Búsqueda y Minería de Información 2020-2021

Universidad Autónoma de Madrid, Escuela Politécnica Superior

Grado en Ingeniería Informática 4º curso

Práctica 2 – Motores de búsqueda e indexación

Fechas

- Comienzo: lunes 1 / martes 2 de marzo
- Entrega: martes 6 de abril (14:00h)

Objetivos

Los objetivos de esta práctica son:

- La implementación eficiente de funciones de ránking, particularizada en el modelo vectorial.
- La implementación de índices eficientes para motores de búsqueda.
- La implementación de un método de búsqueda proximal.
- La dotación de estructuras de índice posicional que soporten la búsqueda proximal.
- La implementación del algoritmo PageRank.

Se desarrollarán implementaciones de índices utilizando un diccionario y listas de postings. Y se implementará el modelo vectorial utilizando estas estructuras más eficientes para la ejecución de consultas.

Los ejercicios básicos consistirán en la implementación de algoritmos y técnicas estudiados en las clases de teoría, con algunas propuestas de extensión opcionales. Se podrá comparar el rendimiento de las diferentes versiones de índices y buscadores, contrastando la coherencia con los planteamientos estudiados a nivel teórico.

Mediante el nivel de abstracción seguido, se conseguirán versiones intercambiables de índices y buscadores. El único buscador que no será intercambiable es el de Whoosh, que sólo funcionará con sus propios índices.

Material proporcionado

Se proporcionan:

- Varias clases e interfaces Python en un comprimido src.zip, con las que el estudiante integrará las suyas propias.

  Las clases parten del código de la práctica anterior con algún ligero retoque y renombrado en el diseño (se adjunta explicación de los mismos en el fichero cambios.txt).

  Igual que en la práctica 1, el módulo `main.py` incluye un programa que deberá funcionar con las clases a implementar por el estudiante.
- Las colecciones de prueba de la práctica 1: toys.zip (que se descomprime en dos carpetas toy1 y toy2), docs1k.zip con 1.000 documentos HTML y un pequeño fichero urls.txt.
- Una colección más grande: docs10k.zip con 10.000 documentos HTML.
- Varios grafos para probar PageRank: graphs.zip.
- Un documento de texto output.txt con la salida estándar que deberá producir la ejecución del programa main.

Ejercicios

1. Implementación de un modelo vectorial eficiente

Se mejorará la implementación de la práctica anterior aplicando algoritmos estudiados en las clases de teoría. En particular, se utilizarán listas de postings en lugar de un índice forward.

La reimplementación seguirá haciendo uso de la clase abstracta `Index`, y se podrá probar con cualquier implementación de esta clase (tanto la implementación de índice sobre Whoosh como las propias).

1.1 Método orientado a términos (3pt)

Escribir una clase `TermBasedVSMSearcher` que implemente el modelo vectorial coseno por el método orientado a términos.

1.2 Método orientado a documentos* (1pt)

Implementar el método orientado a documentos (con heap de postings) en una clase `DocBasedVSMSearcher`.

1.3 Heap de ránking (0.5pt)

Reimplementar la clase entregada `SearchRanking` (módulo bmi.search) para utilizar un heap de ránking. Nótese que esta opción se aprovecha mejor con la implementación orientada a documentos, aunque es compatible con la orientada a términos.

2. Índice en RAM (3pt)

Implementar un índice propio que pueda hacer las mismas funciones que la implementación basada en Whoosh definida en la práctica 1. Como primera fase más sencilla, los índices se crearán completamente en RAM. Se guardarán a disco y leerán de disco en modo serializado (ver librería *pickle*).

Para guardar el índice se utilizarán los nombres de fichero definidos por las variables estáticas de la clase `Config`.

Antes de guardar el índice, se borrarán todos los ficheros que pueda haber creados en el directorio del índice. Asimismo, el directorio se creará si no estuviera creado, de forma que no haga falta crearlo a mano. Este detalle se hará igual en los siguientes ejercicios.

2.1 Estructura de índice

Implementar la clase `RAMIndex` como subclase de `Index` con las estructuras necesarias: diccionario, listas de postings, más la información que se necesite.

Para este ejercicio en las listas de postings sólo será necesario guardar los docIDs y las frecuencias; no es necesario almacenar las posiciones de los términos.

2.2 Construcción del índice

Implementar la clase `RAMIndexBuilder` como subclase de `Builder`, que cree todo el índice en RAM a partir de una colección de documentos.

3. Índice en disco* (1pt)

Reimplementar los índices definiendo las clases `DiskIndex` y `DiskIndexBuilder` de forma que:

- El índice se siga creando entero en RAM (por ejemplo, usando estructuras similares a las del ejercicio 2).
- Pero el índice se guarde en disco dato a dato (docIDs, frecuencias, etc.).
- Al cargar el índice, sólo el diccionario se lee a RAM, y se accede a las listas de postings en disco cuando son necesarias (p.e. en tiempo de consulta).

Se sugiere guardar el diccionario en un fichero y las listas de postings en otro, utilizando los nombres de fichero definidos como variables estáticas en la clase `Config`.

4. Motor de búsqueda proximal* (1pt)

Implementar un método de búsqueda proximal en una clase `ProximitySearcher`, utilizando las interfaces de índices posicionales. Igual que en las prácticas anteriores, se sugiere definir esta clase como subclase (directa o indirecta) de `Searcher`. Para empezar a probar este buscador, se proporciona una implementación de indexación posicional basada en Whoosh (`WhooshPositionalIndex`).

