# 5ª práctica de laboratorio de SIW
# "Creación de un índice"

## Motivación

Como ha quedado claro en varias de las prácticas anteriores no es viable comparar cada consulta recibida por un buscador con todos los documentos de una colección. Para acelerar el proceso de *matching* entre consultas y documentos se empleará un índice[1]—o fichero invertido. Se trata de una estructura de datos que permite encontrar un número limitado de documentos que contienen algún término de una consulta dada; será sobre ese subconjunto limitado de documentos sobre los que se determinará la similitud con la consulta.

En la práctica de hoy **se trata de construir ese índice**, en la próxima se procederá a resolver consultas usándolo. Esta quinta práctica **no tiene entregable.**

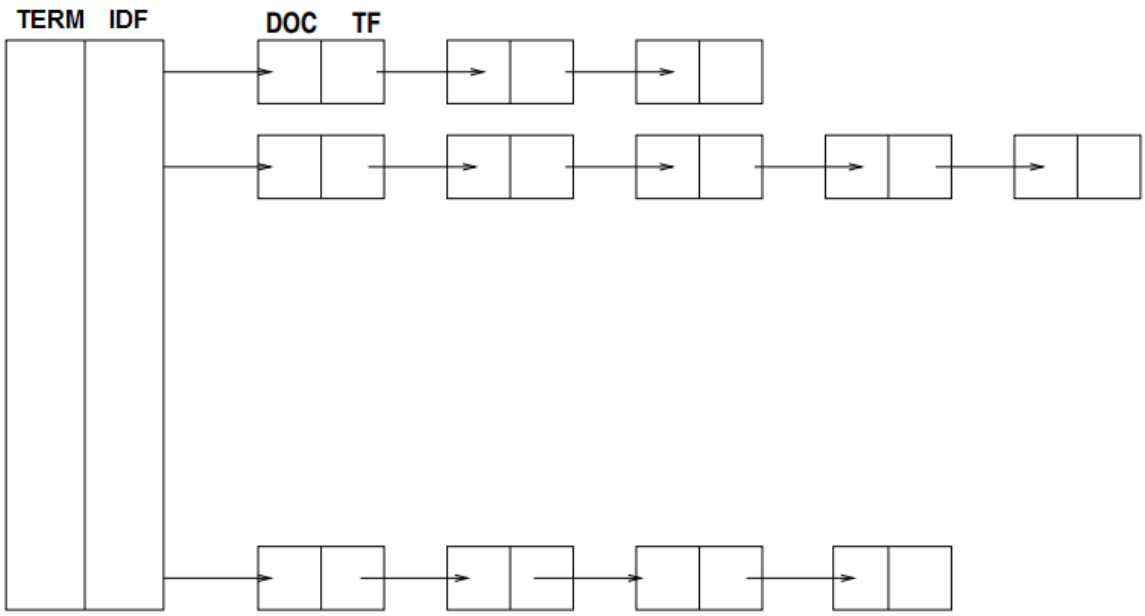
## Descripción del ejercicio

**Advertencia:** te puede ser útil repasar las transparencias de la asignatura Repositorios de Información en lo referente a índices (desde *"El fichero invertido"* hasta *"¿Cómo de grande es el fichero invertido?"*).

**El objetivo del ejercicio** de esta semana es **desarrollar un *script* en Python que permita indexar una colección de documentos**. Esto es, crear **a partir de documentos de texto** plano almacenados en disco **un índice** que debe contener la siguiente información:

- En un primer nivel **un diccionario de términos** donde cada término tendrá asociada la siguiente información: la ponderación[2] ***idf*** del término en la colección y la correspondiente ***post-list***.
- En un segundo nivel irán **las *post-lists* de cada término**. Cada *post-list* consiste en una "lista" (mejor implementarlo con un diccionario) que contendrá los identificadores de todos los documentos que contienen dicho término así como la ponderación ***tf*** del términos en cada uno de los documentos de dicha *post-list*.

[1] Si quieres saber más sobre índices consulta los libros *"Information Retrieval"* o *"Managing Gigabytes"*.
[2] Más adelante se recuerda en qué consisten las ponderaciones *idf* y *tf*.




El índice tiene que poder almacenarse en disco y cargarse completamente en memoria[3] para resolver consultas en la siguiente práctica.

Queda a elección de cada estudiante cómo se extraerán los términos de los documentos, pero se valorará positivamente el uso de tokenizadores y estematizadores, así como el paso a minúsculas de todos los términos—dicho de otro modo, se recomienda reutilizar el código implementado en prácticas anteriores que realizaba esas tareas.

**¡Atención!** Sería muy interesante, además, lo siguiente:

1. Anotar en las post-lists no solo el *tf* de cada término en cada documento sino también el valor *tf·idf*. Dicho valor, obviamente, no podrá calcularse hasta que se terminen de indexar todos los documentos. Este valor *tf·idf* no debería reemplazar al valor *tf* puesto que hay funciones de ranking—p.ej., BM25—que utilizan *tf* y *tf·idf*.
2. Una estructura de datos auxiliar que almacene el texto completo de cada documento indexado asociándolo a su identificador. Téngase en cuenta que cuando una persona lance una consulta necesitará ver el contenido de los documentos retornados además de sus identificadores.

