Búsqueda y Minería de Información. Práctica 2.

Daniel Santo-Tomás y Juan Velasco

Abril 2021

1. Introducción

En esta segunda práctica implementaremos nuevas funciones de ranking más eficientes así como diferentes índices para la indexación de documentos. También implementaremos el algoritmo de búsqueda proximal con su correspondiente índice para poder aplicarlo. Por último implementaremos el algoritmo *PageRank* de puntuación de páginas web.

2. Ejercicios realizados y sus implementaciones

En esta práctica hemos decidido realizar todos los ejercicios tanto los obligatorios como los opcionales así que vamos a repasar apartado a apartado cómo son sus implementaciones.

2.1. Implementación de un modelo vectorial eficiente

La implementación tanto del método orientado a términos como el orientado a documentos no contiene ninguna implementación especial. Hemos usado tanto para el heap de ranking como el de orientado a documentos la librería *heapq*. Destacar que en la implementación de *SearchRanking* hemos implementado el *cutoff* de la siguiente manera: si el tamaño del heap es igual al *cutoff* sacamos el primer elemento, que es el más pequeño por ser un minheap, y comparamos con el nuevo elemento; si este nuevo elemento es mayor, lo introducimos y si no lo es, volvemos a introducir el elemento antiguo.

2.2. Índice en RAM

En el caso de índice en RAM se han creado las siguientes estructuras:

- **Diccionario de palabras:** conjunto de python.
- **Rutas de los archivos:** lista en la que se accede a cada path por su *docid*.

- **Lista de postings:** diccionario de python en el que cada clave es una palabra y su valor es una lista en la que las posiciones pares son los *docids* y las posiciones impares son el número de apariciones de la palabra.

Todo se guarda en archivos tipo pickle.

2.3. Índice en disco

En el caso de índice en disco se han creado las siguientes estructuras:

- **Diccionario de palabras:** diccionario en python en el que cada clave es un una palabra y el valor es una lista de dos elementos que representa el inicio y el final de la lista de postings en el archivo.
- **Rutas de los archivos:** lista en la que se accede a cada path por su *docid*.
- **Lista de postings:** archivo en el que se guarda los postings de la siguiente manera: "docid num_apariciones üno detrás de otro.

Todo se guarda, menos la lista de postings, en archivos tipo pickle.

2.4. Índice posicional y búsqueda proximal

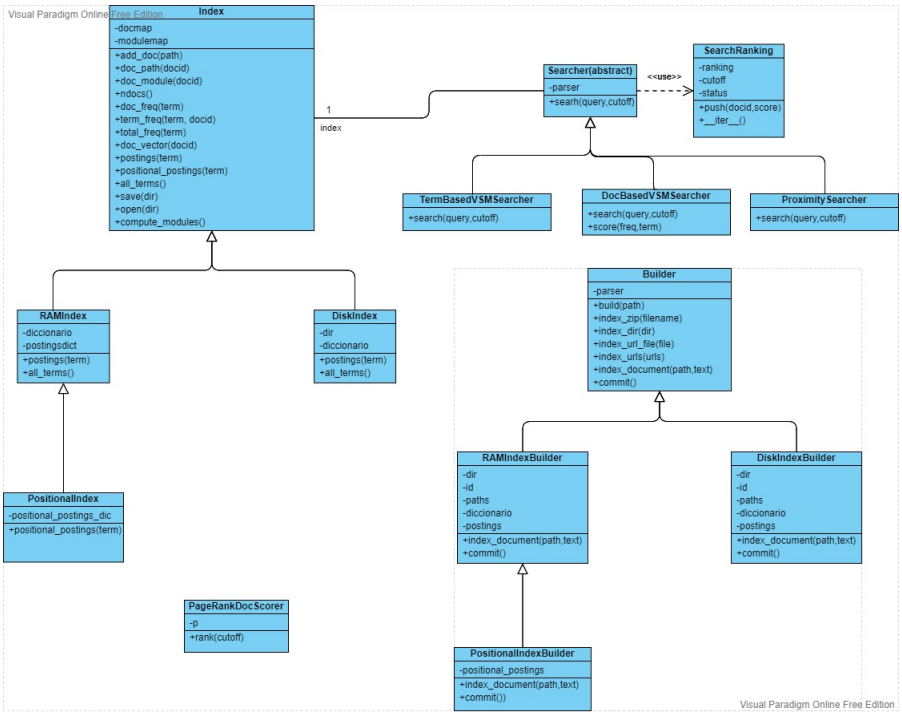
En el índice posicional destacar que hemos usado otra vez el heap de ranking creado en el ejercicio 1. En el caso de índice en posicional se ha heredado del índice en RAM y por tanto se han utilizado todas sus estructuras. Solamente hemos añadido una nueva estructura llamada **positional_postings** que es un diccionario en python cuyas claves son las palabras del diccionario y sus valores son otro diccionario en python cuyas claves son los *docid* asociado a cada documento y sus valores las listas de posiciones de la palabra dentro del documento. Esta estructura es guardada en un archivo tipo pickle.

2.5. PageRank

Únicamente destacar que no hemos usado el algoritmo de *PageRank* que tiene en cuenta los nodos sumidero y por tanto no suman 1 el resultado de todos los valores. 1

3. Diagrama de clases

Se expone a continuación un diagrama de clases que representa las interacciones entre las clases implementadas



4. Coste y Rendimiento

Para los datos de coste y rendimiento se ha usado el procesador *Intel(R) Core(TM) i7-10710U CPU @ 1.10GHz* con 16 GB de RAM. Se adjunta un archivo con información ampliada del procesador.

