Estructura de Datos

Alumno: Gallozo, Luis Angel 156308

1) **Codifique un script que indexe una colección[1] que requiera el volcado parcial a disco (asumiendo que existe un límite de memoria). Su script debe recibir un parámetro *n* que indica cada cuántos documentos se debe hacer el volcado a disco. Al finalizar, debe unir (merge) los índices parciales. Para las pruebas use la colección snapshot de Wikipedia[2] y varios valores de *n* (por ejemplo, n = 10% del tamaño de la colección). Registre los tiempos de indexación y de merge por separado. Grafique la distribución de tamaños de las posting lists. Calcule el overhead de su índice respecto de la colección. Calcule el overhead para cada documento. ¿Qué conclusiones se pueden extraer?**

El script se ejecuta con “python app.py”. Recordar que en el archivo constantes.py estan definidos:

```
MIN_LEN_TOKEN = 3
MAX_LEN_TOKEN = 500
LEN_POSTING_CHUNK = 8# 4bytes id_term + 4bytes docid + 4bytes docfrec
LEN_POSTING=8 #4bytes docid + 4bytes docfrec
LIMIT_DOCS=122000  # Parámetro n que indica cada cuántos documentos se debe hacer el volcado a disco

PATH_INDEX_FILES='./index_files'
PATH_POSTINGLIST=PATH_INDEX_FILES+'/postings_lists.bin'
PATH_VOCABULARY=PATH_INDEX_FILES+'/vocabulary.bin'
PATH_CHUNKS='./chunks_index'
PATH_PLOTS='./plots'
PATH_MAP_FILESIDS=PATH_INDEX_FILES+'/map_filesids.txt'

TRUE_STATS=False # Se quiere obtener los plots y overheads?
```

- MIN_LEN_TOKEN = Longitud mínima del token (usando en el tokenizador)
- MAX_LEN_TOKEN = Longitud máxima del token (usando en el tokenizador)
- LEN_POSTING_CHUNK = Longitud de los chunks en bytes
- LEN_POSTING = Longitud de los chunks en bytes
- LIMIT_DOCS = Parámetro n que indica cada cuántos documentos se debe hacer el volcado a disco
- PATH_INDEX_FILES = Directorio donde se almacenan los archivos utilizando en el indexado.
- PATH_POSTINGLIST= Path donde almacena las posting list en disco.
- PATH_VOCABULARY= Path donde almacena el vocabulario en disco.
- PATH_CHUNKS= Path donde almacena los chunks en disco.
- PATH_PLOTS= Path donde almacena los gráficos en disco.
- PATH_MAP_FILESIDS= Path donde almacena el mapeo de paths de archivos con docid en disco.
- TRUE_STATS = Flag que indica si se quieren o no gráficas y resultados de overhead

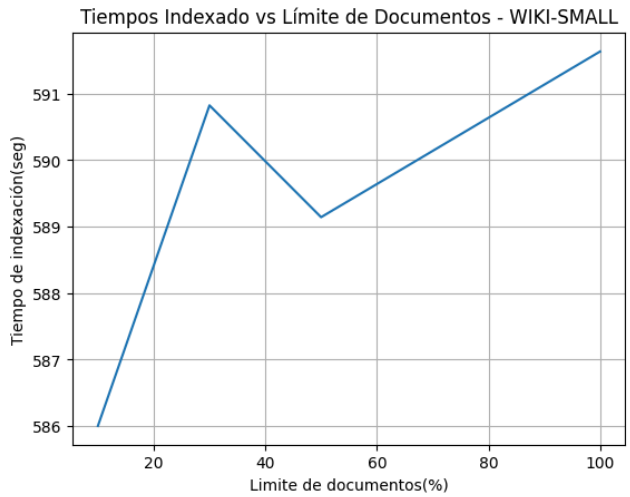
[1] Almacene docID o docID+frecuencia de acuerdo a un parámetro de entrada.
[2] http://dg3rtljvitrle.cloudfront.net/wiki-small.tar.gz(debug), http://dg3rtljvitrle.cloudfront.net/wiki-large.tar.gz (run)

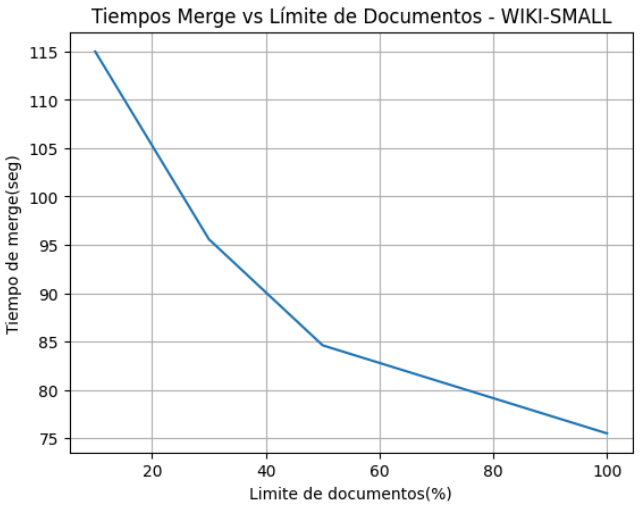
Wiki-small: 6043 docs.