[3] En una implementación realista el proceso de almacenamiento y carga del índice en/desde disco no es un problema trivial. Sin embargo, en este ejercicio lo más sencillo probablemente será usar la biblioteca de Python `pickle`—también le podéis echar un vistazo a `shelve` que os permite trabajar con objetos persistentes como si fueran diccionarios.

3. Una estructura de datos que almacene los valores *tf·idf* de todos los términos de cada documento o que, al menos, los vaya acumulando según se calculan esos valores para término del índice y que finalmente los suma para poder calcular con facilidad el módulo del vector de cada documento (imprescindible para implementar la función coseno).

Todas esas estructuras auxiliares no son imprescindibles para implementar la siguiente práctica **pero** permitirían que fuese mucho más eficiente.

**¡Atención! Para esta parte no hay ningún entregable en el campus virtual.** Se hará una única entrega dos semanas después de la sesión que se dedique a la sexta práctica.

Para las pruebas se recomienda utilizar cualquiera de las siguientes colecciones:

- CACM (3.204 documentos y 64 consultas).
- CISI (1.460 documentos y 112 consultas).
- Cranfield (1.400 documentos y 225 consultas).
- LISA (5.872 documentos y 35 consultas).
- Medline (1.033 documentos y 30 consultas).
- NPL (11.429 documentos y 93 consultas).
- Time (423 documentos y 83 consultas).

Cada colección está organizada de forma ligeramente diferente pero, en términos generales, todas contienen los siguientes elementos:

- El contenido textual de los documentos (normalmente en un único fichero de texto que debería procesarse).
- El contenido textual de las consultas (también en un único documento).
- Un fichero con "juicios de relevancia" (*relevance assessments*). Dicho fichero se utiliza para evaluar el rendimiento de un sistema IR y aunque no es necesario hacerlo en estas prácticas os pueden ser útiles para determinar si vuestra implementación está retornando documentos relevantes o no—hasta la sexta práctica no tienes que resolver consultas.

Se recomienda separar el proceso de carga de la colección desde el archivo de texto de los procesos de vectorización e indexado. De ese modo podrías, si lo quisieras, hacer experimentos con más colecciones cambiando el código necesario para cargarlas.

# Repaso de *tf* e *idf*

Supongamos que tenemos la siguiente colección de documentos (cada línea es un documento diferente):

- `The quick brown fox jumps over the lazy dog.`
- `The brown dog is jumped over by the fast brown fox.`
- `A fox is jumping over a dog.`
- `The brown fox jumps over the dog.`
- `The fast brown fox jumps over the sleeping dog.`

Supongamos que para vectorizar dichos documentos los pasamos a minúsculas, tokenizamos y lematizamos cada término pero no eliminamos palabras vacías; así tendríamos lo siguiente:

- `the quick brown fox jump over the lazy dog`
- `the brown dog be jump over by the fast brown fox`
- `a fox be jump over a dog`
- `the brown fox jump over the dog`
- `the fast brown fox jump over the sleep dog`

La ponderación *tf (term frequency)* es muy sencilla de entender, simplemente es la frecuencia relativa de cada término en cada documento. Si un término se repite mucho en un documento es de suponer que sea muy importante[4]. En este caso tendríamos las siguientes frecuencias absolutas:

- `brown:1 dog:1 fox:1 jump:1 lazy:1 over:1 quick:1 the: 2`
- `be:1 brown:2 by:1 dog:1 fast:1 fox:1 jump:1 over:1 the:2`
- `a:2 be:1 dog:1 fox:1 jump:1 over:1`
- `brown:1 dog:1 fox:1 jump:1 over:1 the:2`
- `brown:1 dog:1 fast:1 fox:1 jump:1 over:1 sleep:1 the:2`

Y, en consecuencia, las frecuencias relativas, los valores *tf* serían los siguientes:

- `brown:0.11 dog:0.11 fox:0.11 jump:0.11 lazy:0.11 over:0.11 quick:0.11 the: 0.22`
- `be:0.09 brown:0.18 by:0.09 dog:0.09 fast:0.09 fox:0.09 jump:0.09 over:0.09 the:0.18`

[4] Sabemos que las palabras vacías se repiten mucho y en consecuencia tendrán valores *tf* elevados. Ahí entra idf, para disminuir su peso al aparecer en prácticamente todos los documentos de la colección.

- `a:0.29 be:0.14 dog:0.14 fox:0.14 jump:0.14 over:0.14`
- `brown:0.14 dog:0.14 fox:0.14 jump:0.14 over:0.14 the:0.29`
- `brown:0.11 dog:0.11 fast:0.11 fox:0.11 jump:0.11 over:0.11 sleep:0.11 the:0.22`

**¡Atención!** En esta práctica ya no se puede representar un vector BOW *(bag of words)* como un conjunto. Una forma sencilla de hacerlo en Python sería usando un objeto **`Counter`** al que se le pase la lista con todos los tokens de cada documento.

