) son expresiones booleanas de términos que se combinan con operadores lógicos AND, OR y NOT.

Ejemplo.

**Query**: ***Brutus*** *AND* ***Caesar*** *AND NOT* ***Calpurnia***

**Respuesta**: ***Documentos relevantes respecto a la query***

Los **términos** son las **unidades de indexación**, generalmente son palabras y, por el momento, se puede pensar en ellas como palabras, pero en recuperación de información se habla de términos porque algunos de ellos, grupos de palabras como 'Hong Kong' o 'L-9', generalmente no se consideran como palabras.

El modelo de recuperación booleano es un modelo de recuperación de información en el que podemos plantear cualquier consulta en forma de expresión booleana de términos, es decir, en la que los términos se combinan con los operadores AND, OR, y NOT.
El modelo ve cada documento como un conjunto de palabras (términos).

## 3.2. Matriz de incidencia binaria

Para cada término de la colección indica si un documento contiene o no el término (palabra) : matriz de incidencia binaria término-documento del corpus.

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| Antony | 1 | 1 | 0 | 0 | 0 | 1 |
| Brutus | 1 | 1 | 0 | 1 | 0 | 0 |
| Caesar | 1 | 1 | 0 | 1 | 1 | 1 |
| Calpurnia | 0 | 1 | 0 | 0 | 0 | 0 |
| Cleopatra | 1 | 0 | 0 | 0 | 0 | 0 |
| mercy | 1 | 0 | 1 | 1 | 1 | 1 |
| worser | 1 | 0 | 1 | 1 | 1 | 0 |

Query: ***Brutus*** *AND* ***Caesar*** *BUT NOT* ***Calpurnia***

**Cálculo y Respuesta:**

110100 *AND* 110111 *AND* 101111 = **100100**

Vector de Brutus **AND** vector de Caesar **AND** vector complementario de Calpurnia

Seguimos con las Obras completas de Shakespeare y las vamos a utilizar para presentar los conceptos básicos del modelo de recuperación booleano.
Supongamos que registramos para cada documento, aquí una obra de Shakespeare, si contiene cada palabra de todas las palabras que Shakespeare usó (Shakespeare usó unas 32.000 palabras diferentes).
El resultado es la matriz de incidencia binaria término-documento de la colección.

Dependiendo de si miramos las filas o columnas de la matriz, podemos tener un vector para cada término, que muestra los documentos en los que aparece, o un vector para cada documento, mostrando los términos que ocurren en él.

Para responder a la consulta 'Brutus AND Caesar AND NOT Calpurnia', tomamos los vectores para 'Brutus', 'Caesar' y 'Calpurnia', complementamos el último y luego hacemos un AND a nivel de bits.

Respuesta al Query: documentos relevantes

Antony and Cleopatra, Acto III, Escena 2

*Agrippa*: Why, Enobarbus,
When Antony found Julius ***Caesar*** dead,
He cried almost to roaring; and he wept
When at Philippi he found ***Brutus*** slain.

Hamlet, Acto III, Escena 2

*Lord Polonius:* I did enact Julius ***Caesar*** I was killed i' the Capitol; ***Brutus*** killed me.

La respuesta para esta consulta son las obras *Antony and Cleopatra* y *Hamlet.*

# 4. ÍNDICE INVERTIDO

4.1. Índice invertido: definición
4.2. Fases de construcción de un Índice invertido
4.3. Pasos en la indexación
4.4. Tipos de índices invertidos

## ¿Qué ocurre con grandes colecciones de documentos?

- Sea un corpus de 1 millón de documentos, cada uno de ellos de 1000 palabras, con un promedio de 6 bytes/palabra: espacio ocupado 6 GB.

- Supongamos que el número de términos diferentes en este corpus es de 500K: matriz de incidencia 500Kx1M de 0's y 1's.

- Normalmente esta matriz es muy *dispersa*, es decir hay muchos 0's: mejor es almacenar solamente los 1's, es decir, cada palabra en qué documento está.

