Twitter y una implementación del TT

Introducción

Twitter es una red social basada en la publicación mensajes cortos (*tweets*). Una de las características de esta red es la publicación de los temas más comentados del momento, llamados *trending topics* (*TT*). Los usuarios pueden contribuir a la popularidad de ciertos temas mencionándolos usando *hashtags* que se crean agregando un caracter numeral a una palabra, por ejemplo `#Algoritmos2`.

El objetivo de este trabajo práctico es modelar un sistema que calcule los *TT*'s de una serie de tweets.

Count Min Sketch

¿Cómo hace Twitter para llevar la cuenta de la cantidad de veces que se menciona cada hashtag? Una primera opción sería utilizar un diccionario cuya clave sea el hashtag, y luego como dato la cantidad de ocurrencias. Almacenar lo que potencialmente pueden ser millones de hashtags puede ser requerir mucha infraestructura, o incluso imposible, y muchas veces alcanza con tener una aproximación. Utilizaremos este enfoque para realizar este trabajo práctico.

Para esto se propone usar una idea basada en el funcionamiento de las tablas de hash. Simplemente almacenaremos la cantidad de apariciones de cada hashtag en un arreglo, al que accederemos a través de una función de hashing. Por lo tanto, si `f("Algoritmos2") = 4`, incrementaremos en uno el valor almacenado en la posición 4 de nuestro arreglo. A esta técnica se la llama Count Min Sketch.

Es cierto que aquí si 3 palabras caen en la misma posición, se le van a sumar las apariciones, lo cual puede ser un problema. Supongamos que tenemos **2 contadores**, cada uno con una función de hashing diferente (y que creamos que es poco probable que esas mismas palabras vuelvan a colisionar). Entonces, podríamos decir que la cantidad de apariciones de una palabra es igual al mínimo de los contadores que le corresponden a cada una. Esto puede ser falso, porque se siguen contando otras colisiones, pero si tenemos *k* contadores, y tomamos el mínimo entre ellos, entonces podemos decir que si bien puede haber aparecido menos veces, seguro no más. Sigue siendo una aproximación, pero es mejor.

Consigna

Implementar dos programas en C:

Procesar Tweets:

Este programa se utilizará para contar los trending topics en un conjunto de tweets, leyendo cada línea proveniente de la *entrada estándar* (`stdin`). Como se sabe que la cantidad de mensajes puede ser muy grande, o bien infinitos, para usarlo se requieren dos parámetros enteros, llamados n y k.

El programa deberá imprimir por *salida estándar* (`stdout`) el **histórico** de los k TTs aproximados cada n lineas, ordenados por ocurrencias (en caso que dos TT tengan la misma cantidad de ocurrencias, se deberá mostrar por orden alfabético). Pueden suponer que n y k serán valores tales que se pueda almacenar en memoria esa cantidad de cadenas o estructuras auxiliares.

El tiempo de procesamiento de cada tag debe ser $\mathcal{O}(1)$, mientras que la impresión de los k TT debe realizarse en $\mathcal{O}(n + k \log n)$.

Ejemplo de invocación: `` `~$ cat tweets.txt | ./procesar_tweets 300000 20` ``

Recomendamos crear un TDA CountMinSketch, para lograr que el código quede modularizado. Esto no limita a la creación de otros TDAs que puedan ser útiles, los cuales también recomendamos fuertemente. La cantidad de funciones de hashing a utilizar (y por ende, contadores), así como el largo de éstos serán dejados a criterio del grupo.

Asimismo, se pide que se imprima la frecuencia estimada para poder realizar durante la corrección una comparación con la frecuencia exacta. La salida debe ser como se indica a continuación, con separadores numerados entre cada grupo de *TT*; por ejemplo, para una entrada de 9 líneas, con $n = 4$ y $k = 2$ la salida sería del estilo:

```
```
--- 1
7 pesquisa
2 zombietalk
--- 2
14 niley
3 squarespace
--- 3
1 charlotte
```
```

Procesar Usuarios

Este programa tendrá como objetivo contar la cantidad de hashtags que usa cada usuario, leyendo cada línea del archivo pasado por parámetro. Como se sabe que la cantidad de usuarios es **mucho** menor a la cantidad de *TTs*, y que dicho archivo termina, consideramos que se puede almacenar en memoria todo lo necesario para procesar la entrada.

