

Programa tu propio Google

El indexador

En este primer artículo vamos a tratar la teoría de los buscadores de internet como Google: cómo funcionan, en qué se basan y cómo consiguen organizar toda esa cantidad de información.

Empezaremos desarrollando un ejemplo de la parte menos conocida de un buscador: su indexador.



- [Google = Biblioteca Digital](#)
- [Google Desktop](#)
- [Empezando por los cimientos: la base teórica](#)
 - [Esas cosas tan raras: índices de texto invertidos](#)
 - [La indexación](#)
 - [Las palabras vacías p términos superfluos](#)
 - [Los documentos contenidos en la biblioteca](#)
- [Implementación del indexador](#)
- [Conclusiones](#)
- [Los ejemplos](#)

Google = Biblioteca digital

Todos estamos muy acostumbrados a teclear “google.com” en nuestro navegador y empezar a buscar todo tipo de páginas. El “buscador” tal o el “buscador” cual está en boca de todos, sin embargo, pocos saben que en realidad, lo que estamos utilizando es una aplicación muy concreta de una “biblioteca digital”, en este caso, una biblioteca cuyos “libros” son todas las páginas que componen Internet.

El estudio e investigación de las bibliotecas digitales corresponde a una rama de la informática llamada *Information Retrieval* o “Recuperación de la información”, también abreviado como “IR”, y abarca temas como la búsqueda, recuperación, catalogación y preparación de gran cantidad de datos, ya sean estos en forma de texto, multimedia o de cualquier otro tipo.

Las bibliotecas digitales existen mucho antes de los buscadores de Internet, y sus aplicaciones han sido muy variadas: desde esos terminales de pantalla de fósforo verde que encontrábamos en nuestras bibliotecas públicas, hasta los CD-ROM con catálogos de productos o textos de todo tipo (jurídicos, científicos, publicaciones, etc.).

Podemos definir sencillamente una biblioteca digital como “una colección organizada en diversos formatos digitales para los cuales existen servicios tales como clasificación, búsqueda, recuperación y administración”. En esta definición vemos que se habla de “diversos formatos digitales”, y de ciertas tareas que podremos hacer: clasificar, buscar, recuperar y administrar.

Si pensamos en Google, el buscador por excelencia, veremos que cumple (más o menos) con esta definición: almacena una colección organizada en diversos formatos (imágenes de

cualquier tipo, textos en formato HTML o PDF, noticias de periódicos...), permitiendo su clasificación (según su relevancia), búsqueda (tecleando las palabras de búsqueda y haciendo clic en “Buscar”), recuperación (llevándonos hasta URL donde reside la información original o a través de su caché) y administración (con la página de preferencias).

Google Desktop

En los últimos meses el mundo de los buscadores está cambiando de rumbo: ya no se limitan a buscar en las páginas de Internet, sino que, con mucho acierto, han pensado que es necesario buscar dentro de nuestro propio equipo. Aunque este tema no es nuevo, y ya existían otros sistemas parecidos, Google lleva más de un año investigando en este tema, y nos empiezan a llegar los primeros frutos: Google Desktop, una aplicación que se encarga de aplicar la tecnología que todos conocemos a las búsquedas en nuestro disco duro.

Bien, como hemos prometido, en este mes vamos a desarrollar un pequeño buscador, desde la base y repasando los conceptos teóricos que necesitaremos. Llamaremos a nuestra criatura: TooPo (como habréis imaginado, este nombre tan ridículo viene del nombre de nuestra revista).



Para simplificar su desarrollo, en vez de hacer un buscador de internet, lo haremos en local, imitando a Google-Desktop, con una pequeña aplicación de escritorio escrita en Delphi, que nos permitirá buscar, a la velocidad del rayo, cualquier documento que contenga una serie de palabras.

Nos centraremos solo en una biblioteca digital orientada a textos, así que no permitiremos búsquedas sobre imágenes u otras cosas raras.

Empezando por lo cimientos: la base teórica

Para emprender esta aventura con garantías de éxito, tenemos que conocer primero las técnicas básicas de los buscadores de Internet, bueno, mejor dicho: de las bibliotecas digitales.

En un buscador existen dos procesos muy distintos e importantes: la introducción de la información (y preparación de las estructuras de búsqueda), y la búsqueda en sí. El primer proceso se llama “indexación”, o “indización” si somos más puristas. Este nombre viene del resultado que esperamos obtener: una serie de índices de búsqueda. El segundo proceso es el de “búsqueda”, que se apoyará en los índices que hemos generado en la primera etapa, concretamente en unos tipos de índices especiales llamados “índices de texto invertidos”.