**Indices Invertidos**

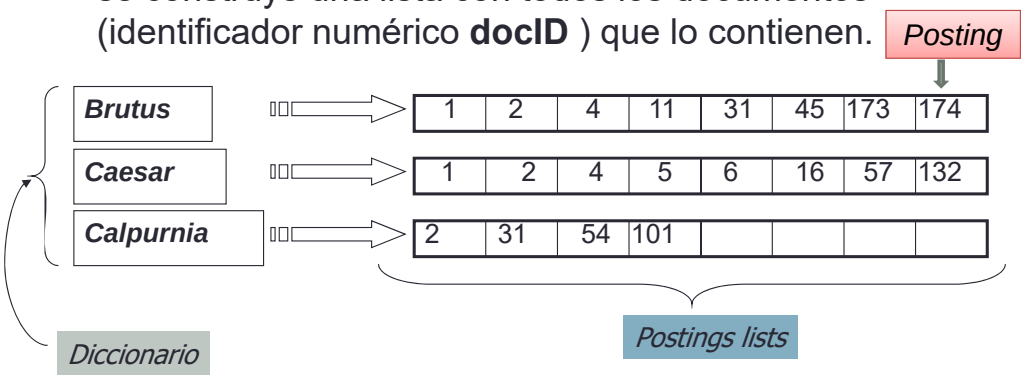
Para almacenar únicamente la información relevante para el proceso, se introduce el concepto de índice invertido.
Esta idea es fundamental en la recuperación de información.

# 4.1. Índice invertido: definición

Esta constituido por dos elementos:

**Diccionario**, lista de términos distintos que contiene el texto, y

**Lista de ocurrencias** (postings list) para cada término se construye una lista con todos los documentos (identificador numérico **docID** ) que lo contienen.




La idea básica de un índice invertido se muestra en la figura.

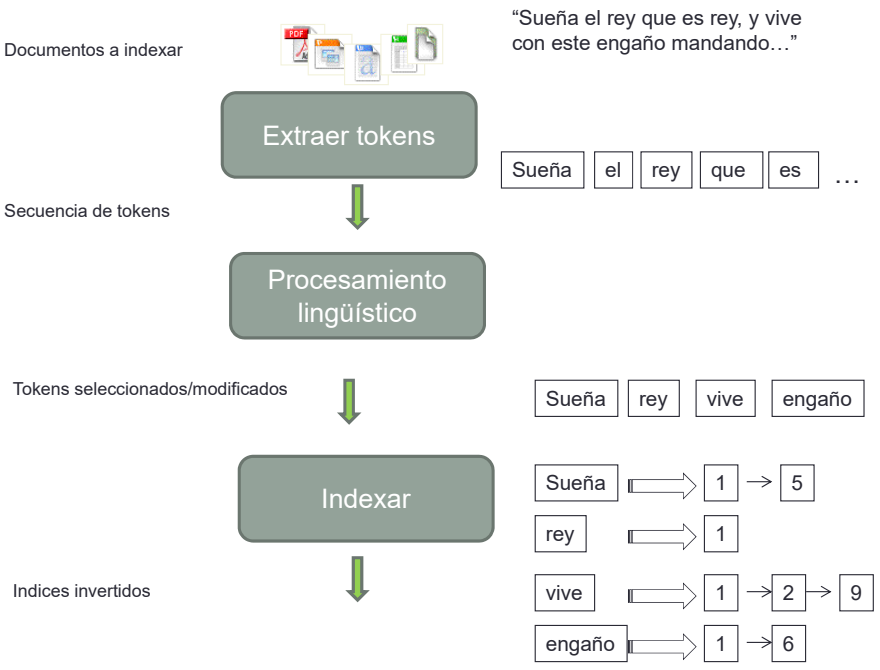
Mantenemos un diccionario de términos (usamos diccionario para la estructura de datos y vocabulario para el conjunto de términos).

Para cada término, tenemos una lista que registra en qué documentos aparece el término. Cada elemento de la lista, que registra que un término apareció en un documento (y, más tarde, a menudo, las posiciones en el documento), se denomina convencionalmente una posting (ocurrencia).

La lista se denomina postings list (lista de ocurrencias), y todas las postings lists tomadas en conjunto se denominan postings.

El diccionario de la figura se ha ordenado alfabéticamente y cada postings list está ordenada por ID de documento. Esta ordenación es útil en ciertos casos pero no siempre se realiza de ese modo.

# 4.2. Fases de construcción de un Índice invertido




Para obtener los beneficios de la velocidad en el momento de la recuperación, hay que crear el índice de antemano.

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.

# 4.3. Pasos en la indexación

**Paso 1**: Secuencia de pares (token, documento)

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

| Term | docID |
|---|---|
| I | 1 |
| did | 1 |
| enact | 1 |
| julius | 1 |
| caesar | 1 |
| I | 1 |
| was | 1 |
| killed | 1 |
| i' | 1 |
| the | 1 |
| capitol | 1 |
| brutus | 1 |
| killed | 1 |
| me | 1 |
| so | 2 |
| let | 2 |
| it | 2 |
| be | 2 |
| with | 2 |
| caesar | 2 |
| the | 2 |
| noble | 2 |
| brutus | 2 |
| hath | 2 |
| told | 2 |
| you | 2 |
| caesar | 2 |
| was | 2 |
| ambitious | 2 |

Veamos un ejemplo con una colección de dos documentos, Doc 1 y Doc 2.