5. Índice posicional* (1pt)

Implementar una variante adicional de índice (como subclase si se considera oportuno) que extienda las estructuras de índices con la inclusión de posiciones en las listas de postings. La implementación incluirá una clase `PositionalIndexBuilder` para la construcción del índice posicional así como una clase `PositionalIndex` para proporcionar acceso al mismo.

6. PageRank (1pt)

Implementar el algoritmo PageRank en una clase `PagerankDocScorer`, que permitirá devolver un ranking de los documentos de manera similar a como hace un `Searcher` (pero sin recibir una consulta).

Se recomienda, al menos inicialmente, llevar a cabo una implementación con la que los valores de PageRank sumen 1, para ayudar a la validación de la misma. Posteriormente, si se desea, se pueden escalar (o no, a criterio del estudiante) los cálculos omitiendo la división por el número total de páginas en el grafo. Será necesario tratar los nodos sumidero tal como se ha explicado en las clases de teoría.

Indicaciones

Se sugiere trabajar en la práctica de manera incremental, asegurando la implementación de soluciones sencillas y mejorándolas de forma modular (la propia estructura de ejercicios plantea ya esta forma de trabajar).

En esta misma línea, para el ejercicio 3 se sugiere inicialmente guardar en disco las estructuras de índice en modo texto para poder depurar los programas. Una vez asegurada la corrección de los programas, puede ser más fácil pasar a modo binario o serializable (usando la librería *pickle* de Python).

Se podrán definir clases o módulos adicionales a las que se indican en el enunciado, por ejemplo, para reutilizar código. Y el estudiante podrá utilizar o no el software que se le proporciona, con la siguiente limitación:

- El programa **main (módulo main.py) deberá ejecutar** correctamente **sin ninguna modificación** (más allá de comentar aquellos ejercicios que no se hayan realizado).

Por otra parte, **en la memoria se reportarán los datos del coste y rendimiento** de cada tipo de índice implementado en una tabla como la siguiente (indicando las características del procesador del ordenador utilizado):

	Construcción del índice			Carga del índice	
	Tiempo de indexado	Consumo máx. RAM	Espacio en disco	Tiempo de carga	Consumo máx. RAM
Toy1					
Toy2					
1K					
10K					

Asimismo, se recomienda indexar sin ningún tipo de stopwords ni stemming, para poder hacer pruebas más fácilmente con ejemplos "de juguete".

Entrega

La entrega consistirá en un único fichero zip con el nombre **bmi-p2-XX.zip**, donde XX debe sustituirse por el número de pareja (01, 02, …, 10, …). Este fichero contendrá:

- Una carpeta **src/** con todos los ficheros con las implementaciones solicitadas en los ejercicios.
- En su caso, un fichero **readme_lib.txt** indicando las librerías adicionales que fuesen necesarias (no se incluirán Whoosh ni BeautifulSoup) así como su forma de instalarlas (comando pip, por ejemplo). Se recomienda no obstante consultar con el profesor antes de hacer uso de librerías adicionales.
- Una **memoria bmi-p2-XX.pdf** donde se documentará:
  - En una primera sección, un listado sucinto de a) qué ejercicios y opciones se han realizado exactamente y b) en los correspondientes apartados, qué estructuras de datos se han utilizado para el diccionario y las listas de postings. Se enfatizarán en su caso los aspectos que puedan evaluarse favorablemente.
  - En el ejercicio 5 (si se entrega), indicar además qué tipo de índice (RAM, disco, etc., se ha implementado). Se enfatizarán en su caso los aspectos que puedan evaluarse favorablemente.
  - Un diagrama de clases indicando cómo se relacionan las clases implementadas.
  - Una sección donde se analicen resumidamente las diferencias de rendimiento observadas entre las diferentes implementaciones que se han creado y probado para cada componente.
  - Y cualquier otro aspecto que el estudiante considere oportuno destacar.

  La calidad de la memoria representará orientativamente el 10% de la puntuación de los ejercicios que se documentan en la misma.

No se deberán incluir en el .zip las colecciones proporcionadas por el profesor, ni los archivos de proyecto del entorno de desarrollo que se esté usando. El fichero de entrega se enviará por el enlace habilitado al efecto en el curso **Moodle** de la asignatura.

Calificación

Esta práctica se calificará con una puntuación de 0 a 10 atendiendo a las puntuaciones individuales de ejercicios y apartados dadas en el enunciado. No obstante, aquellos ejercicios marcados con un asterisco (*) tienen una complejidad un poco superior a los demás (que suman 7.5 puntos), y permiten, si se realizan todos, una nota superior a 10.

El peso de la nota de esta práctica en la calificación final de prácticas es del **40%**.

La calificación se basará en a) el **número** de ejercicios realizados y b) la **calidad** de los mismos. La calidad se valorará por los **resultados** conseguidos (economía de consumo de RAM, disco y tiempo; tamaño de las colecciones que se consigan indexar) pero también del **mérito** en términos del interés de las técnicas aplicadas y la buena programación.

La puntuación que se indica en cada apartado es orientativa, en principio se aplicará tal cual se refleja pero podrá matizarse por criterios de buen sentido si se da el caso.

Para dar por válida la realización de un ejercicio, el código deberá funcionar (a la primera) integrado con las clases que se facilitan. El profesor comprobará este aspecto añadiendo los módulos entregados por el estudiante a los módulos facilitados en la práctica, ejecutando el programa `main.py` así como otros main de prueba adicionales.