TP2: Algogram

Grupo 32:

Pedro Gallino, 107587

Aquiles Abuin, 107742

Ayudante:

Federico Barrios

PLANIFICACIÓN:

Antes de comenzar a codear Algogram, nos planteamos qué forma debía tener el programa en su totalidad. Inicialmente creímos cómodo implementar cada comando como una función por separado.

Luego de plasmar a medias esta idea prematura, coincidimos en que lo más conveniente sería implementar Algogram como un TDA, ya que nos pareció positivo tener un conjunto de primitivas con las cuales dirigir el programa. Las mismas al estar “emparejadas” con los comandos disponibles a ingresar por consola, muestran un código más limpio y organizado. Además, el TDA cumple un rol de “contenedor” para las estructuras extras que implementamos. Por otra parte, nos permite llevar el “estado” actual del programa, ya sea la cantidad de posteos hechos o el usuario logueado actualmente.

Luego implementamos diversas estructuras, ninguna en forma de TDA ya que consideramos que todas sólo tienen sentido dentro de la implementación interna del Algogram, y no tienen uso fuera de él.

COMANDOS:

LOGIN:

Un usuario para acceder ingresa su Nick-name por consola. Al ser necesario buscarlo entre los usuarios registrados de forma instantánea, decidimos utilizar el TDA Hash para almacenar los Nick-names de los mismos y la información correspondiente. El hash, nos permite acceder al usuario en O(1), tal como es requerido. Dentro de cada elemento del hash guardamos una estructura interna que tiene una referencia al feed de posts propio del usuario, como también a su nombre y a su ID (el número de línea en la que el usuario fue leído). Como fue dicho anteriormente, el TDA Algogram nos permite llevar el “estado” actual del programa. En este caso, simplemente se toman los datos del usuario y se actualiza el usuario actual.

LOGOUT:

Para este comando, simplemente actualizamos el estado del programa, quitando al usuario activo. Al no ser más que eso, es instantáneo, consiguiendo la complejidad óptima. O(1).

PUBLICAR_POST:

Teniendo en mente la afinidad entre usuarios y la prioridad entre posteos en los feeds de cada usuario, decidimos implementar los feeds mediante el TDA Cola de prioridad (heap). Cada heap almacena los posteos publicados hasta el momento comparando la afinidad del autor con los demás usuarios.

De esta forma, para publicar un post se debe encolar en el feed de cada usuario.

Teniendo en cuenta que la complejidad de encolado es O(log(p)) (siendo “p” la cantidad de posteos almacenados) y que debe hacerse para cada usuario

registrado, la complejidad total del comando es O(ulog(p)) (con “u” como la cantidad de usuarios).

VER_PRÓXIMO_POST:

Como se adelantó en el comando “publicar_post”, elegimos representar el feed de cada usuario con un heap que almacena todos los posts. La cola de prioridad nos permite tener en su “tope” el siguiente post que debemos mostrar al usuario.

El post situado en el tope se desencola del heap, cumpliendo con la complejidad asignada. O(log(p)) (siendo “p” la cantidad de posteos almacenados).

LIKEAR_POST:

Para poder acceder a los posteos rápidamente, y actualizar la cantidad de likes del mismo, fue necesario implementar un nuevo hash (Almacenando en él los posteos). Acceder a los elementos del mismo es O(1).

También se podría haber utilizado una lista de punteros a la que acceder por índices, pero no teníamos una lista que se redimensione ya hecha, y la complejidad temporal en ese caso hubiese sido la misma.

Cada post es almacenado en un struct donde guardamos una referencia al usuario que lo creó, su contenido, su ID y un listado con los usuarios que lo likearon.

Para mantener ordenados alfabéticamente los usuarios que likearon los posteos, elegimos que dicho listado fuese representado por el TDA ABB.

El ABB nos permite mantener una complejidad de O(log(u)) cada vez que encolamos un nuevo usuario, manteniendo la lista ordenada.

MOSTRAR_LIKES:

Al estar el listado de likes implementado mediante un ABB, necesariamente mostrarlos al usuario es O(u), cumpliendo con la complejidad estimada.

Simplemente se recorre el ABB in-order imprimiendo los usuarios que likearon dicho post.