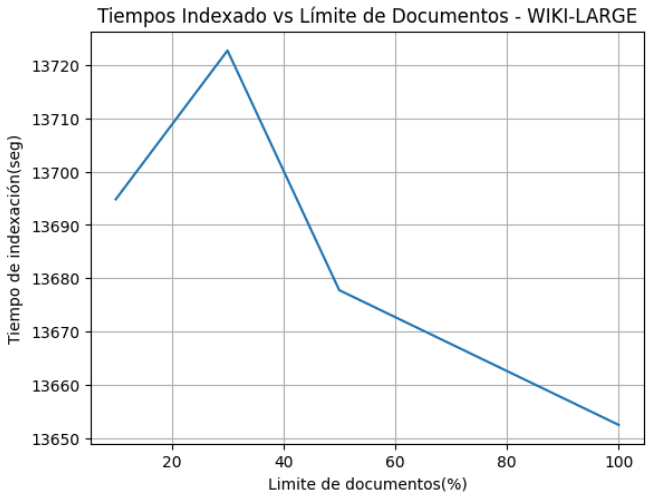
	Wiki-small		
Límite Docs	Cant_docs	Time Index(seg)	Time Merge(seg)
10%	604	586,00	115,01
30%	1812	590,83	95,58
50%	3021	589,14	84,60
100%	6043	591,64	75,48

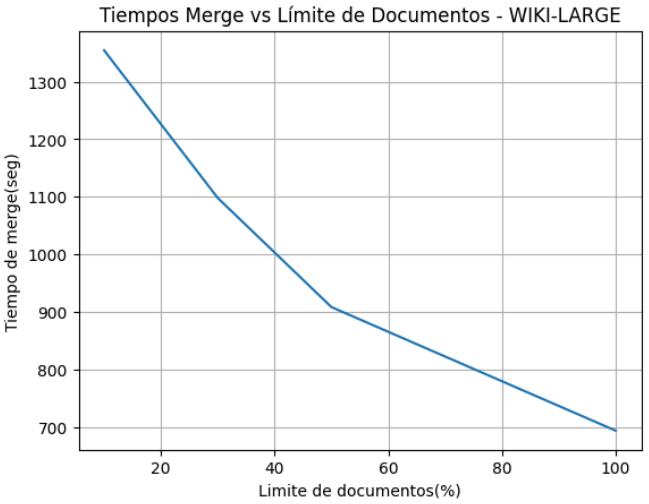
Wiki-large: 121818 docs.

	Wiki-Large		
Límite Docs	Cant_docs	Time Index(seg)	Time Merge(seg)
10%	12181	13.694,78	1.354,43
30%	36545	13.722,71	1.097,81
50%	60909	13.677,75	908,18
100%	121818	13.652,48	693,37







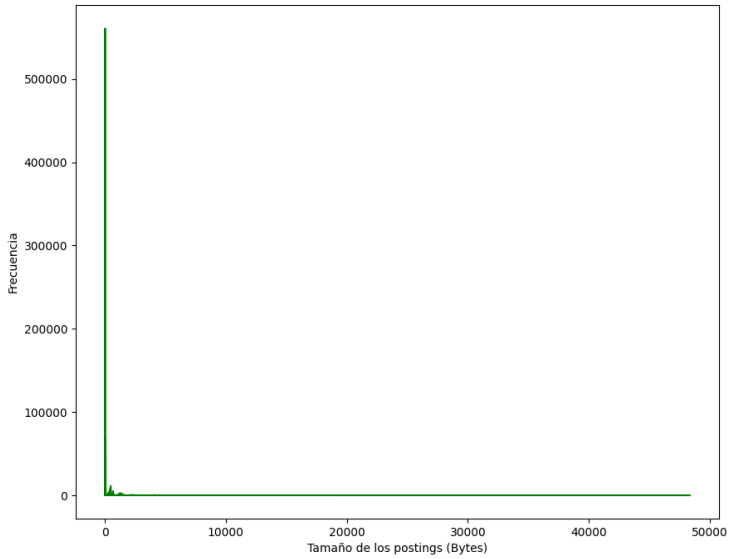


Conclusiones: Para este experimento el tiempo de indexación se mantiene constante independientemente del límite de documentos (n) que se utilice para el volcado parcial a disco. Esto sugiere que el proceso de indexación no se ve significativamente afectado por la frecuencia de los volcados parciales a disco, al menos en este caso. Podemos mencionar cómo a medida que aumenta el límite de documentos (n), es decir, se realizan menos volcados parciales a disco, el tiempo de merge disminuye. Esto se debe a que menos volcados parciales significan menos índices parciales que unificar al final del proceso de indexación. Para finalizar, se debería considerar el trade-off entre elegir un valor “n” específico. Reducir "n" puede disminuir el tiempo de merge, pero también puede aumentar el uso de memoria y la complejidad del código.

Tamaño Colección: 153097389 bytes
Tamaño Índice: 57410559 bytes
Overhead: 0.27 = 27%

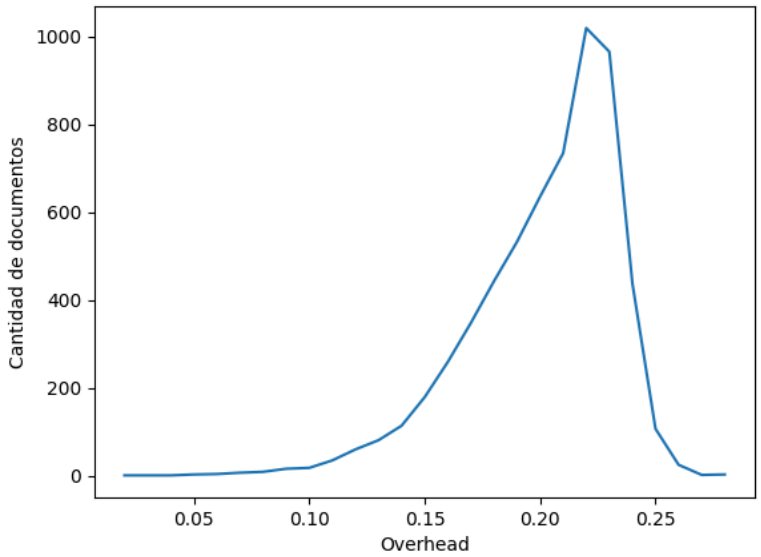
Como se ve en los resultados el overhead del índice en relación con la colección es del 27%, esto nos da información valiosa sobre cuánto espacio de almacenamiento adicional requerimos para el índice.

