Algogram

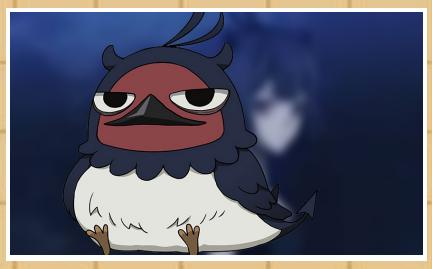
Integrantes

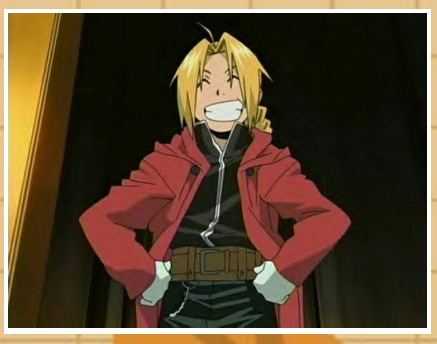
Facundo Aguirre Argerich

Francisco Juarez

Ayudante corrector

Jasmina Sella Faena





Estructura de Algogram

Debido al comportamiento que necesitamos recrear decidimos implementar Algogram con lógica segmentada.

TDA Usuarios

Decidimos crear este TDA para poder recrear el comportamiento que un usuario puede tener. Este TDA nos simplifica mucho las interacciones del Usuario y nos ayuda a ordenar la lógica detrás de este, a la vez de poder usar sus primitivas para tener siempre a mano la información necesaria.

Este TDA implica poder almacenar su información como su nombre, la posición de entrada según el archivo pasado (para calcular su afinidad) y su feed, el cual implementamos con un heap. Esto porque nos deja emular las prioridades que tienen los post de otros usuarios en el feed de cada uno, respetando las afinidades que tienen entre ellos. Todo esto cumpliendo la complejidad pautada en la consigna. El usuario entonces podría ver y avanzar su feed, además de que se le puedan agregar nuevos elementos.

Posts

También planteamos la estructura de los post, los cuales son creados por el usuario con un contenido que éste escribe. Estos dentro tienen, además de su ID, contenido y autor, un ABB que representa sus likes. Decidimos usar un ABB ya que necesitamos que se ordenen alfabéticamente, además se nos hace fácil poder ver la cantidad de likes, es O(1). Este ABB se comporta como un set con prioridad, debido a esta razón, al guardar dentro de este el dato que ponemos siempre es NULL. Nos preguntamos si sería necesario implementar un TDA con este comportamiento para el TP pero al hablar con varios profesores nos dimos cuenta que no hace falta ya que es un uso del ABB.

Además el ABB nos permite cumplir las complejidades pedidas y de una forma bastante cómoda.

TDA red_social

El TDA “red_social” emula el comportamiento de una red social, en este manejaremos las estructuras principales de los datos de los usuarios, sus feeds, posts, y sus variaciones en el flujo del programa acorde a las acciones que se realicen. Decidimos realizar este TDA ya que nos permite tener un manejo global de las estructuras que van cambiando a la vez que el usuario utiliza la red, a la vez de tener una manera cómoda de acceder a este comportamiento. También nos ayuda en la separación de la lógica del programa con el parseo de la entrada estándar.
Este TDA tiene dentro de su estructura un Hash de Posts (con sus ID como clave), este nos sirve para acceder en O(1) a estos y poder cambiarlos (o destruirlos) rápidamente, acá se van agregando los posts a medida que los usuarios publiquen.
También tiene un Hash de Usuarios, donde, al crear este TDA con un path a un archivo, se cargan todos los usuarios a este Hash, con sus nombres como clave y un TDA Usuario como valor. Esto nos sirve para las mismas razones que el Hash de Posts, osea poder hacer el login, logout y demás en O(1). En este caso también pensamos en implementar un TDA ya que este Hash nunca va a ser cambiado y existen maneras de implementar Hashes que tienen un factor de carga ideal para que sea más rápido. Esta idea también la descartamos ya que no hay un cambio significativo.
Otros atributos que tiene el struct de este TDA es el último número de ID de post creado, para poder nombrar los nuevos posts que se creen con sus respectivos ID 's desde 0 a infinito y más allá. El último atributo de su struct es el usuario logueado, este lo usamos para poder chequear si lanzar un error en las operaciones que necesitan que haya, o no, un usuario logueado.
Dentro de este TDA tenemos las funciones que corresponden a los funcionamientos de la red social. Estas se manejan acorde a la entrada del usuario y en función de ello utilizaremos primitivas del TDA Usuario para poder actualizar el estado de los feeds de cada usuario, mostrar los likes pertinentes y realizar todas las funcionalidades correspondientes.

Manejo de la entrada estándar

Para esta parte implementamos un archivo llamado “parseo”, este se encarga del parseo de la entrada estándar. Detecta lo que escribe el usuario y ejecuta su función correspondiente, basándose en las primitivas de red_social.h. La idea de este archivo es la separación entre el parseo de la entrada de la lógica de algogram. Implementando de esta manera el TP, se puede cambiar el parseo de la entrada o la lógica de la red social con un mínimo esfuerzo de parte del programador.

Reflexiones

A la hora de estructurar el TP tuvimos muchos problemas ya que no terminabamos de entender el concepto de TDA. Estuvimos varios días deliberando entre los dos cuál era la mejor manera de implementar lo que buscábamos, pudiendo separar la lógica y organizar el código. Lo que nos generaba muchas dudas era que el TDA necesitaba ser genérico, esto nos trabó al punto de tener que estar días charlando con diferentes profesores para poder entender exactamente que significaba esto. Al final llegamos a la conclusión de que los TDAs son genéricos hasta cierto punto, no podemos realizar todo de manera genérica ya que se complica excesivamente o hasta se hace imposible.
Debido a esto decidimos implementar los TDAs de la forma en la que lo hicimos, intentando que sean genéricos pero esto dentro de lo posible.