El programa deberá procesar la entrada y luego deberá escribir por **salida estándar** los usuarios y la cantidad de hashtags que utilizaron en tiempo lineal: $\mathcal{O}(u + t)$ siendo u la cantidad de usuarios encontrados y t la cantidad de hashtags diferentes, ordenados según ocurrencias. Los usuarios que tienen la misma cantidad de hashtags tienen que ser impresos por orden alfabético (tener en cuenta que para ordenar alfabéticamente a los usuarios, los nombres de los mismos no superan más de 15 caracteres).

Ejemplo de invocación: `~$ ./procesar_usuarios tweets.txt`

Ejemplo de salida: El usuario Mile tiene 5 tags, Martín 3, Jorge 3, Javier 5, Nacho 8, Cami 2, Ceci 5, Ezequiel 3, Matías 2, Dato 6, Anita 1, Gian 1. Se debe escribir por salida estándar:

```
1: Anita, Gian
2: Cami, Matias
3: Ezequiel, Jorge, Martín
5: Ceci, Javier, Mile
6: Dato
8: Nacho
```

Sets de Datos

En ambos casos, la entrada debe tener el siguiente formato:

```
Usuario1,tag1,tag2,...
Usuario2,tag3,tag4,...
Usuario1,tag5,tag1,...
```

Donde cada línea corresponde a un tweet realizado por el usuario mencionado, continuado por una los tags en dicho tweet. Por eso, un mismo usuario puede aparecer más de una vez. No hay líneas que no contengan al menos un tag.

Para esto, contarán con un archivo de prueba provisto por el curso. Dicho archivo fue procesado de sus originales aquí. También les brindamos una versión con los primeros cien tweets para usar durante la etapa de desarrollo.

Pueden crear otros archivos de pruebas que a ustedes les sirva para probar el programa.

Atención: tener en cuenta que el archivo provisto por el curso pesa, descomprimido, 460 Mb, y cuenta con más 19 millones de líneas. Se recomienda utilizar el comando `less` de Unix para leerlo, ya que no lo carga en memoria: `less tweets.txt`.

En el sitio de descargas pueden encontrar un set de pruebas que pueden utilizar.

Criterios de aprobación

Los siguientes aspectos son condición necesaria para la aprobación del trabajo práctico.

Estructuras de datos

Es necesario emplear la estructura de datos más apropiada para cada función del programa, en particular **teniendo en cuenta la complejidad temporal**. También es necesario utilizar dicha estructura de la forma más apropiada posible para optimizar de manera correcta el funcionamiento de la clínica.

Se puede (y alentamos) implementar otros TDAs o estructuras que faciliten o mejoren la implementación de este Trabajo Práctico. Les recordamos que un TDA no es un simple `struct` donde se guarda información, sino que debe tener comportamiento. Esto será un punto muy importante en la evaluación del Trabajo Práctico.

Todas las estructuras deben estar implementadas de la forma más genérica posible y correctamente documentadas. En general, pueden emplear sin modificar los tipos implementados en entregas anteriores en la materia. En cualquier caso, si fuera necesario modificar algún TDA, la funcionalidad agregada debe estar implementada de forma completamente abstracta al presente trabajo práctico.

Condiciones de la entrega

El código entregado debe ser claro y legible y ajustarse a las especificaciones de la consigna. Debe compilar sin advertencias y correr sin errores de memoria. La fecha de entrega del TP es el ** 3/12**.

La entrega incluye, obligatoriamente, los siguientes archivos de código:

- Código fuente de `procesar_tweets`.

- Código fuente de `procesar_usuarios`.
- El código de los TDAs programados en la cursada que se requieran.
- Un archivo `deps.mk` con las dependencias del proyecto en formato make. Este deberá contener sólamente una línea por programa que indique qué *objetos* necesita para compilar el ejecutable de cada uno de los archivos, por ejemplo:

```
procesar_tweets: procesar_tweets.o count_min_sketch.o
procesar_usuarios: procesar_usuarios.o hash.o
```

El corrector automático va a interpretar ese archivo de dependencias y va a compilar todos los `.o` especificados a partir de los `.h` y `.c` que deberán enviar, con los siguientes flags de `GCC`:

```
-g -std=c99 -Wall -Wtype-limits -pedantic -Wconversion -Wno-sign-conversion
```

La entrega se realiza únicamente en forma digital a través del sistema de entregas, con todos los archivos mencionados en un único archivo ZIP.