Distribución Posting Lists(WIKI-SMALL):



Como vemos la distribución de las posting list sigue una ley de potencia (power law), significa que algunas de las listas son muy grandes (con muchos documentos que contienen ese término) mientras que la mayoría son relativamente pequeñas. Esto se alinea con la idea de la ley de Zipf, donde unos pocos términos son muy comunes y aparecen en muchos documentos, mientras que la mayoría de los términos son menos comunes y aparecen en menos documentos.

Distribución de Overhead por documento(WIKI-SMALL):



Se observa que el overhead en la mayoría de documentos ronda entre el 15 y 25%, cada uno impactando en el tamaño final del índice generado.

2) Codifique un script que empleando la estrategia TAAT sobre el índice creado en el ejercicio 1 y operaciones sobre conjuntos permita buscar por dos o tres términos utilizando los operadores AND, OR y NOT.

El script se ejecuta con “python app.py”. Recordar que en el archivo constantes.py estan definidos:

- PATH_INDEX_FILES = Directorio donde se almacenan los archivos utilizando en el indexado.
- PATH_POSTINGLIST= Path donde almacena las posting list en disco.
- PATH_VOCABULARY= Path donde almacena el vocabulario en disco.
- PATH_MAP_FILESIDS= Path donde almacena el mapeo de paths de archivos con docid en disco.
- MIN_LEN_TOKEN = Longitud mínima del token (usando en el tokenizador)
- MAX_LEN_TOKEN = Longitud máxima del token (usando en el tokenizador)
- LEN_POSTING = Longitud de la posting en bytes

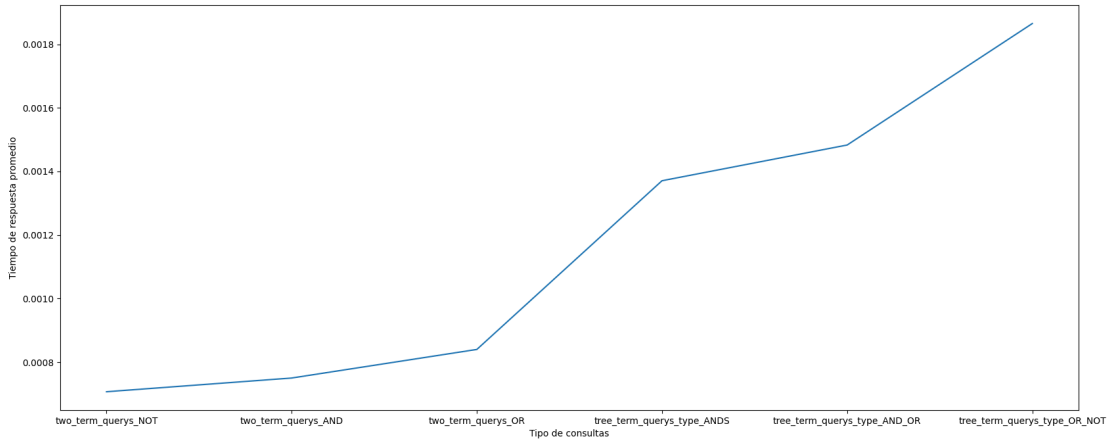
3) **Utilizando el código e índice anteriores ejecute corridas con el siguiente subset de queries[3] (filtre solo los de 2 y 3 términos que estén en el vocabulario de su colección) y mida el tiempo de ejecución en cada caso. Para ello, utilice los siguientes patrones booleanos:**

a) Queries |q| = 2
- t1 AND t2
- t1 OR t2
- t1 NOT t2

b) Queries |q| = 3
- t1 AND t2 AND t3
- (t1 OR t2) NOT t3
- (t1 AND t2) OR t3

¿Puede relacionar los tiempos de ejecución con los tamaños de las listas? (pruebe con el índice en disco o cargándolo completamente en memoria antes). ¿Qué conclusiones se pueden extraer?

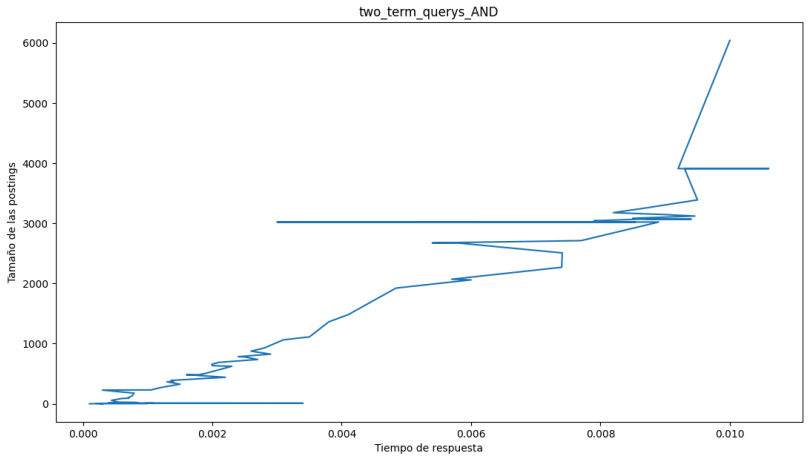
Aclaración: Las pruebas se realizaron solo con el índice en disco del ejercicio 1, definido en "constantes.py". Para obtener los resultados se debe ejecutar "python evaluation.py" y luego "python ploting_results". Se utiliza un txt de stopwords para eliminar palabras vacías.