Esas cosas tan raras: índices de texto invertidos

Creo yo que más o menos, todos tenemos el concepto de índice como “esa lista de claves que nos permite llegar rápidamente a los valores”. En vez de recorrer la colección completa en busca de nuestros resultados, recorreremos una lista mucho más pequeña, el índice. Si pensamos en el índice de los libros normales, su función es la misma: evitarnos pasar páginas y más páginas buscando un capítulo, si podemos hacerlo echando un vistazo rápido en dos o tres páginas, recorriendo todos los títulos con el dedo.

A la hora de buscar los textos que contengan ciertas palabras, podemos pensar en recorrer

todas las palabras de cada documento para ver si está la palabra que buscamos. Esta solución es tremendamente lenta, ya que hay que ir recorriendo toda la colección de textos (que puede ser muy muy grande), hacer un análisis de cada uno de los documentos para buscar espacios, guiones de separación, etc., y así poder concluir si la palabra buscada está o no está en el texto.

Sin embargo, podemos pensar de otra forma: mantener una lista de todas las palabras existentes en los documentos, anotando en qué documentos aparece cada palabra. A esta lista de palabras, junto con los documentos donde aparece cada una, le llamaremos “índice invertido”. Muchos libros también incluyen este tipo de índice al final de sus páginas, donde se listan todas las palabras relevantes y las páginas donde aparece cada una de estas palabras.

La indexación

La indexación es un proceso lento y costoso, pero contamos con la ventaja de que solo debe ejecutarse en el momento de crear la biblioteca digital, poniendo atención de actualizarlo cuando haya modificaciones.

La indexación consta de los siguientes pasos:

1. Recorrer todos los documentos sobre los que queremos buscar. Este puede ser un conjunto finito y conocido, por ejemplo: las páginas HTML de una carpeta, o puede ser desconocido: todas las páginas de Internet. En este último caso, los buscadores utilizan los que denominan robots o crawlers (el de Google tiene nombre propio: googlebot): son pequeños programas que va rastreando la estructura de la web en busca de nuevas páginas. Este tipo de programas (también llamados spider, en referencia a la araña que recorre la red), van recolectando páginas a través de los enlaces a otras páginas, así en un ciclo interminable. Una vez que el robot localiza una página, puede procesarla a través del siguiente paso.
2. Procesar el documento a indexar: consiste en descomponerlo hasta obtener la lista de palabras que lo forman. Este proceso puede ser muy sencillo, como en los archivos de texto plano, que tiene todas las palabras separadas por espacios u otros caracteres, o muy complejo, como un documento PDF que debe ser decodificado, separando el formato y las imágenes, extrayendo solamente el texto plano.
3. Una vez que tenemos la lista de palabras de un documento, lo único que nos queda es invertir el índice: almacenar esta lista, apuntando en qué documento hemos encontrado cada palabra. Habrá algunas palabras que solo aparezcan en un documento (la búsqueda de esa palabra nos dará un solo resultado), mientras que otras palabras más comunes aparecerán en muchos documentos.
4. Opcionalmente podemos almacenar el documento, en su estado actual, en la propia biblioteca digital. De esta forma, podemos consultar cualquier documento aunque el original deje de estar disponible.

Vamos a poner un pequeño ejemplo. Pensemos en los tres pequeños textos que aparecen a continuación:

- Texto 1: “Un coche es más rápido que un caballo”
- Texto 2: “Este es un caballo que necesita de un cuidado especial”
- Texto 3: “Cuidado con lo que digas de mi coche”

Estos tres textos serán los tres documentos que formarán nuestra raquílica biblioteca digital.

La estructura del índice invertido es la que vemos en la siguiente tabla: el índice consta de una serie de entradas, una por palabra, y en cada entrada aparecen los documentos donde se encuentra la palabra.

Palabra Aparece en...

caballo	1, 2
coche	1, 3
con	3
cuidado	2, 3
de	2, 3
digas	3
es	1, 2
especial	2
este	2
lo	3
más	1
mi	3
necesita	2
que	1, 2, 3
rápido	1
un	1, 2

Pensad que este índice puede ser enorme: tantas entradas como todas las palabras distintas que existan en toda la biblioteca digital.

En el siguiente **Listado 1** podemos ver el pseudocódigo del algoritmo que necesitamos para crear este índice invertido.

