Algoritmos y Programación II
TP N°2: AlgoGram

Docente corrector: FORTE, Federico
Nro. de grupo: 30
Integrantes:

- GENTILINI, Franco (108733)
- KOO, Hangyeol (108401)

Introducción

El trabajo práctico N°2 consistió en la implementación de una red social “Algogram” que recibe un archivo(.txt) con los nombres de usuarios registrados, en donde los usuarios pueden realizar las siguientes acciones:

- Login: ingresar a la plataforma a nombre de un usuario registrado.
- Logout.
- Publicar Post: publicar un post con texto a la vista de todos los demás usuarios, que tendrá un ID asignado según el orden de publicación.
- Ver próximo post en el feed: ver el post más “cercano” determinado con el criterio a mencionar.
- Likear un post: sumarle un like al post con el ID indicado.
- Mostrar likes: ver los usuarios que likearon el post con el ID indicado.

Los requisitos en términos de complejidad temporal de cada instrucción fueron los siguientes:

- Login: debe funcionar en O(1).
- Logout: debe funcionar en O(1).
- Publicar Post: debe funcionar en O(u log(p)), siendo u la cantidad de usuarios y p la cantidad de posts que se hayan creado hasta ese momento.
- Ver próximo post en el feed: debe funcionar en O(log(p)).
- Likear un post: debe funcionar en O(log u).
- Mostrar likes: debe funcionar en O(u).

Análisis y diseño de la solución

Para procesar el archivo con los nombres de usuarios decidimos utilizar el hash, que nos permitiría el comando de login en O(1); ya que ingresar a la plataforma con un nombre registrado implica buscar ese nombre en la base de datos, y buscar en el hash se realiza en O(1). Estos nombres de usuarios tendrán asociados como valor un TDA Usuario, que desarrollaremos más adelante.

En nuestro TP tuvimos una variable estado, que nos permite saber si hay alguien logueado y en el caso de que sí, quién está logueado. Con lo que, el logout se haría en O(1) ya que sólo tendríamos que cambiar dicho estado.

Para facilitar el manejo de los comandos, creamos un archivo auxiliar en donde fueron hechas las funciones en base a ellos para ser llamadas en el archivo main; en éste manejamos los posts y los usuarios con sus primitivas y mayoritariamente las salidas y las entradas del programa.

Para recibir los comandos principales utilizamos el scanf, recibiéndolos hasta llegar al final de la entrada estándar:

```c
char comando[50] = "";
while (scanf("%s", comando) != EOF) {
    getchar();
    if (strcmp(comando, "login") == 0) {
        login(usuarios, estado);
    } else if (strcmp(comando, "logout") == 0) {
        logout(estado);
    } else if (strcmp(comando, "publicar") == 0) {
        publicar(usuarios, *estado, posts);
    } else if (strcmp(comando, "ver_siguiente_feed") == 0) {
        ver_siguiente_feed(*estado);
    } else if (strcmp(comando, "likear_post") == 0) {
        likear_post(posts, *estado);
    } else if (strcmp(comando, "mostrar_likes") == 0) {
        mostrar_likes(posts);
    }
}
```

Siendo que los posts tienen un ID que coincide al orden de publicado, decidimos usar un TDA vector(creciente) personalizado para que sólo se le pueda sumar al final de ello, así pudiéndose acceder a un post por ID obteniendo el ítem del mismo índice.

Fueron creados TDAs Usuario y Post, a fin de poder emprolijar y manejar más eficientemente los códigos:
El TDA Post contiene los datos del usuario publicador, su ID, el contenido, la cantidad de likes y un ABB con los usuarios que likearon el post; este último utiliza dicha estructura a fin de poder realizar Likear post en O(log u) pudiendo verificar si el usuario ya likeó el post en esa complejidad y Mostrar likes en O(u), con el peor caso del post likeado por todos los usuarios.
El TDA Usuario contiene los datos nombre, su ID(definido por su orden en la base de datos), un feed y un vector de posts publicados por sí mismo. El feed es un heap que guarda los posts en base a la afinidad, siendo ésta la proximidad en el archivo de usuarios entre el dueño del feed y el del post.

Dicho lo previo, publicar el post se realizará en O(u log p) ya que agregar un post en el vector toma O(1) y sumar un post en un heap tomará O(log p); y esto lo hacemos para todos los usuarios cada vez que se publica un post para que todos los usuarios tengan el feed actualizado. Todos los posts se crean en esta instancia mediante la primitiva del TDA Post. Ver próximo en el feed costará O(log p) de esta forma ya que en nuestro TP se desencola el post del feed del usuario logueado para ello.