Por lo que respecta al valor *idf (inverse document frequency)* responde—en su versión más sencilla—a la siguiente ecuación:

$$\operatorname{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Aquí *t* es un término, *D* es una colección de documentos, *N* es el número de documentos en la colección—es decir, *|D|*, y *d* es un documento de la colección que contiene el término *t*. Dicho de otro modo, cuantos más documentos de la colección contenga un documento más próximo a 0 será su valor *idf* y cuantos menos lo contengan mayor será ese valor.

Ni que decir tiene, no se puede calcular el valor *idf* de un término hasta que se ha terminado de procesar toda la colección. En nuestro caso los únicos términos que aparecen en la colección son los siguientes:

- `a`: aparece en 1 documento.
- `be`: aparece en 2 documentos.
- `brown`: aparece en 4 documentos.
- `by`: aparece en 1 documento.
- `dog`: aparece en 5 documentos.
- `fast`: aparece en 2 documentos.
- `fox`: aparece en 5 documentos.
- `jump`: aparece en 5 documentos.
- `lazy`: aparece en 1 documento.
- `over`: aparece en 5 documentos.
- `quick`: aparece en 1 documento.
- `sleep`: aparece en 1 documento.
- `the`: aparece en 4 documentos.

Puesto que la colección contiene 5 documentos (*N=5*) los valores *idf* serían los siguientes:

- `a: 0.70`
- `be: 0.40`
- `brown: 0.10`
- `by: 0.70`
- `dog: 0`
- `fast: 0.40`
- `fox: 0`
- `jump: 0`
- `lazy: 0.70`
- `over: 0`
- `quick: 0.70`
- `sleep: 0.70`
- `the: 0.10`

Puede parecer extraño que términos como *brown*, *dog*, *fox*, *jump* y *over* tengan un valor de *idf* de 0 pero es totalmente comprensible puesto que aparecen en **todos** los documentos de la colección y, en consecuencia, no aportan ninguna información para diferenciar a un documento de otro—en ese sentido son irrelevantes.

Combinando todo lo anterior el índice que se generaría para esa pequeña colección de documentos sería el siguiente:

| | | | | | | |
|---|---|---|---|---|---|---|
| (a: 0.70) | → | (doc3: 0.29) | | | | |
| (be: 0.40) | → | (doc2: 0.09) | (doc3: 0.14) | | | |
| (brown: 0.10) | → | (doc1: 0.11) | (doc2: 0.18) | (doc4: 0.14) | (doc5: 0.11) | |
| (by: 0.70) | → | (doc2: 0.09) | | | | |
| (dog: 0) | → | (doc1: 0.11) | (doc2: 0.09) | (doc3: 0.14) | (doc4: 0.14) | (doc5: 0.11) |
| (fast: 0.40) | → | (doc2: 0.09) | (doc5: 0.11) | | | |
| (fox: 0) | → | (doc1: 0.11) | (doc2: 0.09) | (doc3: 0.14) | (doc4: 0.14) | (doc5: 0.11) |
| (jump: 0) | → | (doc1: 0.11) | (doc2: 0.09) | (doc3: 0.14) | (doc4: 0.14) | (doc5: 0.11) |
| (lazy: 0.70) | → | (doc1: 0.11) | | | | |
| (over: 0) | → | (doc1: 0.11) | (doc2: 0.09) | (doc3: 0.14) | (doc4: 0.14) | (doc5: 0.11) |
| (quick: 0.70) | → | (doc1: 0.11) | | | | |
| (sleep: 0.70) | → | (doc5: 0.11) | | | | |
| (the: 0.10) | → | (doc1: 0.22) | (doc2: 0.18) | (doc4: 0.29) | (doc5: 0.22) | |

Puesto que los términos con *idf* exactamente igual a cero no aportarían absolutamente nada podrían obviarse y el índice sería entonces el siguiente:

| | | | | | |
|---|---|---|---|---|---|
| (a: 0.70) | → | (doc3: 0.29) | | | |
| (be: 0.40) | → | (doc2: 0.09) | (doc3: 0.14) | | |
| (brown: 0.10) | → | (doc1: 0.11) | (doc2: 0.18) | (doc4: 0.14) | (doc5: 0.11) |
| (by: 0.70) | → | (doc2: 0.09) | | | |
| (fast: 0.40) | → | (doc2: 0.09) | (doc5: 0.11) | | |
| (lazy: 0.70) | → | (doc1: 0.11) | | | |
| (quick: 0.70) | → | (doc1: 0.11) | | | |
| (sleep: 0.70) | → | (doc5: 0.11) | | | |
| (the: 0.10) | → | (doc1: 0.22) | (doc2: 0.18) | (doc4: 0.29) | (doc5: 0.22) |