Listado 1

```

.....
FOR "todos los documentos de la colección"
  FOR "todas las palabras del documento actual"
    Entrada = "buscar en el índice la palabra actual"
    IF "entrada no encontrada" THEN
      "Crear una entrada en el índice para palabra actual"
      Entrada = "entrada recién creada"
    END-IF
    "meter documento actual en la lista de Entrada"
  END-FOR
END-FOR

```

Como podéis ver, el proceso de indexación recorre todas las palabras, sin excepción alguna. El resultado, que ya hemos visto en la anterior tabla, es el índice invertido donde aparecen todas las palabras, incluso algunas tan repetidas como “que”, “de”, “un”, etc. ¿Es esto correcto?

Las palabras vacías o términos superfluos

Si abrimos Google e introducimos la búsqueda “domicilio en salamanca de juanita y perico”, se nos informará de que “Las siguientes palabras son muy comunes y no se incluyeron en su búsqueda: en de y”. ¿Qué significa esto? ¿Acaso no puedo buscar frases completas, incluyendo preposiciones y conjunciones? Sí, las búsquedas permiten todo tipo de palabras, pero algunas de ellas no aportan ninguna información a la consulta.

Al conjunto de palabras que consideramos superfluas llamaremos “lista de palabras vacías” (o “lista de parada” o “stoplist”) y tendremos que ignorarlas tanto a la hora de construir los índices como de realizar las búsquedas. Otra posibilidad es ignorar también las palabras que no lleguen a una longitud mínima. De esta forma conseguiremos ignorar palabras muy

comunes (como las conjunciones), signos y dígitos: “a”, “y”, “de”, “%” etc.

Para ello, debemos modificar ligeramente el algoritmo que vimos en el **Listado 1**, tal como aparece en el siguiente **Listado 2**. Simplemente tenemos que controlar que la palabra a introducir en el índice no sea una palabra vacía y que tenga un mínimo de caracteres.

Listado 2

```
.....  
  
FOR "todos los documentos de la colección"  
  FOR "todas las palabras del documento actual"  
    IF "palabra actual no es palabra vacía"  
      AND "palabra actual tiene un mínimo de X caracteres"  
    THEN  
      Entrada = "buscar en el índice la palabra actual"  
      IF "entrada no encontrada" THEN  
        "Crear una entrada en el índice para palabra actual"  
        Entrada = "entrada recién creada"  
      END-IF  
      "meter documento actual en la lista de Entrada"  
    END-IF  
  END-FOR  
END-FOR
```

Aunque parezca una optimización tonta, tenemos que tener en cuenta que en un idioma como el castellano, las preposiciones y conjunciones suponen un alto porcentaje de todas las palabras del texto, así que, eliminando estas palabras, conseguiremos reducir considerablemente el tamaño de los índices y así acelerar tanto la búsqueda como la indexación. La lista de palabras vacías podría variar dependiendo de las características de los textos a indexar.

En el algoritmo de búsqueda también tenemos que hacer un pequeño cambio: eliminar las palabras vacías y las pequeñas de la cadena de búsqueda. Por ejemplo, aunque la consulta que enviamos a Google fue “domicilio en salamanca de juanito y perico”, se transforma automáticamente en la consulta “domicilio salamanca juanito perico”, una vez eliminados los términos superfluos.

Los documentos contenidos en la biblioteca

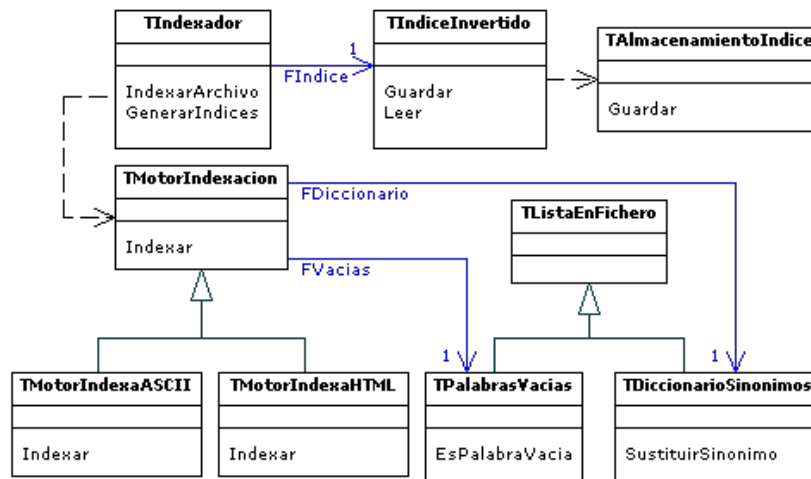
Como decíamos, las bibliotecas digitales pueden, opcionalmente, almacenar los propios documentos que indexan. Ya no se trata de almacenar la ruta (o URL) donde podemos localizar el documento, sino que podemos almacenar el contenido del documento para extraerlo completamente. De este modo, teniendo acceso a la biblioteca digital podemos acceder al contenido de los documentos, aunque estos ya no existan o no estén disponibles en este momento. Google consigue esto a través de lo que denominan el caché de páginas, donde hacen una copia de la página en el momento que fue indexada. Gracias a él, podemos acceder a páginas que han desaparecido, simplemente haciendo clic en el enlace del caché.

