Trabajo Práctico 2
AlgoGram

Enunciado
https://algoritmos-rw.github.io/algo2/tps/2022_2/tp2/

Fecha de publicación
4 de Noviembre

Fecha de entrega
18 de Noviembre

Ayudante
Eliana Harriet

Alumnos
Facundo Xu (108295) || Dalmiro Vilaplana (109875)

Introducción

▸ Para este trabajo práctico se utilizaron varios **TDAs** vistos en clase tales como **Hash, ABB, Heap**, etc. Así mismo también se han creado otros TDAs propios del trabajo práctico que son vitales para el funcionamiento principal del programa.

▸ La dificultad que tuvimos ante la solución de este trabajo práctico fue principalmente cuáles TDAs utilizar y cuáles funcionalidades implementar para que todas las acciones realizadas cumplan con la **complejidad** requerida.

▸ En un principio pensamos en utilizar estructuras para **almacenar** información, pero posteriormente llegamos a la conclusión de que podemos aprovechar estas estructuras dándoles comportamiento con el uso apropiado de **primitivas** que nos permiten facilitar ciertas acciones tales como **ver el nombre de un usuario** o **ver el ID de una publicación**. Así mismo convirtiéndolos en TDAs como **Usuario** y **Post**.

Login

▸ Al principio pensamos que sería buena idea almacenar al usuario loggeado en un TDA tal como **Pila** o **Cola**. Pero no tendría mucho sentido utilizar un TDA específico **solo para guardar un solo individuo constantemente**.

▸ Concluimos en utilizar una variable que almacene el nombre del **usuario loggeado**, de esta forma podemos acceder rápidamente al nombre del usuario cumpliendo con una complejidad de **O(1)**.

▸ A su vez, utilizamos el **TDA Diccionario (Hash)** para almacenar todos los usuarios que **están registrados** en esta red social. Como el diccionario tiene una naturaleza de complejidad **O(1)**, podemos acceder rápidamente en este y verificar si un usuario es **válido o no**.

Logout

▸ Como mencionamos previamente, utilizamos una variable para almacenar el nombre de un usuario loggeado. De esta forma podemos saber si **hay alguien loggeado, o no.**

▸ Una vez que el programa detecte que efectivamente **hay un usuario loggeado**, se le asignará un nuevo valor indicando que **no hay nadie loggeado**. Como estamos constantemente asignando un nuevo valor a la variable, esto cumple con la complejidad de **O(1)**.

Publicar

▸ Decidimos utilizar otro **TDA Diccionario (Hash)** que almacene las **publicaciones** realizadas por los usuarios, con sus respectivos **ID** que funcionan como identificador del post realizado. De esta forma podemos acceder rápidamente en **O(1)** utilizando el ID para saber si un post existe o no. En el caso de que el usuario loggeado sea válido, la publicación se realizará con éxito.

▸ Una vez realizada la publicación, todos los demás usuarios **podrán ver** dicha publicación en su **historial**. Dicho historial está representado como un **feed** de publicaciones que, internamente, es un **TDA Cola Prioridad (Heap)**. Esta decisión fue tomada debido que el **orden de las publicaciones** debe cumplir con la característica de **afinidad**. Mientras más afinidad tenga el usuario con la persona que **realizó el post**, más probabilidad de que dicho post sea el primero en ser visualizado. De esta forma, y con una **función de comparación** que cumple con esta característica, la complejidad es de **O(ulog(p))**.

Ver siguiente feed

▸ Como mencionamos anteriormente, cada usuario registrado tiene un feed de publicaciones que internamente es una **cola de prioridad** y las publicaciones a visualizar tendrán una **prioridad** con la característica de **afinidad**.

▸ El usuario loggeado podrá ver, en caso de que hayan, **todas las publicaciones realizadas** por otros usuarios, hasta que se quede sin más publicaciones para ver, cumpliendo con una complejidad de **O(log(p))**.

▸ Como las publicaciones son internamente un **TDA Post**, cuando el usuario vea las publicaciones visualizará los atributos que fueron utilizados en el TDA, tales como **ID de publicación, nombre de la persona que realizó el post, el mensaje del mismo, etc**.

Likear post

▸ El usuario loggeado podrá likear una publicación utilizando el identificador que es el ID de la publicación, y en el caso de que exista, el post recibirá un like.

▸ Para almacenarlos decidimos utilizar como atributo interno de cada publicación el **TDA Diccionario Ordenado (ABB)** por dos razones. La primera es que cada usuario de la red social solo podrá dar **un solo like** a una publicación, entonces decidimos guardar el nombre de la persona que dio like como clave del diccionario ordenado. De esta forma podemos saber si una persona **yá dio like o no** ya que el diccionario sólo puede tener claves diferentes. La segunda razón, y la más importante, es por una característica que explicaremos en el siguiente comando. Gracias a las características del diccionario ordenado, cumple con una complejidad de **O(logup)**.

Mostrar likes

▸ Como mencionamos anteriormente, cada publicación tiene internamente un **diccionario ordenado** que almacena **las personas que dieron like** a dicha publicación y a su vez **la cantidad de likes**.

▸ La razón por la que decidimos utilizar este TDA es porque, no solo debemos mostrar la **cantidad de likes**, sino que también debemos mostrar los **nombres de los usuarios que le dieron like** a dicha publicación en **orden alfabético**.

▸ Como su nombre lo indica, el diccionario guarda las claves de forma ordenada, en este caso, guarda los nombres de las personas que le dieron like al post alfabéticamente.

▸ Para mostrar a los usuarios utilizamos el **iterador** que ya nos proporciona el diccionario ordenado, de esta forma cumpliendo con una complejidad de **O(up)**.