**Pruebas teóricas**

**Algoritmo indexación**

|  |  |  |
| --- | --- | --- |
| Caso de Prueba | Pasa | Observaciones |
| Primera palabra del primer documento | Si |  |
| Primera palabra no existente de un documento cualquiera dentro de un lote | Si |  |
| Primera palabra existente de un documento cualquiera dentro de un lote | Si | Obs 1: la llamada a vocabularySlot.put() debería hacerse en la rama en la que no existe el token dentro del vocabulario (antes, dentro del else). Funciona bien, pero en los casos en los que el token ya existe la llamada es redundante y una pérdida de tiempo.  Obs 2: Cuando se carga la lista de posteo se supone que el slot de vocabulario ya contiene el índice del comienzo de la lista en el archivo en disco. |
| Palabra cualquiera del primer documento (primera aparición) | Si |  |
| Palabra cualquiera repetida del primer documento | Si |  |
| Palabra cualquiera de un documento dentro de un lote (primera aparición) | Si |  |
| Palabra repetida de un documento dentro de un lote | Si |  |
| Palabra repetida – primera aparición en lote actual | Si |  |
| Primera aparición en un lote de una palabra ya existente (en lote anterior) | Si |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Observaciones adicionales:**

* Cuando se escriben las listas de posteo en el archivo en disco es cuando se setean los índices de la lista en su correspondiente vocabularySlot, además cuando se escribe cada lote se deben borrar las referencias a las listas de posteo en memoria (por cada lista que se escriba se debe buscar el slot del vocabulario para el correspondiente token y desreferenciar la lista de posteo mediante un método de la forma “deletePostingList()” dentro del slot de vocabulario), además se deberá desreferenciar la lista de tokens modificados y sugerir la pasada del garbage collector.
* Tal como está planteado el ordenamiento de la lista de tokens modificados, los tokens nuevos van a estar al principio de la lista (tienen el índice de inicio en el archivo nulo), y los tokens que ya tengan su lista grabada en disco estarán a continuación ordenados en orden creciente. Por lo que primero se almacenarán los tokens nuevos, y luego se irán intercalando los tokens que no modificados y modificados según el índice de inicio.  
  Ejemplo:  
  Suponiendo que los tokens modificados que ya tenían sus listas de posteo cargadas en disco corresponden a las listas 7, 10 y 15, entonces primero se escribirán todas las listas de posteo correspondientes a los tokens nuevos, luego se copiarán tal como están las listas desde 0 a 6 al archivo nuevo, se escribirá la lista modificada 7, se copiarán todas las listas de la 8 a la 9, se actualiza la 10, se copian de 11 a 14, se actualiza la 15 y por último se copian todas las listas restantes a partir de la 16. Mientras se realiza todo esto, con cada lista que se copia se deberán ir actualizando los índices de inicio en cada slot de vocabulario para el correspondiente término.
* Lo voy a probar así con el archivo de acceso aleatorio, para comprobar el rendimiento (aunque no espero que sea demasiado bueno una vez que se empiecen a indexar muchos documentos), si no tiene buen rendimiento directamente pruebo con la base de datos.
* El ordenamiento dentro de addModifierToken está de más, puedo agregar el modified token directamente en un índice. Puedo buscar con un binary search, ya sea propio o el que provee la api collections e insertar en el índice que corresponde, los demás se desplazan hacia la derecha