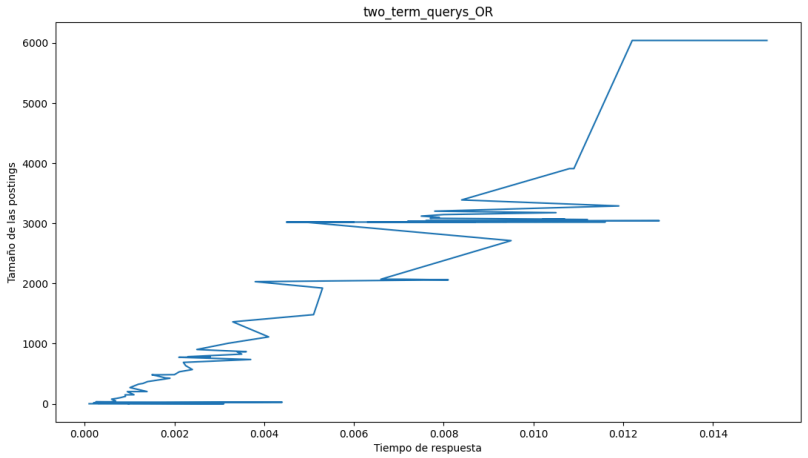


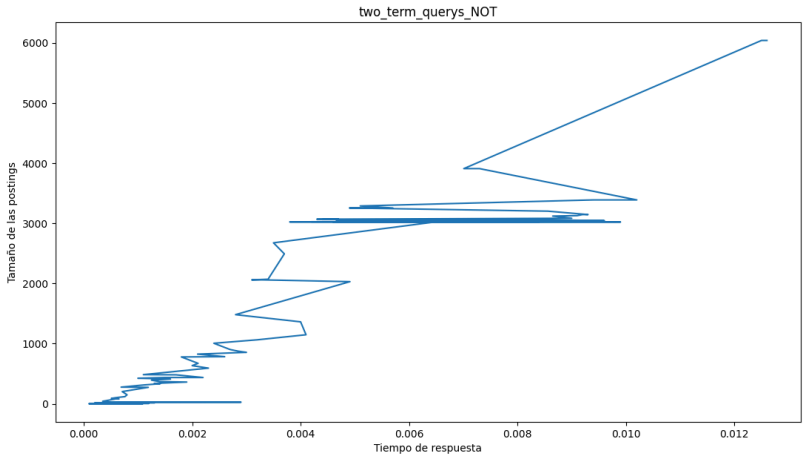
El tiempo de ejecución varía según el tipo de consulta booleana. De forma obvia vemos que las consultas más complejas son consultas con más términos, ya que tienden a tomar más tiempo promedio en ejecutarse por su complejidad computacional al evaluar estas consultas en comparación con consultas más simples, como OR o AND de dos términos.

[3] https://www.labredes.unlu.edu.ar/sites/www.labredes.unlu.edu.ar/files/site/data/ri/EFF-10K-queries.txt

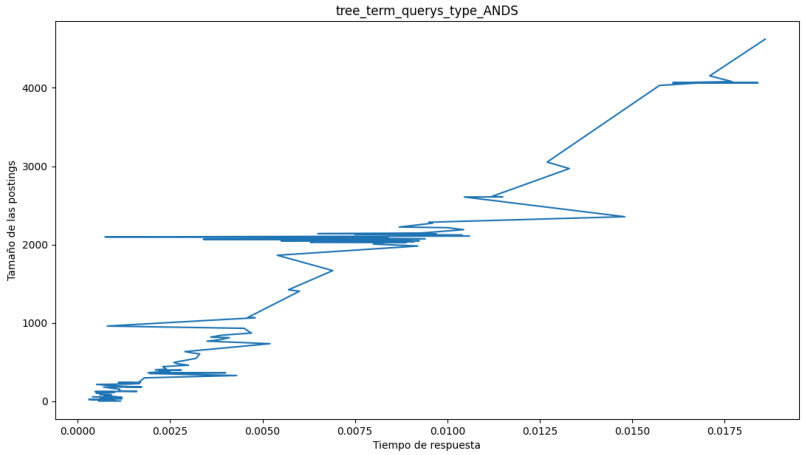
Relacion tiempos de ejecución con tamaños de posting lists(2 Términos):

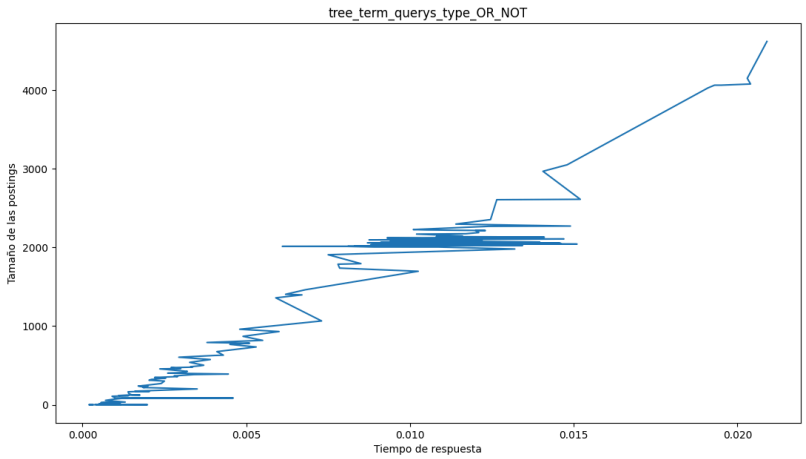


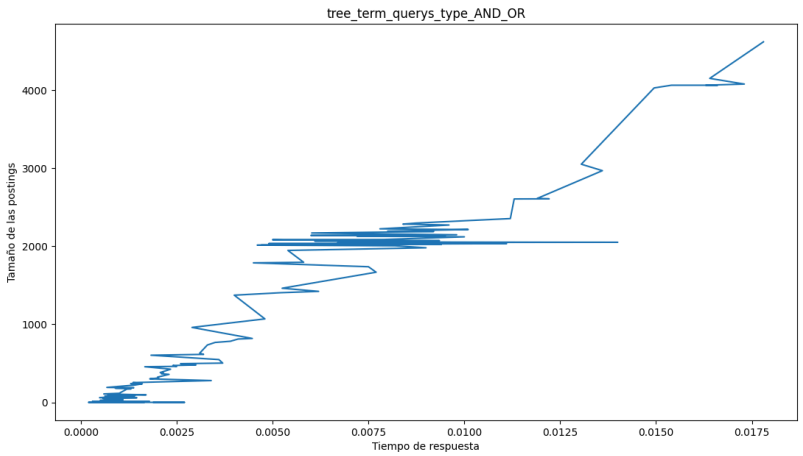




Relacion tiempos de ejecución con tamaños de posting lists(3 Términos):







