UMAG Universidad de Magallanes | Facultad de Ingeniería

DevGraph: Simulación de red social

Franco Aguilar[†], **Iván Mansilla**[†], **Diego Sanhueza**[†], **Duvan Figueroa**[†] y **Miguel Maripillan**[†]

[†] *Universidad de Magallanes*

Informe fue compilado el día 13 de diciembre de 2024

Resumen

POR HACER

Keywords: *C, Grafos, Red Social, Simulación, Estructuras de Datos*

Índice

1. Introducción

Una red social es una aplicación diseñada para conectar a las personas y permitirles compartir información a través de publicaciones. Estas plataformas toman en cuenta los intereses y preferencias de los usuarios, ayudándoles a formar comunidades, establecer amistades y ver contenido relevante basado en sus gustos o conexiones.

DevGraph es una simulación de una red social que se enfoca en la creación de usuarios, la conexión entre ellos, la visualización y creación de publicaciones, y la posibilidad de compartirlas. Además, incluye algoritmos que simulan sugerencias de nuevos amigos y publicaciones relevantes para los usuarios.

Para su implementación en C, se utilizaron algoritmos y estructuras de datos para simular DevGraph como:

- **Estructuras de datos:**
  - **Grafos:** En representación de las conexiones entre usuarios, donde cada persona corresponde a un vértice y cada conexión una arista en un grafo dirigido.
  - **Tablas Hash:** Para una busqueda eficiente de datos en la red social.
  - **Colas de prioridad (Heap):** Para crearon recomendaciones de usuarios y publicaciones en base a nuestros intereses o amigos.
  - **Listas enlazadas:** Para manejar publicaciones como listas enlazadas de nodos.
- **Algoritmos:**
  - **Algoritmo de caminos cortos (Dijkstra):** Para calcular la distancia de un nodo a otro en un grafo y de ésta manera encontrar usuarios cercanos en la red.
  - **Similaridad de Jaccard:** la cual mide el peso del enlace en base a los intereses comunes entre dos usuarios.
- **Funcionalidades adicionales:**
  - Generación de usuarios aleatorios.
  - Almacenamiento y recuperación de datos.
  - Perfil psicológico.

2. Objetivo Principal

El objetivo principal del proyecto es crear una simulación de una red social eficiente y rápida, combinando armónicamente los distintos tipos de estructuras de datos vistos durante el curso de Estructuras de Datos (grafos, listas, hash tables, colas de prioridad, etc.).

2.1. Objetivos secundarios

- Implementar algoritmos de búsqueda eficientes, priorizando el tiempo de respuesta.
- Implementar buenas prácticas de codificación.
- Reforzar habilidades de trabajo en equipo y coordinación de tareas entre pares.
- Reforzar habilidades de investigación y análisis de datos.

3. Planteamiento del desarrollo del proyecto

3.1. Grafos

Un grafo es una estructura de datos utilizada para modelar relaciones entre entidades. En este caso, se utilizarán grafos para representar nuestra red social, donde cada nodo será un usuario. Se detallarán las características principales que contiene el grafo desarrollado, explicando las decisiones tomadas. Para implementar el grafo, se utilizó la representación de listas de adyacencia, dado que esta es eficiente en términos de memoria y se adapta bien a grafos dispersos.

3.1.1. Estructuras de Datos

- Nodo: Representa cada conexión entre vértices.
- Lista: Almacena todos los nodos conectados a un vértice específico.
- Grafo: Contiene el número total de vértices y un arreglo de listas.

3.1.2. Creación del Grafo

La función `crearGrafo` inicializa un grafo con un número especificado de vértices.

```
Grafo* crearGrafo(int numVertices) {
    Grafo* grafo = malloc(sizeof(Grafo));
    grafo->numVertices = numVertices;
    grafo->listas = malloc(numVertices * sizeof(Lista));

    for (int i = 0; i < numVertices; i++) {
        grafo->listas[i].cabeza = NULL;
    }

    return grafo;
}
```

3.1.3. Adición de Aristas

La función `agregarArista` añade una conexión (arista) entre dos usuarios.

```
void agregarArista(Grafo* grafo, int origen, int destino) {
    Nodo* nuevoNodo = malloc(sizeof(Nodo));
    nuevoNodo->vertice = destino;
    nuevoNodo->siguiente = grafo->listas[origen].cabeza;
    grafo->listas[origen].cabeza = nuevoNodo;

    // Para grafos no dirigidos
    nuevoNodo = malloc(sizeof(Nodo));
    nuevoNodo->vertice = origen;
    nuevoNodo->siguiente = grafo->listas[destino].cabeza;
    grafo->listas[destino].cabeza = nuevoNodo;
}
```

3.2. Generación de usuarios aleatorios

La generación de usuarios aleatorios es un proceso común en simulaciones y pruebas de software. En este caso, se diseñó un algoritmo para generar nombres de usuarios aleatorios utilizando listas predefinidas de nombres y apellidos, combinándolos de forma aleatoria, lo que permite crear usuarios en nuestra red social. Estos usuarios posteriormente serán usados para conectarlos, crear sus intereses, analizar su perfil y generar un perfil psicológico.

