INFORME

Integrantes:
José Ignacio Castro Martínez
Mario Rafael Janampa Salazar
Corrector:
Federico Brasburg

FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Índice

Introducción

La nueva red, Algogram surge de la necesidad de implementar un modelo de red tal como se solicita en el trabajo practico #2 de la asignatura Algoritmos y programación II en la Universidad de Buenos Aires, cuya consigna puede encontrarse en el link: https://algoritmos-rw.github.io/algo2/tps/2022_1/tp2/#introducci%C3%B3n. El trabajo practico propone el desarrollo de una red social con seis funcionalidades básicas: logear un usuario, hacer el logout de un usuario, ver un post perteneciente al feed, darle like a un post y a su vez tener la capacidad de ver los likes dados a un post cualquiera. Debido a la alta probabilidad de que se encuentre en uso por muchos usuarios, los algoritmos que se ejecutan por comando deben ser los más óptimos y así, habiendo sido establecidos las complejidades temporales pertinentes a cada uno se desarrolló el Algogram

TDAS utilizados:

Para el desarrollo del Algogram se utilizan TDAS implementados anteriormente, como también, una serie de nuevos TDAS, estos serán: el Algogram, el user y el post, por último los TDAS anteriormente implementados seleccionados serán: la cola de prioridad, el árbol de búsqueda binaria, el diccionario o hash y el vector

Explicación de los TDAS creados:

Vector:

Se implementa un vector con modificaciones, es una generalización del anterior vector creado y consiste en un vector de punteros genéricos que permite almacenar cualquier tipo de dato y una función de destrucción que permite destruir cualquier dato interno del TDA si así se desea

Post:

Este nuevo TDA se implementa por la necesidad de almacenar un mensaje, un creador y también de contener los likes dados a un determinado post, en particular para almacenar la información relacionada a los likes se utiliza un árbol de búsqueda binaria que almacene los nombres de los usuarios a quienes les ha gustado el post, permitiendo así, optimizar la velocidad de agregar un like y facilitar la vista ordenada de los nombres a quienes ha gustado el post.

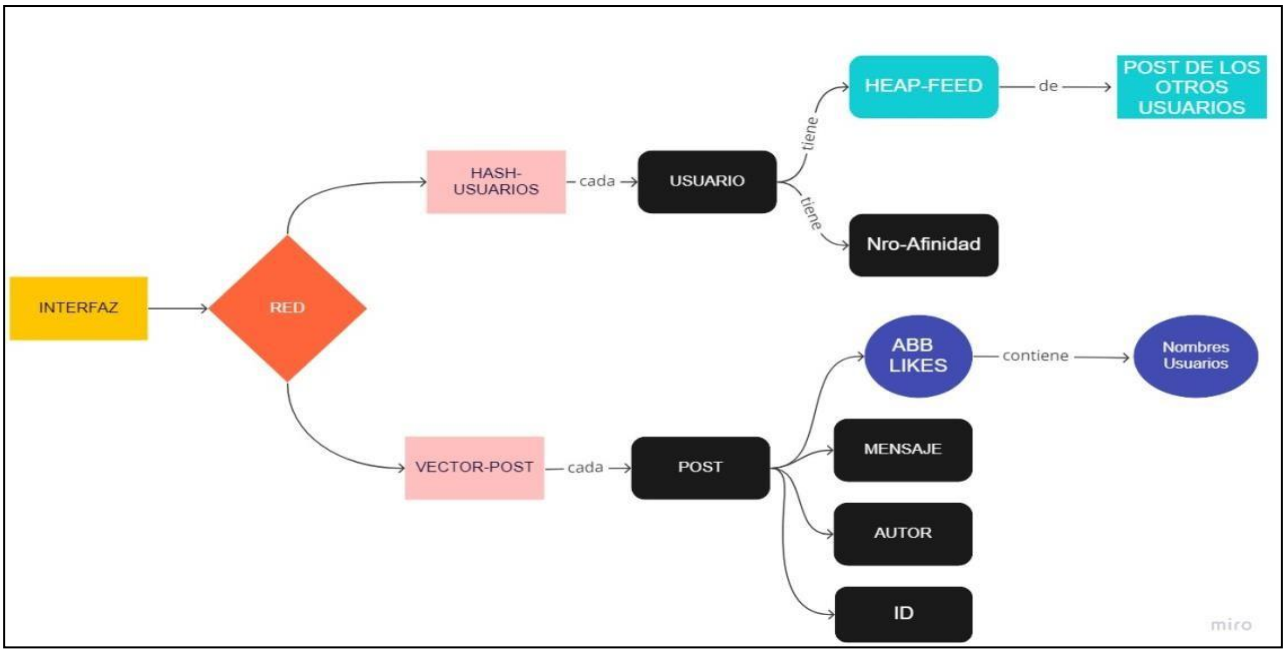
User:

Surge de la necesidad de tener un feed personalizado que muestre las publicaciones dependiendo de los intereses de cada usuario, cada user tendrá para sí mismo: un nombre, un feed y una constante que permitirá calcular su afinidad con otro user. El feed será una cola de prioridad que almacene una referencia a la publicación ordenado de más afín a menos afín dependiendo del usuario autor de la publicación.

TDA Algogram

Este TDA engloba a los anteriores, consiste en un hash que contiene a los usuarios, teniendo por clave el nombre y por dato una instancia del TDA user, esto permite que la búsqueda de un usuario sea realizada en O(1), un vector que sirve como almacén a las publicaciones hechas por cada usuario, como cada post debe tener un código identificador, ocurre que, este va de 0 a n, donde n es la cantidad menos uno de publicaciones, el hecho de que este ID coincida con la posición en el vector permite encontrar el post dentro del almacén en O(1), por último se tiene una referencia al usuario activo en el momento

Diagrama



TDAs y sus correspondientes primitivas:

User:

a.) Crear usuario: es una función que permite crear un usuario, asignándole un nombre y una constante de afinidad que se obtiene por parámetro y una cola de prioridad que servirá de feed. Dicha funcionalidad tiene un complejidad temporal O(1)

b.) Nombre usuario: función que permite saber cuál es el nombre de un usuario determinado. Su complejidad temporal es O(1)

c.) Insertar un post: almacena un post dentro del feed del usuario, esta funcionalidad depende del TDA heap, en donde hacer una inserción tiene complejidad temporal de O(log p) donde p es la cantidad de post dentro del feed

d.) Afinidad usuario: devuelve la constante de afinidad del usuario, su complejidad temporal es O(1)

e.) Calcular afinidad entre usuarios: Esta es una funcionalidad que toma dos usuarios y calcula la afinidad entre ellos, dada por su posición en el archivo, esta función tiene complejidad temporal O(1)

f.) User feed esta vacio: Función que verifica que existan post dentro del feed, esto es verificar la cantidad de elementos dentro de la cola de prioridad, función que se realiza en O(1)

g.) Ver feed: Función que desencola la información proveniente del feed del usuario si esta tiene dentro de si algún posteo, esta función tiene un complejidad temporal O(log p) donde p es la cantidad de post que contenga

h.) Destruir usuario: Destruye el usuario, esta función tiene una complejidad temporal O(log p) donde p es los publicaciones, esto debido al heap del feed

Post:

a.) Post crear: función que crea un post, este método tiene complejidad temporal O(1)

b.) Post id: una función que devuelve el id de un post, esta función tiene complejidad O(1)

c.) Post autor: función que devuelve la referencia contenida al autor del post, su complejidad temporal es O(1)

d.) Post dar like: esta función agrega el nombre de un usuario al árbol de búsqueda binaria que contiene los likes dentro del post, tiene una complejidad temporal O(log u) donde u es la cantidad de usuarios

e.) Post cantidad de likes: es una función que devuelve la cantidad de likes dentro de un post, dicha función tiene una complejidad de O(1) y se basa en ver la cantidad de elementos dentro del árbol denominado “likes”

f.) Post ver likes: función que itera in order el arbol de búsqueda binaria de likes mostrando por pantalla los usuarios quienes han dado like al post, su complejidad temporal es O(u) donde u es la cantidad de usuarios quienes han dado like

g.) Post mensaje: devuelve el mensaje contenido en el post, su complejidad es O(1)

h.) Post printear: función que muestra al usuario el contenido de un post, su complejidad temporal es O(1)