En general, parece haber una tendencia creciente en los tiempos de respuesta a medida que aumenta el tamaño de las listas de posting. Esto podría ser esperado, ya que con más documentos que

cumplen los términos de la consulta, se requiere más tiempo para evaluar y recuperar los resultados relevantes.
Sin embargo, es interesante notar que hay momentos en los que el tamaño de la lista de posting parece estabilizarse mientras que el tiempo de respuesta sigue creciendo o decreciendo. Esto podría indicar que otros factores, como la complejidad de la consulta o la estructura del índice, también influyen en el tiempo de respuesta.

4) **Codifique un script que empleando la estrategia DAAT sobre el índice creado en el ejercicio 1 resuelva consultas usando el modelo vectorial y retorne los top-k documentos de score mayor.**
El script se ejecuta con “python app.py”. Recordar que en el archivo constantes.py estan definidos, las variables de path, pero se agrega:
   - TOP_K = Tiene el número de documentos relevantes a recuperar.

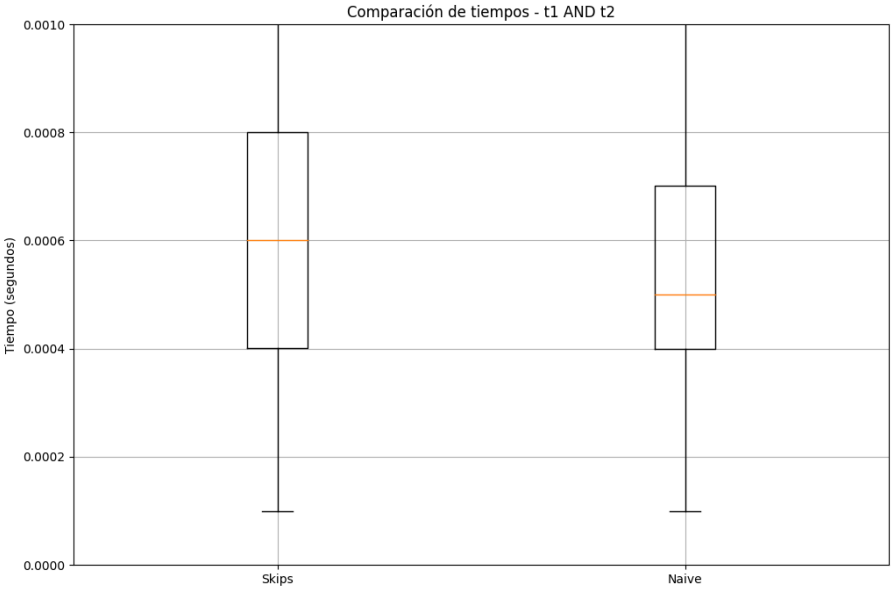
5) **Agregue *skip lists* a su índice del ejercicio 1 y ejecute un conjunto de consultas AND sobre el índice original y luego usando los punteros. Compare los tiempos de ejecución con los del ejercicio 3. Luego, agregue un script que permita recuperar las skips list para un término dado. En este caso la salida deberá ser la lista ordenada por docID.**
El script se ejecuta con “python app.py”. Recordar que en el archivo constantes.py estan definidos, las variables de path, pero se agrega:
   - K_SKIP = Cantidad K de las skiplist.

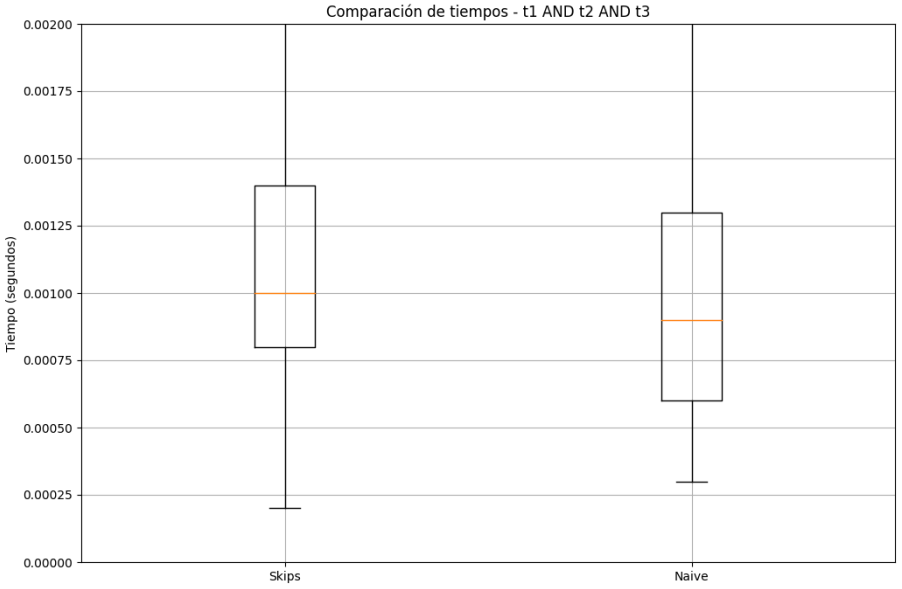
   Para obtener los resultados se debe ejecutar “python evaluation.py” y luego “python ploting_results”. Para el plotting se utilizan los resultados del punto 3 por lo que se recomienda tenerlo ya generado al archivo json.

