FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2: Informe de diseño

- Materia: Algoritmos y Programación 2
- Alumnos: García Sánchez Julián, Zimbimbakis Francisco Manuel
- Corrector/Profesor: Martín Buchwald
- Fecha de entrega: 03/12

Introducción

Se nos pide implementar una aplicación de consola llamada AlgoGram. Esta funciona como una red social del tipo de sistema de publicaciones con prioridad dependiendo de quien este logeado, y la opción de darle like a la publicación vista.
En lo que sigue de este informe se explicará, de la forma más resumida posible, el diseño y uso de estructuras que se utilizó para desarrollarlo, así también el porque de la decisión de usarlo.

Diseño general y presentación de TDAS

Al pensar el diseño se tomó en cuenta principalmente la complejidad de cada comando, puesto que la eficiencia es, además de una de las consignas, una cualidad importante al desarrollar un software.
Para cada "objeto" (y lo llamo objeto, aunque en C no existan) del programa se desarrolló un encapsulamiento de los datos, para poder generar un nivel de abstracción alto y, además, simplificar y ordenar cada uno en distintos bloques de archivos.
Para empezar, se creó un TDA llamado usuario, el cual contiene su nombre, su número de id y su feed de posts en orden de prioridad. El otro TDA desarrollado es el de las publicaciones (o post para simplificar). Este guarda los datos del creador, del texto del post en sí, y del número de ID que identifica cada post.
Para organizar estos objetos se decidió guardarlos en orden en dos vectores del tipo dinámico (el TDA se llama vector), el ordenamiento es según sus IDs(sean de usuarios o de posts). Estos vectores a su vez se encuentran en una estructura llamada global. Esto nos garantiza tener todos los usuarios y posts ordenados para un fácil acceso, así como también poder liberar la memoria, sin perder referencias, al cerrar el flujo de entrada del programa.
Para el procesamiento de las entradas por teclado del usuario se utiliza un último TDA llamado diccionario. Este chequea que cada comando del usuario sea válido y posteriormente llama a las funciones necesarias para hacer lo que este usuario pidió.
Internamente en cada uno de estos objetos se utilizan mucho los TDAs desarrollados alrededor de este cuatrimestre, elegidos dependiendo de su complejidad algorítmica.

Especificaciones

A partir de lo pedido en la consigna, en global se guardan los vectores ya mencionados anteriormente y un hash de punteros a cada usuario. Por medio de este hash podemos ejecutar un login en tiempo O(1) (despreciamos el procesamiento del nombre pasado por teclado). El logout también es en O(1), ya que un usuario esta logeado por medio de un puntero que se encuentra en global.
Respecto a publicar un post, su implementación es a partir de un heap guardado en cada usuario, ya que las prioridades para visualizarlas dependen de cada uno de estos. Al ser guardadas en un heap, donde encolar es de complejidad log(p), y cómo cada usuario tenga uno, la complejidad final de publicar un post será de O(u log (p)), siendo u la cantidad de usuarios y p la cantidad de publicaciones. Ahora, si analizamos ver un post, internamente es desencolar el de máxima prioridad en el heap del usuario logeado, y esto tiene una complejidad de O(log (p)).
Para la parte de los likes de cada publicación se decidió incluir un tercer vector dinámico, también guardado en la estructura global, que almacena un ABB en la posición correspondiente al id del post al que pertenece. La complejidad de likear un post es O(log u), ya que internamente es guardar en un ABB el nombre del usuario que likeó. En caso de que un usuario intente darle like de nuevo a la misma publicación, el ABB se encarga de ver que está repetido el usuario y lo ignora.
Se mencionó mucho sobre los vectores dinámicos, pero se habló muy poco de su complejidad. Si bien buscar en un vector es O(n), como la posición de cada usuario, publicación y abb coinciden (apropósito) con sus respectivos id, al saber su posición se accede a cada una con complejidad O(1).
Por último, se desarrolló un hash que contiene los comandos que se pueden utilizar en nuestra red social y un puntero a función asociado con cada funcionalidad. De esta forma se procesa en O(1) todas las entradas por teclado.

Quiero destacar que la estructura global no es un TDA, por lo que dentro de ella se pueden imprimir mensajes por pantalla. La estructura post con prioridad tampoco lo es, pero al estar relacionada con el TDA usuario se deja la estructura expuesta en el header de usuarios.

Correcciones:

Dadas las correcciones pedidas, movimos la estructura de prioridad al TDA usuario, dado que las prioridades de salida en cada feed son un problema interno de cada usuario, y no es importancia del desarrollador de global.
También movimos el abb de likes a cada publicación, dado que las publicaciones deben tener acceso a sus propios likes, y le sacamos trabajo a global.
Por último, los errores de las primitivas de global ya no se imprimen por pantalla directamente, sino q se devuelven como un enum y en procesamiento se analiza y decide que imprimir (o no). La forma en q se leen los usuarios se puede cambiar desde afuera a través de una función que recibe como parámetro global_crear.