95.12 Algoritmos y Programación II
Trabajo práctico 2

3 de diciembre de 2021

Corrector	Federico Brasburg		
Alumnos	Santiago Somoza	102774	ssomoza@fi.uba.ar
	Facundo Arball	105096	farballo@fi.uba.ar

1. Introducción

Se quiere implementar una nueva red social llamada AlgoGram. De momento, debemos realizar la implementación del sistema de posteos y que esto le llegue a los demás usuarios, y que cada usuario pueda tener un feed que priorice la relación con otros usuarios, poder likear un post, entre otras opciones. Para la realización del trabajo, se utilizaron los TDAs implementados en trabajos anteriores para cumplir con requisitos de complejidad en los comandos del programa. También se crearon estructuras específicas para la implementación.

2. Funciones implementadas

La red social tiene como comandos posibles hacer login y logout con usuarios, publicar posts, ver posts y likear y ver likes de posts. Estos comandos se implementaron de acuerdo a las siguientes restricciones de complejidades:

- Login: debe funcionar en $\mathcal{O}(1)$.
- Logout: debe funcionar en $\mathcal{O}(1)$.
- Publicar Post: debe funcionar en $\mathcal{O}(u \log(p))$, siendo u la cantidad de usuarios y p la cantidad de posts que se hayan creado hasta ese momento.
- Ver próximo post en el feed: debe funcionar en $\mathcal{O}(\log(p))$.
- Likear un post: debe funcionar en $\mathcal{O}(\log(u))$.

- Mostrar likes: debe funcionar en $\mathcal{O}(u)$

Para poder realizar el comando de login en $\mathcal{O}(1)$, se debe poder acceder al usuario que se quiere loguear sin tener que recorrer por todos el resto de los usuarios. La forma de realizar esto es guardando los usuarios en un diccionario. Para esto se utilizó el TDA Hash. La función de Logout fue implementada de forma que solo implica cambiar variables para indicar que ya no hoy un usuario logueado.

Cada usuario tiene un orden diferente de posts que ve. Por esto, al publicar post, la complejidad depende linealmente de la cantidad de usuarios. Por cada usuario, se debe guardar un registro de los posts que son publicados de forma que puedan ser mostrados en el orden correspondiente. Para cumplir con la condición de que la complejidad también dependa de la cantidad de posts de forma logarítmica, se decidió guardarlo en un TDA Heap por cada usuario ya que realiza el ordenamiento de prioridad con esa complejidad. Ver el próximo post en el feed implica desencolar el primer post del Heap perteneciente al usuario logueado y esto es ideal ya que en las restricciones de la red social solo se permite a los usuarios ver por única vez a los posts. Las prioridades se calculan mediante una función que tiene en cuenta la afinidad entre usuarios (dada por la distancia entre sus nombres al ingresar el listado de usuarios).

Los posts tienen un listado de usuarios que lo likearon. Cuando se muestra los likes del post, se los debe mostrar en orden alfabético. Para que se pueda guardar los usuarios que likearon el post con una complejidad logarítmica y que queden en este orden, se utilizó el TDA ABB. Esta estructura permite también recorrerla utilizando el iterador de forma ordenada con complejidad lineal. Para poder acceder al post a likear mediante el Id con complejidad constante, se utilizó un Hash para guardarlos. Esto permite también no perder la referencia cuando son desencolados de los Heaps que tienen los usuarios y poder liberar memoria.

3. Modularización

3.1. Post

Se realizó una estructura post que contiene un Id, el autor, el contenido y un ABB con los usuarios que lo likearon como fue mencionado en la sección anterior. Se le crearon primitivas para poder acceder a información sobre el post sin necesidad de conocer la implementación interna como por ejemplo, imprimir los usuarios que likearon el post.

3.2. Usuarios

Para los usuarios también se creó una estructura. Esta contiene los datos del usuario y el Heap con los posts ordenados según la afinidad. Este modulo utiliza el de post. Se creó una función de comparación que utiliza un numero de usuario asignado de acuerdo al listado de usuarios ingresado al inicio del programa para calcular la prioridad de los posts, teniendo en cuenta el autor. Se creó también una estructura interna del archivo para integrar los posts con la prioridad local de cada usuario. Al igual que con los posts, se crearon primitivas para usar la estructura externamente.

3.3. Flujo

Por último, se creó un modulo encargado de manejar el flujo del programa. Este contiene los comandos que son llamados desde el main. También tiene una estructura que contiene el estado del programa. Esta guarda la información del usuario logueado, el Hash con posts para poder likearlos y ver sus likes y el Hash con usuarios para poder publicar posts en sus feeds y por loguear.

3.4. Main

El programa corre desde un main en el archivo *algogram.c*. Este realiza validaciones de argumentos y lee la entrada de comandos. Para poder realizar comandos en la complejidad correspondiente, se creó un Hash con las funciones a ejecutar que tiene como clave el nombre del comando a ejecutar.

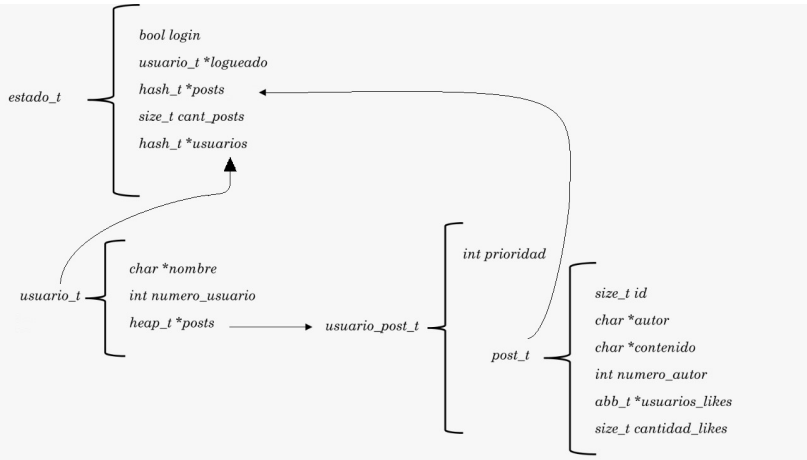

Figura 1: Estructuras y dependencias entre ellas.

4. Conclusión

En el trabajó se puso en práctica el conocimiento adquirido durante la cursada para identificar los TDAs adecuados para conseguir cumplir los con requisitos de complejidades impuestos. También modularizó el programa utilizando estructuras nuevas creadas.