La entrada para la indexación es una lista de tokens normalizados para cada documento, que podemos considerar igualmente como una lista de pares de term y docID, como en la figura.

## **Paso 2**: Ordenar por términos

| Term | docID |
|---|---|
| I | 1 |
| did | 1 |
| enact | 1 |
| julius | 1 |
| caesar | 1 |
| I | 1 |
| was | 1 |
| killed | 1 |
| i' | 1 |
| the | 1 |
| capitol | 1 |
| brutus | 1 |
| killed | 1 |
| me | 1 |
| so | 2 |
| let | 2 |
| it | 2 |
| be | 2 |
| with | 2 |
| caesar | 2 |
| the | 2 |
| noble | 2 |
| brutus | 2 |
| hath | 2 |
| told | 2 |
| you | 2 |
| caesar | 2 |
| was | 2 |
| ambitious | 2 |

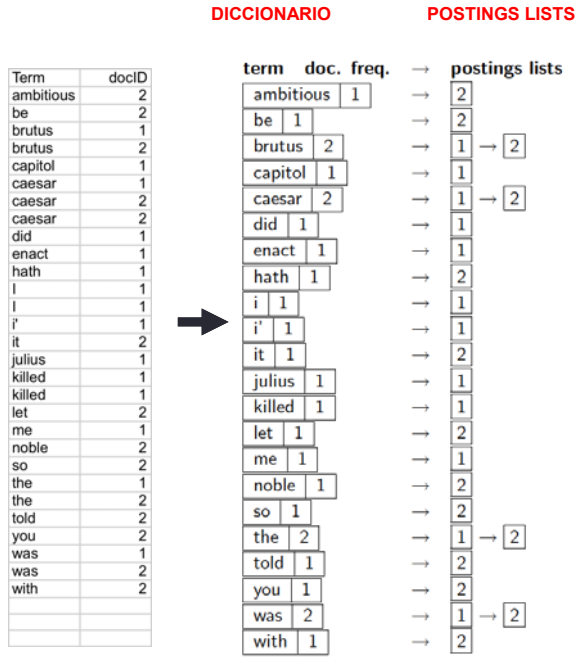
| Term | docID |
|---|---|
| ambitious | 2 |
| be | 2 |
| brutus | 1 |
| brutus | 2 |
| capitol | 1 |
| caesar | 1 |
| caesar | 2 |
| caesar | 2 |
| did | 1 |
| enact | 1 |
| hath | 1 |
| I | 1 |
| I | 1 |
| i' | 1 |
| it | 2 |
| julius | 1 |
| killed | 1 |
| killed | 1 |
| let | 2 |
| me | 1 |
| noble | 2 |
| so | 2 |
| the | 1 |
| the | 2 |
| told | 2 |
| you | 2 |
| was | 1 |
| was | 2 |
| with | 2 |

El paso principal de la indexación es ordenar alfabéticamente esta lista de términos.

## **Paso 3**: Creación del diccionario y las listas de ocurrencias (postings lists)

- Se diferencia entre diccionario y listas de ocurrencias

- Se juntan las repeticiones de un término en un mismo documento y se apunta su frecuencia.




A continuación, se combinan varias apariciones del mismo término en el mismo documento.

Las instancias del mismo término se agrupan y el resultado se divide en el diccionario y las postings, como se muestra en la figura.

Dado que un término generalmente aparece en varios documentos, esta organización de datos ya reduce los requisitos de almacenamiento.

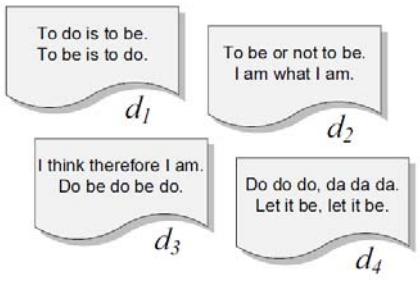
El diccionario también registra algunas estadísticas, como el número de documentos que contienen cada término (la frecuencia del documento, que es aquí también la longitud de cada lista de publicaciones). Esta información no es vital para un motor de búsqueda booleano básico, pero nos permite mejorar la eficiencia del motor de búsqueda en el momento de la consulta, y es una estadística que se utiliza posteriormente en otros modelos de recuperación.