   En la comparación de tiempos la versión Naive termina superando a mi implementación con Skiplist, por lo que tal vez tenga demasiada complejidad mi versión con Skiplist, aún así muestro los resultados:

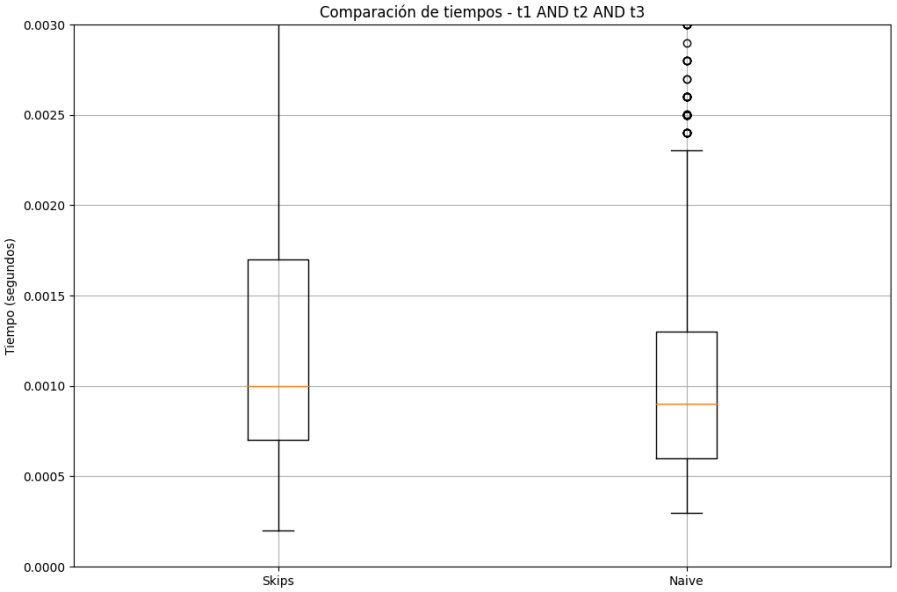
   Se hicieron comparaciones con distintos K, para las Skiplist (en las páginas siguientes se ven los gráficos).

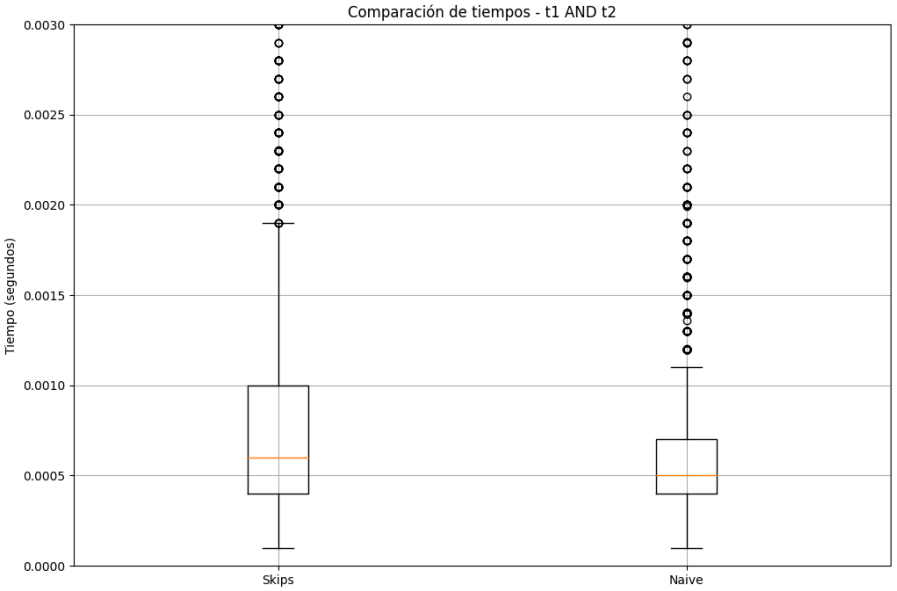
K=2


Comparación de tiempos - t1 AND t2
Tiempo (segundos)
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
Skips
Naive

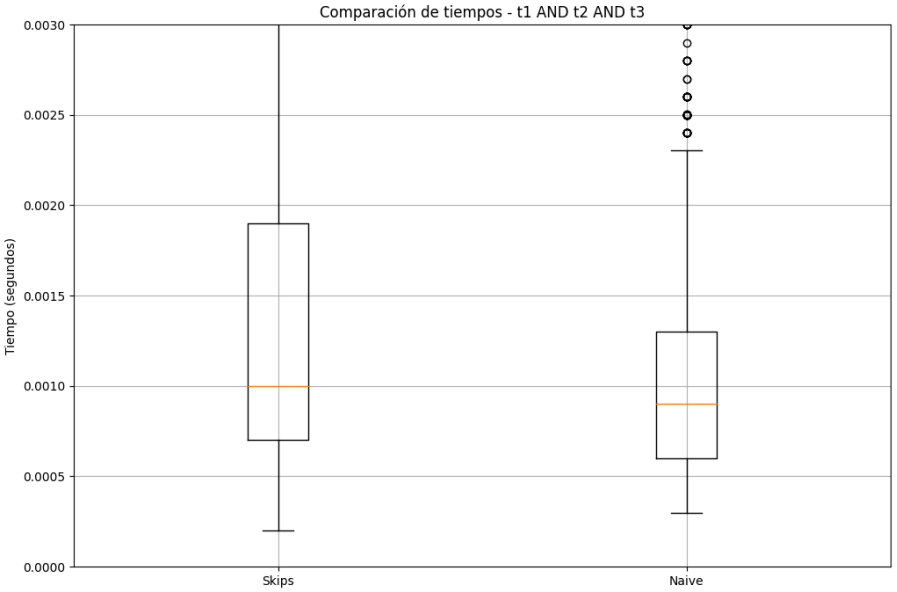

Comparación de tiempos - t1 AND t2 AND t3
Tiempo (segundos)
0.00200
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
Skips
Naive