Vamos a realizar las tablas de rendimiento por cada uno de los índices así como un pequeño análisis de los resultados.

4.1. Tablas de Rendimiento

WhooshIndex					
	Construcción del índice			Carga del índice	
Colección	Tiempo de indexado en segundos	Consumo máx. RAM en bytes	Espacio en disco en bytes	Tiempo de carga en segundos	Consumo máx. RAM en bytes
Toy1	0.00637	34410496	12736	0.00039	34443264
Toy2	0.00588	34467840	12102	0.00039	34549760
1K	48.324	196288512	22841239	0.0055	166596608
10K	312.890	1224585216	95728208	0.0243	587079680

Tabla 1: WhooshIndex

WhooshForwardIndex					
	Construcción del índice			Carga del índice	
Colección	Tiempo de indexado en segundos	Consumo máx. RAM en bytes	Espacio en disco en bytes	Tiempo de carga en segundos	Consumo máx. RAM en bytes
Toy1	0.00724	34443264	13740	0.00037	34443264
Toy2	0.00649	34545664	12858	0.00037	34549760
1K	52.558	220831744	28601480	0.0053	166604800
10K	340.874	1224314880	141085865	0.0212	587223040

Tabla 2: WhooshForwardIndex

WhooshPositionalIndex					
	Construcción del índice			Carga del índice	
Colección	Tiempo de indexado en segundos	Consumo máx. RAM en bytes	Espacio en disco en bytes	Tiempo de carga en segundos	Consumo máx. RAM en bytes
Toy1	0.00640	34443264	13342	0.00037	34443264
Toy2	0.00597	34549760	12647	0.00037	34549760
1K	50.345	224858112	31287603	0.0069	175247360
10K	326.962	1255317504	158675051	0.0330	713265152

Tabla 3: WhooshPositionalIndex

RAMIndex					
	Construcción del índice			Carga del índice	
Colección	Tiempo de indexado en segundos	Consumo máx. RAM en bytes	Espacio en disco en bytes	Tiempo de carga en segundos	Consumo máx. RAM en bytes
Toy1	0.00036	34443264	1514	0.00022	34443264
Toy2	0.00032	34549760	1409	0.00021	34549760
1K	33.274	112168960	5930728	0.9425	160718848
10K	231.863	289918976	35520125	3.7945	661147648

Tabla 4: RAMIndex

DiskIndex					
	Construcción del índice			Carga del índice	
Colección	Tiempo de indexado en segundos	Consumo máx. RAM en bytes	Espacio en disco en bytes	Tiempo de carga en segundos	Consumo máx. RAM en bytes
Toy1	0.00057	34443264	1209	0.00202	34443264
Toy2	0.00052	34549760	1107	0.00197	34549760
1K	33.702	117936128	6397209	4.6461	145047552
10K	238.057	316342272	43950957	18.0790	474718208

Tabla 5: DiskIndex

PositionalIndex					
	Construcción del índice			Carga del índice	
Colección	Tiempo de indexado en segundos	Consumo máx. RAM en bytes	Espacio en disco en bytes	Tiempo de carga en segundos	Consumo máx. RAM en bytes
Toy1	0.00313	34443264	2606	0.00297	34443264
Toy2	0.00327	34549760	2459	0.00300	34549760
1K	36.544	355545088	17770827	1.3064	345362432
10K	262.416	1866969088	114085747	8.8921	1891467264

Tabla 6: PositionalIndex

4.2. Análisis de Resultados

Empezamos comparando nuestros índices con los de Whoosh. Lo primero que observamos es que el tiempo de creación de nuestros índices es mucho menor que el de los índices de Whoosh mientras que el tiempo de carga de nuestros índices es mucho mayor que el de los índices de Whoosh. También observamos que, tanto el espacio en disco como el consumo de RAM en la creación de nuestros índices es menor que el de Whoosh mientras que el consumo de RAM en la carga del índice es mucho menor en los índices Whoosh que en nuestros índices.

Todas estas diferencias nos indican que los índices Whoosh optimizan mucho la carga del índice por encima de su creación, para que a la hora de utilizarlo sea más rápido que a la hora de crearlo.

Comparemos ahora solo nuestros índices. Lo primero que debemos comentar es que el *PositionalIndex* es una apmliación del *RAMIndex*, lo que hace que siempre consuma más RAM, tiempo y disco del que lo hace el de RAM. Por tanto solo nos queda comprar *RAMIndex* y *DiskIndex*. El tiempo de construcción del índice es muy similar en ambos casos. Sin embargo, el tiempo de carga del índice en disco es mucho mayor (sobre todo en las colecciones más grandes) que en el índice en RAM. Esto es debido a que el cálculo de los módulos es más lento al tener que acceder a los postings en disco. El consumo de RAM en ambos casos es parecido. Solo existe una ligera diferencia a favor del índice en disco en la creación, que se debe a que la estructura usada a la hora de crear el índice es algo más compleja que la estructura de RAM, mientras que en la carga el consumo del índice en RAM es mayor que el de disco. Esto es debido a que cargamos toda la estructura creada anteriormente. Por último vemos que con colecciones más pequeñas el índice en disco es más pequeño mientras que conforme aumentan las colecciones el tamaño del índice en disco es mayor que el de RAM.