Las postings se ordenan de forma secundaria por docID. Esto proporciona la base para un procesamiento de consultas eficiente.

Esta estructura de índice invertido es esencialmente la estructura más eficiente para soportar búsquedas.

# 4.4. Tipos de índices: índice simple

| Vocabulary | $n_i$ | Occurrences as inverted lists |
|---|---|---|
| to | 2 | [1,4],[2,2] |
| do | 3 | [1,2],[3,3],[4,3] |
| is | 1 | [1,2] |
| be | 4 | [1,2],[2,2],[3,2],[4,2] |
| or | 1 | [2,1] |
| not | 1 | [2,1] |
| I | 2 | [2,2],[3,2] |
| am | 2 | [2,2],[3,1] |
| what | 1 | [2,1] |
| think | 1 | [3,1] |
| therefore | 1 | [3,1] |
| da | 1 | [4,3] |
| let | 1 | [4,2] |
| it | 1 | [4,2] |

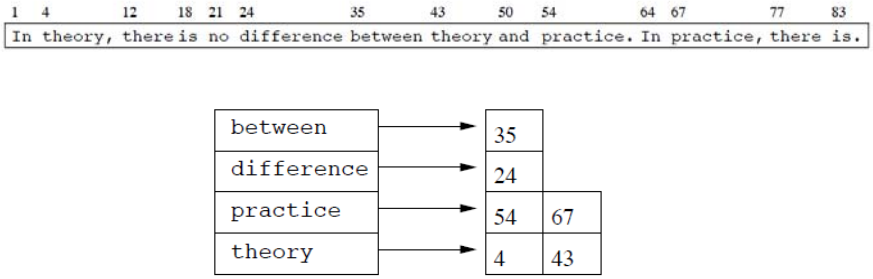



En cada entrada del diccionario se guarda el término, el número de documentos de la colección en los que aparece, es decir, la longitud de su postings list y el enlace a la misma.

En cada entrada de una postings list se guarda el ID del documento y la frecuencia de ocurrencia del término en dicho documento.

Para responder secuencias de palabras o búsqueda de palabras próximas el índice simple no sirve.

Se puede añadir información como posición de la palabra dentro del texto donde aparece. Ejemplo:

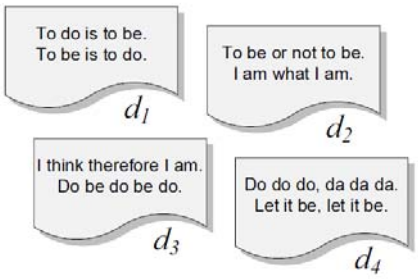



Si se requiere apuntar la posición de la palabra en el documento (caracteres o por palabras) se debe construir un ***Full inverted index***

## Ejercicio Construye un Full inverted Index, conteo de posición por palabras (no cuentan los símbolos de puntuación).

| Vocabulary | $n_i$ |
|---|---|
| to | 2 |
| do | 3 |
| is | 1 |
| be | 4 |
| or | 1 |
| not | 1 |
| I | 2 |
| am | 2 |
| what | 1 |
| think | 1 |
| therefore | 1 |
| da | 1 |
| let | 1 |
| it | 1 |

Occurrences as inverted lists

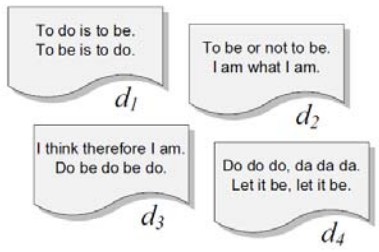



En cada entrada del diccionario se guarda el término, el número de documentos de la colección en los que aparece, es decir, la longitud de su postings list y el enlace a la misma.

En cada entrada de una postings list se guarda el ID del documento, la frecuencia de ocurrencia del término en dicho documento y las posiciones en las que aparece.