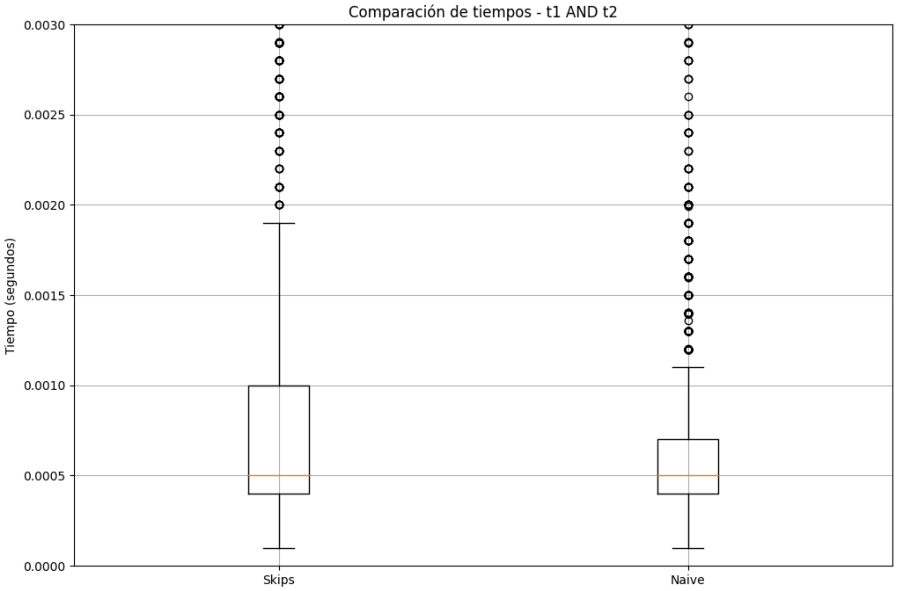
K=20


Comparación de tiempos - t1 AND t2 AND t3
Tiempo (segundos)
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
Skips
Naive


Comparación de tiempos - t1 AND t2
Tiempo (segundos)
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
Skips
Naive

K=50





Como se ve los tiempos son mejores para la versión Naive, aunque se ve que los tiempos medios de ejecución con un K=50 pueden ser similares, hay una mayor variabilidad en los tiempos de ejecución.

6) **Sobre la colección Dump10k escriba un programa que realice una evaluación TAAT y otro usando DAAT. Compare los tiempos de ejecución para un conjunto de queries dados[4]. Separe su análisis por longitud de queries y de posting lists.**

El script se ejecuta con “python app.py”. Recordar que en el archivo constantes.py estan definidos, las variables de path, pero se agregan:

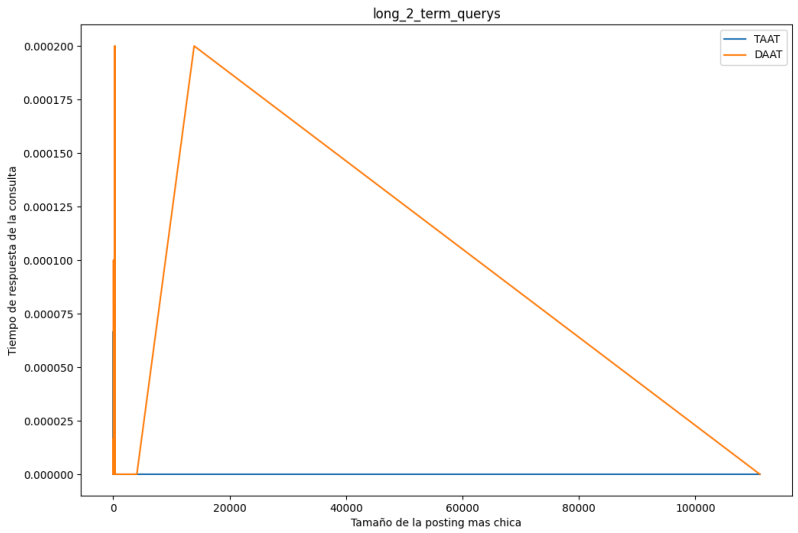
- PATH_DUMP_10K = Path del documento “dump10k.txt”
- PATH_QUERYS_DUMP_10K = Path del documento “queriesDump10K.txt”.
- TYPE_RETRIEVAL = Método de Recuperación en caso de que se haga una consulta manual “TAAT/DAAT”.

Para obtener los resultados se debe ejecutar “python evaluation.py” y luego “python ploting_results”. Se utiliza un txt de stopwords para eliminar palabras vacías.

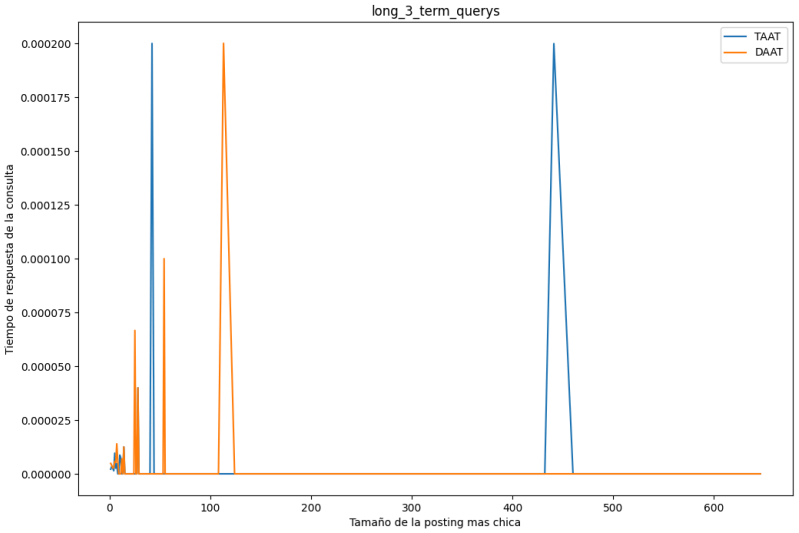
En este caso de estudio se usaron consultas AND. Además para las gráficas de de longitud de posting vs Tiempos de respuesta, se hizo uso de la postinglist con longitud más pequeña.
Resultados tiempos promedio:

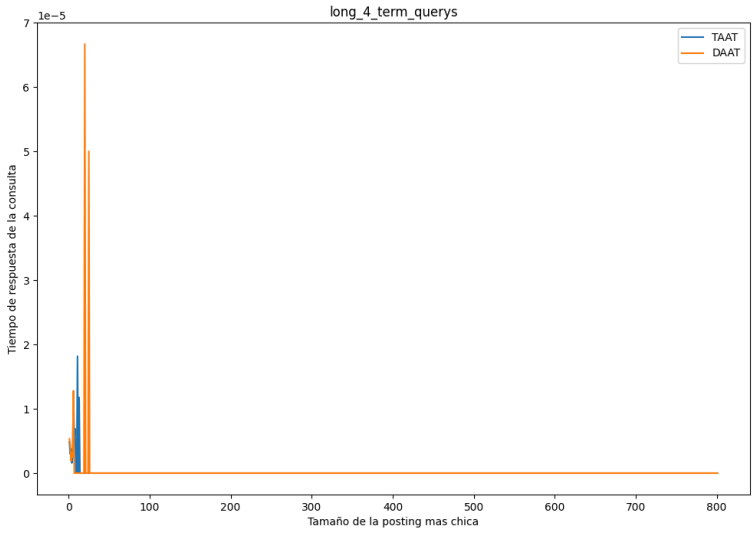
```
long_2_term_querys: TAAT-tiempo_promedio: 3.839969635009766e-06, DAAT-tiempo_promedio: 3.519935607910156e-06
long_3_term_querys: TAAT-tiempo_promedio: 2.960071563720703e-06, DAAT-tiempo_promedio: 4.479789733886719e-06
long_4_term_querys: TAAT-tiempo_promedio: 4.159679412841798e-06, DAAT-tiempo_promedio: 4.3999099731445314e-06
```

Como se ve en el DAAT se logró un tiempo menor que el TAAT con respecto a query de 2 términos. Mientras que el TAAT, tiene una mejora en la velocidad de 3 términos y sigue siendo un poco menor en las querys de 4 términos.



[4] http://www.labredes.unlu.edu.ar/sites/www.labredes.unlu.edu.ar/files/site/data/ri/queriesDump10K.txt.tar.gz





En los tres gráficos, se observa una considerable variación entre ambos métodos de recuperación. Aunque el enfoque TAAT tiende a mantenerse con picos de tiempo más bajos, debido a esta variabilidad, los tiempos promedio difieren solo por milisegundos en este caso de estudio.

7) **Comprima el índice del ejercicio 1 utilizando Variable-Length Codes para los docIDs y Elias-gamma para las frecuencias (almacene docIDs y frecuencias en archivos separados). Calcule tiempos de compresión/descompresión del índice completo y tamaño resultante en cada caso. Realice dos experimentos, con y sin DGaps. Compare los tamaños de los índices resultantes.**

El script se ejecuta con “python app.py”. Recordar que en el archivo “constantes.py” estan definidos:

- PATH_INDEX_FILES= Path del índice a utilizar (se usa el del punto1)
- PATH_POSTINGLIST= Path del archivo binario con las posting (se usa el del punto1)
- PATH_VOCABULARY= Path del archivo binario con el vocabulario (se usa el del punto1)
- LEN_DOCID = Longitud en bytes del docid
- LEN_FREQ = Longitud en bytes de la frecuencia
- LEN_POSTING = Longitud en Bytes de la postinglist para la recuperación del índice
- PATH_COMPRESSION_FILES = Directorio donde se almacenan los bloques comprimidos
- PATH_COMPRESSED_BLOCKIDS = Path del archivo docids comprimido
- PATH_COMPRESSED_BLOCKFREQS = Path del archivo freqs comprimido
- USE_DGAPS= Flag para saber si usar o no Dgaps

Aclaración: Las pruebas se realizaron con el índice de la wiki-small.

Tiempos:

Docids	T-Compression(segs)	T-Descompresión(segs)	Size_Block(bytes)	Size_Comp(bytes)
Variable Length	9,943989	85,947989	16229216	8021500
VL+DGaps	5,853000	80,176599	16229216	5203073

Docids	Tasa de Compresión	Ahorro de Espacio(%)
Variable Length	2,023	50,57
VL+DGaps	3,119	67,94

Frecuencias	T-Compression(segs)	T-Descompresión(segs)	Size_Block(bytes)	Size_Comp(bytes)
Elias-Gamma	6,470011	82,805997	16229216	1436798

Frecuencias	Tasa de Compresión	Ahorro de Espacio(%)
Elias-Gamma	11,295	91,15

Conclusiones:
Para este experimento, tenemos que para Elias Gamma(EG) y Variable Length(VL), los tiempos de compresión son más bajos que sus tiempos de descompresión, aunque EG termina siendo casi 4 segundos más rápido que VL. Luego para la tasa de compresión del bloque con docids la compresión VL es 2 veces más eficiente, mientras que para las frecuencias la compresión EG es 11 veces más eficiente. Sobre el Ahorro del espacio los resultados muestran que VL ahorro un 50,57% para los docids y EG un 91,15% para las frecuencias.

Al utilizar Dgaps para los docids, se obtuvo que el tiempo de compresión disminuye un poco menos de la mitad y para la descompresión se reduce casi 6 segundos comparado a la versión sin Dgaps. Por último, la tasa de compresión resulta ser mayor siendo 3 veces más eficiente y con un ahorro de espacio del 67,94%, una mejora significativa.