Y por si fuera poco, también podemos almacenar la posición, dentro del documento, en que se ha encontrado cada palabra. Es decir: cuando indexamos un documento lo descomponemos en palabras, almacenamos las palabras en el índice invertido, almacenamos el documento completo en el caché, y podemos almacenar también la posición donde se encontró cada palabra. Haciendo esto, podemos extraer el párrafo donde aparece el texto buscado, o presentarlo en un color destacado, como hace Google en su caché.

Implementación del indexador

En este archivo podréis encontrar el [código fuente de nuestro indexador](#). Hemos intentado simplificar al máximo la implementación, así que solo procesará una serie de archivos, indexando aquellos que sean de texto plano o HTML. De todas formas, dejaremos la estructura abierta para que vosotros mismos podáis modificarlo para indexar otros formatos y otras colecciones de archivos.

En la siguiente imagen podéis ver un pequeño diagrama UML que describe las clases que hemos utilizado.



- **TIndexador**: se trata de la clase principal, la cual utilizaremos desde el exterior. Tan solo debemos crear una instancia de este objeto, indicar en qué carpeta se encuentran nuestros archivos con la biblioteca digital, e invocar al método “IndexarArchivo” por cada uno de los documentos que queramos incluir en la biblioteca. Una vez que hayamos terminado de indexar archivos, llamaremos al método “Generar”, y se crearan los índices invertidos de los archivos que hayamos indexado.
- **TMotorIndexacion**: es una clase interna (no se debe utilizar desde el exterior) y abstracta, que nos permite definir distintos tipos de indexación, una por cada formato soportado. El objetivo de los descendientes de esta clase es obtener una lista de palabras a partir de un bloque de memoria con el contenido a indexar. Si quisiéramos soportar un nuevo formato (páginas web, documentos PDF, etc.), tendríamos que crear un descendiente de esta clase.
- **TMotorIndexacionASCII**: genera la lista de palabras para archivo en texto plano. Esta operación es relativamente sencilla, ya que tan solo consiste en ir extrayendo las palabras del archivo de texto, poniendo cuidado a los distintos separadores que pueden existir: espacio, signos de puntuación, etc.
- **TMotorIndexacionHTML**: genera la lista de palabras para una página web en HTML. Esta operación algo más compleja que la anterior, ya que hay que ir buscando la etiquetas del HTML, indexando los que se encuentre solo entre etiquetas y no en los atributos. Por ejemplo, en el siguiente trozo de HTML:

```
<b>el texto</b>
```

Se indexará la frase “el texto” pero no “descripción del icono”.

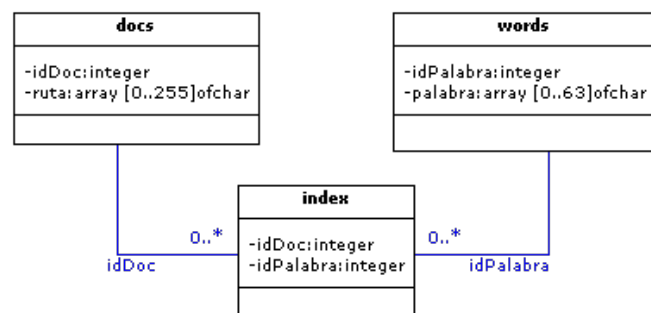
- **TPalabrasVacias**: esta clase mantendrá una lista de palabras que consideramos

vacías, y nos informará si una palabra dada está considerada como vacía o no.