## Ejercicio Full inverted Index, conteo de posición por palabras (no cuentan los símbolos de puntuación).

| Vocabulario | $n_i$ | Ocurrencias |
|---|---|---|
| to | 2 | [1,4,[1,4,6,9]],[2,2,[1,5]] |
| do | 3 | [1,2,[2,10]],[3,3,[6,8,10]],[4,3,[1,2,3]] |
| is | 1 | [1,2,[3,8]] |
| be | 4 | [1,2,[5,7]],[2,2,[2,6]],[3,2,[7,9]],[4,2,[9,12]] |
| or | 1 | [2,1,[3]] |
| not | 1 | [2,1,[4]] |
| I | 2 | [2,2,[7,10]],[3,2,[1,4]] |
| am | 2 | [2,2,[8,11]],[3,1,[5]] |
| what | 1 | [2,1,[9]] |
| think | 1 | [3,1,[2]] |
| therefore | 1 | [3,1,[3]] |
| da | 1 | [4,3,[4,5,6]] |
| let | 1 | [4,2,[7,10]] |
| it | 1 | [4,2,[8,11]] |



Qué estructura de datos se puede usar para las *postings lists*?

- **¿Vector de talla fija?** Problemas con las inserciones y borrados.
- Se necesitan estructuras de talla variable: listas enlazadas o vectores de longitud variable.
- Compromiso entre talla/facilidad de inserción.

¿Cómo indexar eficientemente?

¿Cuánto espacio de almacenamiento necesitamos?

Algunos algoritmos de indexación y métodos de almacenamiento eficientes se verán más adelante.

# 5. PROCESADO BOOLEANO DE CONSULTAS

## **Consultas** de un único término

- La búsqueda más sencilla es encontrar todas las ocurrencias del término
- La estructura y búsqueda en el diccionario puede hacerse utilizando estructuras de datos clásicas como: Tabla Hash, Trie, B-tree,…
- Normalmente el diccionario puede caber en memoria central, mientras que las *postings lists* se almacenan en disco (acceso más lento).

## **Consultas** de más de un término

Podemos considerar dos casos:

- **Operación AND (intersección):** Se debe buscar la aparición de todos los términos de la consulta obteniendo una lista para cada término. Después hay que hacer la intersección de las listas para encontrar la solución.

- **Operación OR (unión):** Se debe buscar la aparición de todos los términos de la consulta obteniendo una lista para cada término. Después hay que hacer la unión de las listas para encontrar la solución.

- Importante: optimizar en función del orden en que se aplican los operadores.

## Ejercicio

Partiendo de la siguiente matriz de incidencia binaria, vamos a obtener la solución a las siguientes consultas:

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| Antony | 1 | 1 | 0 | 0 | 0 | 1 |
| Brutus | 1 | 1 | 0 | 1 | 0 | 0 |
| Caesar | 1 | 1 | 0 | 1 | 1 | 1 |
| Calpurnia | 0 | 1 | 0 | 0 | 0 | 0 |
| Cleopatra | 1 | 0 | 0 | 0 | 0 | 0 |
| mercy | 1 | 0 | 1 | 1 | 1 | 1 |
| worser | 1 | 0 | 1 | 1 | 1 | 0 |

Query1: ***Brutus*** *AND* ***Caesar***

Query1: ***Brutus*** *OR* ***Caesar***

Suponiendo que se dispone de la matriz de incidencia binaria, vamos a resolver las siguientes consultas.

## Ejercicio

Obtener la solución a las siguientes consultas:

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| Antony | 1 | 1 | 0 | 0 | 0 | 1 |
| Brutus | 1 | 1 | 0 | 1 | 0 | 0 |
| Caesar | 1 | 1 | 0 | 1 | 1 | 1 |
| Calpurnia | 0 | 1 | 0 | 0 | 0 | 0 |
| Cleopatra | 1 | 0 | 0 | 0 | 0 | 0 |
| mercy | 1 | 0 | 1 | 1 | 1 | 1 |
| worser | 1 | 0 | 1 | 1 | 1 | 0 |

Query1: ***Brutus*** *AND* ***Caesar***

Query1: ***Brutus*** *OR* ***Caesar***

**Cálculo y Respuesta:**

110100 *AND* 110111 = **110100**
110100 *OR* 110111 = **110111**

¿Qué pasa si lo tenemos representado mediante un índice invertido?

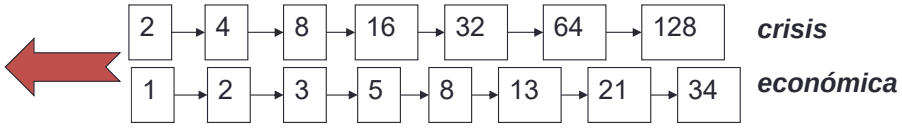
Para responder a la consulta 'Brutus AND Caesar', tomamos los vectores para 'Brutus' y 'Caesar' y luego hacemos un AND a nivel de bits.
Para responder a la consulta 'Brutus OR Caesar', tomamos los vectores para 'Brutus' y 'Caesar' y luego hacemos un OR a nivel de bits.

