75.41 Algoritmos y Programacion I I
Catedra Wachenchauzer
Cuatrimestre II, 2014

Trabajo Practico II:
TWITTER

Ayudante: Martín Buchwald

Alumno: Braian Vicente
Padron: 96542

Alumno: Mauro Gaston Jareño
Padron: 96824

1. **PROBLEMATICAS DE LA CONSIGNA**

Luego de leer detenidamente la consigna, se encontraron las siguiente problemáticas que deberán ser resueltas para poder cumplir con la consigna:

1. La forma de almacenar los twitt's en si mismos (texto) con su identificador unico, y ademas la cantidad de favoritos que recibe.
2. La forma de almacenar, para luego acceder, a cada usuario o hashtag que vayan apareciendo en cada uno de los twitt's.
3. El mecanismo de busqueda por especificacion, ya sea cronologica o popular, de un hashtag o usuario, teniendo en cuenta sus apariciones en cada twitt. Ademas, ajustandose la misma con el modificador de cantidad de resultados.

2. **RESOLUCION DE PROBLEMATICAS**

El criterio que se utilizo para la resolucion de las problemáticas fue la optimizacion en tiempo de ejecucion.

Almacenamiento de Twitt's:

Para el almacenamiento de los twitt's, para representar a cada twitt se utilizo una estructura que agrupa el texto, el ID y cantidad de favoritos que se va modificando o no. Ademas, todos estos se guardan en un vector dinamico. Este permite el acceso a cada estructura de twitt en *O(1)*, cuando se quiere agregar un favorito o acceder al texto para mostrarlo por pantalla. Esto es posible ya que el ID del twitt es el indice mismo del vector.

Se podria haber optado por utilizar una tabla de hash, en el cual como clave se utilizaria el ID del twitt y como dato asosiado la estructura del twitt pero, para este caso, no era necesario utilizar una estructura tan compleja para la resolucion de la problemática planteada.

Almacenamiento Usuarios y Hashtags:

Se opto por usar una tabla de hash donde las claves serian los usuarios y hashtags, y como datos asociado una lista que representan sus apariciones en los twitt's. Lo mas destacable es que el tiempo de acceso a las caracteristicas de cada twitt (cuerpo del twitt, favoritos, ID) es *O(1)*, en comparacion es mucho mas rapido que usar Listas *(O(n))* o un ABB *(O(n log(n)))*. La unica desventaja a tener en cuenta es que conlleva un mayor gasto de memoria en comparacion con una Lista o un ABB, pero para el criterio elegido es despreciable.

En cuanto a almacenar las apariciones en los twitt's se decidio

utilizar listas ya que facilitaba el manejo de estas con los iteradores y porque el insertar en la lista al principio es *O(1)*.

Se podria haber utilizado un vector, el problema es que insertar al principio para mantener el orden cronologico de las apariciones es mucho peor que *O(1)*.

Mecanismo de busqueda:

Debe resolver si la especificacion es por cronologia o por popularidad y acorde a ello llevar a cabo una estrategia de resolucion.

En caso de la primera se obtiene del hash, según sea el hashtag o usuario a buscar, la lista con las apariciones en los twitts. Como en ella la insercion se hace al principio, el resultado es una lista que esta ordenada cronologicamente, de la mas reciente a la mas antigua aparicion. Lo unico que resta es iterar dicha lista K veces (k modificador de cantidad de resultados) mostrando los textos de los twitt's junto con su ID. El orden en tiempo es *O(k)*.

En el caso de la segunda se itera la lista completa y se utiliza una estructura auxiliar, un heap de maximo, que luego de haberse terminado la lista tiene las apariciones en los twitt's ordenadas por favoritos. Luego, se extraen K elementos, siendo el mismo nombrado anteriormente, para mostrarlos. El orden en tiempo, es *O(n log(k))*. Se tarda *O(n)* en iterar la lista, *O(n log(k))* para insertar en el heap y *O(log (k))* extraer los K elementos necesarios.

En lugar de usar un metodo ordenamiento, se utilizo el heap de maximo porque resulta muy conveniente, ya que al insertar la misma estructura mantiene todo ordenado de mayor a menor, y el orden en tiempo es menor considerablemente.

La desventaja seria que se usa una estructura auxiliar a la lista, que en caso de usar un heapsort no seria necesario (teniendo un vector), es decir se consume mas memoria, pero lo que que se busco fue la optimizacion en tiempo de ejecucion y no el consumo de memoria.

3. **FLUJO DEL PROGRAMA PRINCIPAL**