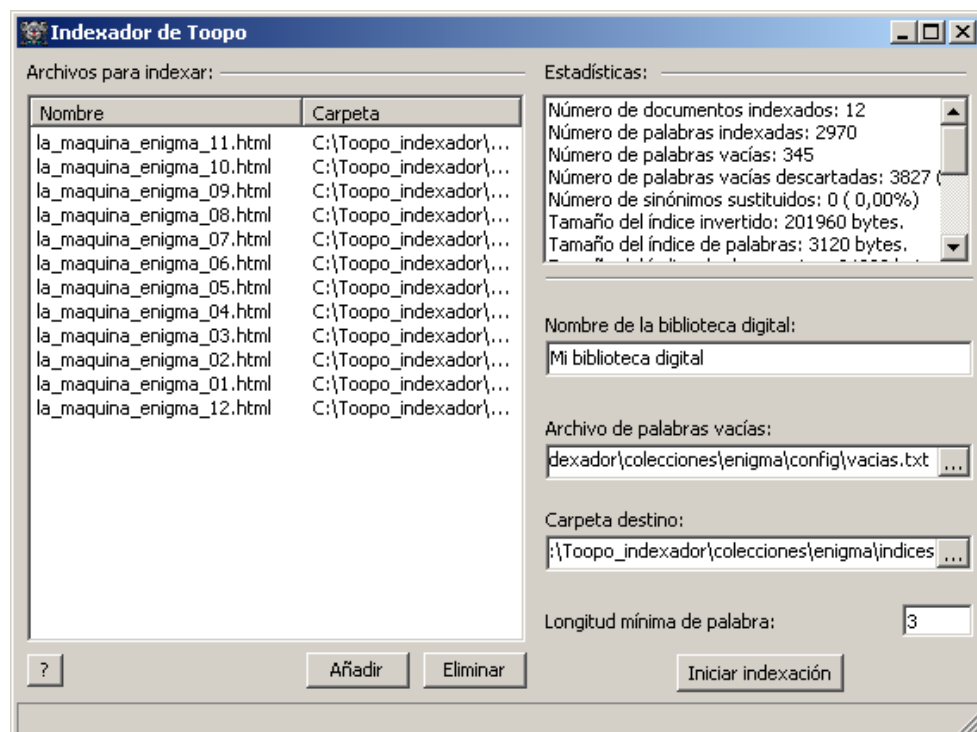
- **TDiccionarioSinonimos**: al igual que **TPalabrasVacias**, con esta clase mantendremos una lista de sinónimos, proporcionando un método para encontrar el sinónimo de una palabra. (lo veremos en la próxima entrega)
- **TListaEnFichero**: como las clases **TPalabrasVacias** y **TDiccionarioSinonimos** tienen una función muy parecida (son listas almacenadas en un fichero), heredan de esta clase que codifica el código común.
- **TIndiceInvertido**: representa la estructura del índice invertido que ya hemos explicado. Se crea a partir de todas las listas de palabras extraídas (con los motores de indexación) de todos los documentos. Además, asigna identificadores a cada documento y palabra.
- **TAlmacenamientoIndice**: esta clase nos guarda y lee del disco el índice invertido representado por la clase **TIndiceInvertido**.

El resultado de nuestro indexador serán tres archivos: “docs.dat”, “words.dat” e “index.dat”, que formarán juntos la estructura del índice invertido.

En la imagen de la derecha podéis ver los datos que se almacena en cada archivo y las relaciones entre ellos.



En la siguiente imagen podéis ver el aspecto de nuestro indexador en ejecución:



Conclusiones

Bueno, después de este artículo creo que hemos aprendido algo más sobre lo que ocurre cada vez que hacemos una búsqueda en Google.


Lo más importante que hemos visto han sido las partes de las que se compone una biblioteca digital como Google, y la teoría sobre la que se sustenta y cómo podemos hacer una implementación muy sencilla de un indexador en Delphi.

En [el próximo artículo](#) tratamos el tema del buscador, basándonos en lo que hemos aprendido en este artículo, en el código que hemos escrito y en los índices que hemos generado con el indexador de Toopo.

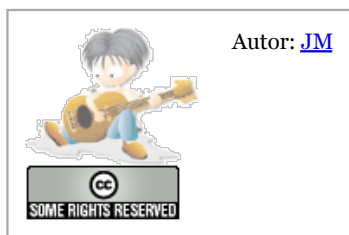
Los ejemplos

Delphi 6

La primera parte de nuestro buscador Toopo: las clases comunes de la arquitectura general (usadas tanto por el indexador y el buscador) y la parte del indexador funcionando.

 [Fuentes en ZIP](#)

 [El ejecutable ya compilado](#)



Autor: [JM](#)



Este artículo apareció publicado por primera vez en el número 9 de la revista Todo Programación, editada por [Studio Press, S.L.](#) y se reproduce aquí con la debida autorización.



[Página principal](#)



[Programación](#)



[Delphi](#)



[Volver](#)



2006 by [JM](#)

