

Trabajo Práctico - Algoritmos y Estructuras de Datos

Licenciatura en Tecnologías Digitales, UTDT

Primer semestre 2022

- El TP se debe realizar en grupos de 3 personas (no es necesario que las 3 personas estén en la misma comisión).
- La fecha de entrega es hasta el viernes 24 de junio inclusive.
- Se evaluará no solo la correctitud técnica de la solución propuesta sino también la claridad del código escrito.

Descripción del problema

Se desea implementar un tipo de datos `RedSocial` que modela un sistema de registro e interacciones de usuarios en una red. Cada usuario está identificado con un entero `id` y un `string` `alias` que son únicos para ese usuario (no hay dos usuarios con el mismo `id` o el mismo `alias`). Todos los `alias` tienen como máximo 100 caracteres. Los usuarios pueden conectarse estableciendo una relación de amistad, esta relación siempre es simétrica (si A es amigo de B entonces B es amigo de A). El usuario más popular será aquel que tenga la mayor cantidad de amigos, y es de interés saber cuál es su conjunto de amigos.

Consigna

1. Definir una estructura de representación en el archivo `RedSocial.h` que permite satisfacer los **requerimientos de complejidad**.
2. Escribir como comentario en `RedSocial.h` el invariante de representación (en español) que debe cumplir la estructura elegida.
3. Escribir en el archivo `RedSocial.cpp` la implementación de los métodos respetando los **requerimientos de complejidad**. No está permitido modificar la interfaz pública de la clase.

Sugerencias para la estructura de representación:

- Utilizar clases provistas por la *Biblioteca Estándar* de C++, aprovechando sus órdenes de complejidad.
- No es necesario diseñar estructuras manejando memoria dinámica de manera explícita.
- Algunas funcionalidades pueden requerir uso/almacenamiento de iteradores.

Interfaz de la clase

```

1 class RedSocial{
2     public:
3         // Constructores
4         RedSocial();
5
6         // Observadores
7         const set<int> & usuarios() const;
8         string obtener_alias(int id) const;
9         const set<string> & obtener_amigos(int id) const;
10        int cantidad_amistades() const;
11
12        // Modificadores
13        void registrar_usuario(string alias, int id);
14        void eliminar_usuario(int id);
15        void amigar_usuarios(int id_A, int id_B);
16        void desamigar_usuarios(int id_A, int id_B);
17
18        // Otras operaciones
19        int obtener_id(string alias) const;
20        const set<string> & amigos_del_usuario_mas_popular() const;
21
22    private:
23        /* ... */
24 };

```

Notar que los métodos `usuarios()`, `obtener_amigos(int id)` y `amigos_del_usuario_mas_popular()` devuelven un contenedor *por referencia*. Esto significa que al momento en que la función devuelve el contenedor no se computa ningún costo de copiarlo.

Requerimientos de complejidad

<code>RedSocial()</code>	$O(1)$
<code>usuarios()</code>	$O(1)$
<code>obtener_alias(int id)</code>	$O(\log n)$
<code>obtener_amigos(int id)</code>	$O(\log n)$
<code>cantidad_amistades()</code>	$O(1)$
<code>registrar_usuario(string alias, int id)</code>	$O(\log n)$
<code>eliminar_usuario(int id)</code>	sin requerimiento
<code>amigar_usuarios(int id_A, int id_B)</code>	sin requerimiento
<code>desamigar_usuarios(int id_A, int id_B)</code>	sin requerimiento
<code>obtener_id(string alias)</code>	sin requerimiento
<code>amigos_del_usuario_mas_popular()</code>	$O(1)$

El tamaño n es la cantidad de usuarios registrados en la `RedSocial`.

Descripción detallada de las operaciones

- `RedSocial();`

Pre: Verdadero

Post: Construye una red social vacía.

- **const** set<int> & usuarios() **const**
Pre: Verdadero
(Devuelve *por referencia* el conjunto de identificadores (id) de todos los usuarios actualmente registrados)
- string obtener_alias(int id) **const**
Pre: id está en el conjunto usuarios()
(Devuelve el alias del usuario correspondiente al identificador id)
- **const** set<string> & obtener_amigos(int id) **const**
Pre: id está en el conjunto usuarios()
(Devuelve *por referencia* el conjunto de los alias de los amigos del usuario id)
- **int** cantidad_amistades() **const**
Pre: Verdadero
(Devuelve la cantidad total de relaciones de amistad actualmente en la red. Si dos usuarios A y B son amigos eso cuenta como una relación de amistad.)
- **void** registrar_usuario(string alias, int id)
Pre: id no está en el conjunto usuarios(), alias no está asociado a ningún id, y la longitud de alias es menor o igual a 100.
Post: El conjunto usuarios() tiene un elemento más: id; y obtener_alias(id) devuelve alias.
- **void** eliminar_usuario(int id)
Pre: id está en el conjunto usuarios()
Post: Se elimina al usuario id del sistema y se eliminan todas las relaciones de amistad de las que participaba.
- **void** amigar_usuarios(int id_A, int id_B)
Pre: id_A y id_B están en el conjunto usuarios()
Post: El conjunto obtener_amigos(id_A) tiene un elemento más: id_B y el conjunto obtener_amigos(id_B) tiene un elemento más: id_A.
- **void** desamigar_usuarios(int id_A, int id_B)
Pre: id_A y id_B están en el conjunto usuarios() y son amigos.
Post: El conjunto obtener_amigos(id_A) tiene un elemento menos: id_B. El conjunto obtener_amigos(id_B) tiene un elemento menos: id_A.
- **int** obtener_id(string alias) **const**
Pre: Hay un id en usuarios() que tiene asociado el alias alias.
Post: Devuelve el id de usuarios() que tiene asociado el alias alias.
- **const** set<string> & amigos_del_usuario_mas_popular() **const**
Pre: cantidad_amistades() es mayor a cero.
Post: Devuelve *por referencia* el conjunto de los alias de los amigos del usuario más popular (aquel que tenga la mayor cantidad de amigos de la red). Si hay más de un usuario más popular, devuelve los amigos correspondientes a cualquiera de ellos.