I.) Post destruir: función que destruye un post, dicha destrucción tiene una complejidad temporal O(u) donde u es la cantidad de usuarios, esto dado por el árbol binario de likes interno en el post

Algogram:

a.) Algogram crear: crea la red que va a ser utilizada y a su vez crea para si los TDAs que contiene en su interior, esta función es O(1)

b.) Algogram agregar usuario: función que agrega un usuario dentro del hash usuarios, la complejidad temporal depende del agregar clave al hash lo cual es O(1)

c.) Algogram user existe: se encarga de revisar si el usuario existe o no, lo cual depende de hash pertenece, esto tiene complejidad temporal O(1)

d.) Algogram hay user activo: verifica que haya un usuario activo en la red, su complejidad temporal es de O(1)

e.) Algogram nombre del actual: esta funcionalidad hace uso de user nombre, lo cual hace que sea O(1) en su complejidad temporal

f.) Algogram logear user: verifica que no haya un usuario activo y busca el usuario dentro del hash, posteriormente coloca la referencia al usuario dentro del actual en la red. Consiste en buscar en un hash si pertenece o no la clave, y obtener su dato, lo cuales son todos O(1)

g.) Algogram logout: quita la referencia al usuario activo, lo cual es O(1)

h.) Algogram publicar: crea un post y empieza a iterar el hash por todos los usuarios de este, luego hace uso de la primitiva del TDA usuario insertar post en feed, como esta funcion debe pasar por todos los usuarios y debe hacer una inserción dentro de todos los feeds esto será: O(u) donde u es el número de usuarios y por cada usuario O(log p) donde p es la cantidad de publicaciones en el feed, terminando así con una complejidad temporal de O(u log p)

I.) Algogram dar like: verifica que exista el id del post y da like al mismo, esta funcionalidad depende de la primitiva del TDA post que es O(log u) donde u es la cantidad de usuarios, y las verificaciones previas que hace para ver si el post existe o no que son O(1) resultando así en una complejidad temporal de O(log u)

j.) Algogram ver likes: llama a la función del TDA post, post ver likes, la búsqueda del post se realiza en O(1) y ver los likes se había establecido en O(u) donde u es la cantidad de usuarios que han dado like

k.) Algogram ver feed: llama a la función del TDA usuario, user ver feed, definida como O(log p), el post devuelto es mostrado por pantalla con la funcion post printear que realiza su tarea en O(1)

l.) Algogram destruir: destruye la red creada, llama a hash destruir y a vector destruir, con las funciones de destrucción de datos respectivas a cada caso, su complejidad temporal es O(u + p) ya que debe destruir todos los usuarios y todas las publicaciones

Interfaz

También se construye una interfaz que se encargue de recibir y orientar las ordenes que recibe del usuario, como también que indique los casos de error donde el usuario realizo una acción indebida, las funcionalidades que posee la interfaz son:

Lectura de usuarios: función que lee el archivo de usuarios y se encarga de cargarlos dentro de la red, es previa al uso de la red, y es de complejidad temporal O(n) donde la n viene dada por la cantidad de usuarios

Comparar instrucciones: es una función que compara la instrucción recibida con las instrucciones posibles, y devuelve el código relacionada a las mismas

Verificar: es una función que verifica que se cumplan determinados errores, estos serían: en el caso de lograr que haya un usuario activo, en de ver un like, que este no exista o que no haya un usuario logeado y por último que que no haya un usuario activo al momento de pedir publicar un mensaje

Ejecutar: toma el código de la instrucción, una línea que paso el usuario y la red y se encarga de ejecutar según sea el código de la orden, la función de la red asociada a lo que pidió el usuario, muestra por pantalla el resultado de ejecutar la orden

Lectura de instrucciones: lee el conjunto de instrucciones y va llamando las funciones anteriormente descritas.