## Ejercicio: ***crisis*** *AND* ***económica***

1) Localizar ***crisis*** en el Diccionario y obtener su postings list.

2) Localizar ***económica*** en el Diccionario y obtener su postings list.

3) Obtener la intersección de las dos listas ("*Merge*")



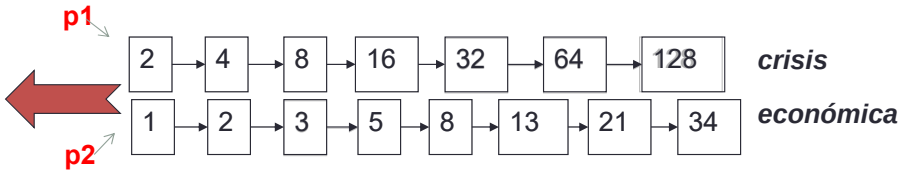

Dada una colección de documentos, vamos a resolver la consulta 'crisis AND económica' partiendo de que se ha construido un índice invertido para dicha colección.

En la diapositiva se muestran los pasos a realizar sobre el índice invertido.

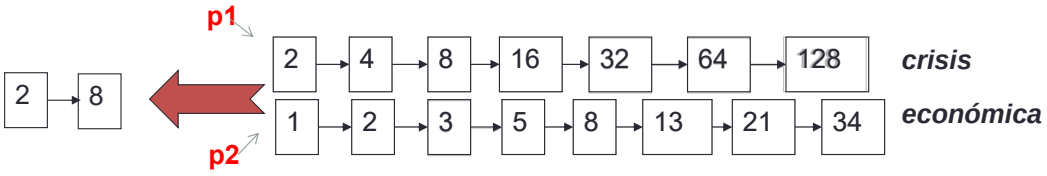
- Las listas deben estar ordenadas por docID
- Algoritmo “Intersección” : Recorre simultáneamente las dos listas extrayendo los elementos comunes.
- El coste es lineal con el número de elementos. Si las longitudes son *n* y *m* el coste es *O(n+m)*

# Ejercicio: *crisis AND económica*

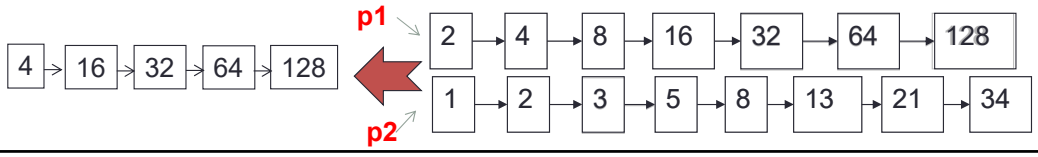
```
ALGORITMO INTERSECCION (p1, p2)
    respuesta ← {}
    mientras No_FINAL( p1) AND No_FINAL( p2)
    hacer  si docID (p1) = docID (p2)
           entonces Añadir (respuesta, docID (p1))
                    p1 ← Avanzar_Siguiente(p1)
                    p2 ← Avanzar_Siguiente(p2)
           sino  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



```
ALGORITMO INTERSECCION (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  p1 ← Avanzar_Siguiente(p1)
                   sino         p2 ← Avanzar_Siguiente(p2)
```

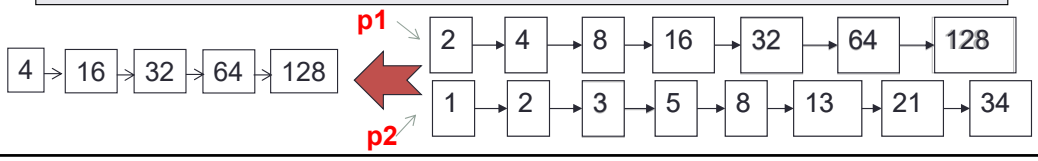


Ejercicio: Escribir el algoritmo que, a partir de las postings list correspondientes a la búsqueda de los términos A y B, nos proporciona el resultado de la consulta: (A) AND (NOT B)



Ejercicio: escribir el algoritmo que, a partir de las postings list correspondientes a la búsqueda de los términos A y B, nos proporciona el resultado de la consulta: (A) AND (NOT B)

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




El bucle final es necesario ya que puede darse el caso de que en el recorrido simultáneo de las dos listas, se llegue al final de la lista apuntada por 'p2' y no haya acabado la lista apuntada por 'p1', cuyos elementos hay que considerar para construir la solución.