3.2.1. Proceso de Generación

- **Definición de Listas:** Se crearon dos listas predefinidas: una para los nombres y otra para los apellidos, que contienen un conjunto variado de cadenas de texto representando nombres comunes.
- **Selección Aleatoria:** Para cada usuario, se selecciona aleatoriamente un nombre y un apellido de sus respectivas listas usando la función `rand()`.
- **Concatenación:** Una vez seleccionados el nombre y apellido, se concatenan para formar un nombre completo.
- **Asignación:** El nombre generado se asigna al campo correspondiente del usuario.

3.2.2. Ejemplo de Código

```
const char *nombres[] = {"Carlos", "María", "Pedro", "Ana"};
const char *apellidos[] = {"Pérez", "Gómez", "Rodríguez", "López"};
```

```
for (int i = 0; i < quantity; i++) {
    int nameIndex = rand() % numNames;
    int usernameIndex = rand() % numUsernames;
    int passwordIndex = rand() % numPasswords;

    char *name = strdup(names[nameIndex]);
    char *username = strdup(usernames[usernameIndex]);
    char *password = strdup(passwords[passwordIndex]);

    if (search_in_hash_table(table, username)) {
        printf("Advertencia: El nombre de usuario '%s' ya existe. Saltando...\n", username);
        free(name);
        free(username);
        free(password);
        continue;
    }

    User newUser = create_new_user(username, password, name, table, graph, globalInterests);
    if (!newUser) {
        printf("Error al crear el usuario '%s'.\n", username);
        free(name);
        free(username);
        free(password);
        continue;
    }

    for(int i=0; i<rand()%globalInterests.numInterests; i++) {
        add_interest(newUser, globalInterests, rand()% globalInterests.numInterests);
    }

    printf("Usuario creado: %s (%s)\n", name, username);

    free(name);
    free(username);
    free(password);
}
```

3.2.3. Ventajas y Limitaciones

Ventajas:

- Permite simular usuarios según la cantidad indicada por el usuario.
- Garantiza la variabilidad en los datos generados.

Limitaciones:

- La calidad de los nombres generados depende del tamaño y diversidad de las listas predefinidas.
- En escenarios de gran escala, es posible que se repitan nombres si las listas son pequeñas.

3.3. Conexión entre Usuarios

La conexión entre usuarios en el sistema se implementa mediante un grafo, donde cada nodo representa un usuario y las aristas indican la relación entre ellos. Esta estructura permite modelar redes sociales o sistemas similares.

3.3.1. Proceso de Creación de Conexiones

- **Inicialización del Grafo:** El grafo se inicializa en memoria, permitiendo agregar nodos y conexiones dinámicamente.
- **Selección Aleatoria de Conexiones:** Se selecciona aleatoriamente un número de conexiones para cada usuario, limitado por un valor máximo.
- **Verificación de Conexiones:** Se verifica que las conexiones no sean duplicadas ni auto-conexiones.
- **Establecimiento de Conexiones:** Si la conexión es válida, se actualiza la lista de conexiones de ambos usuarios.

3.3.2. Ventajas y Limitaciones

Ventajas:

- Proporciona una forma sencilla y efectiva de simular una red de usuarios.
- La implementación dinámica permite adaptarse a cambios en el tamaño del sistema.

Limitaciones:

- La selección aleatoria podría generar distribuciones desiguales de conexiones.
- En sistemas de gran escala, el proceso puede ser computacionalmente costoso.

3.4. Implementación de Dijkstra

El algoritmo de Dijkstra es una técnica eficiente para encontrar la ruta más corta en un grafo ponderado. En este caso, se utiliza para encontrar usuarios cercanos en la red social, asignando un peso entre las conexiones.

3.4.1. Descripción del Algoritmo

Este algoritmo se utiliza para calcular las distancias mínimas desde un nodo fuente hacia todos los demás nodos. Una vez que se calculan las distancias, es posible filtrar a los usuarios más cercanos mediante un parámetro.

3.4.2. Estructura de Datos

```
struct _edge {
    User dest; /*!< Usuario destino */
    double weight; /*!< Peso de la arista */
    Edge next; /*!< Siguiente en la lista de aristas */
};
```

3.5. Planteamiento para las recomendaciones de amistades y publicaciones

Para lograr estas recomendaciones se decidió utilizar colas de prioridad (heaps) para guardar las recomendaciones de amistades y las publicaciones, y ordenarlas de manera que cuando extraigamos un elemento de la cola, este sea el máximo o el mínimo, dependiendo del tipo de cola.

Para simular las recomendaciones de amistades, se planteo utilizar Dijkstra o BFS para calcular la distancia de un nodo a otro en un grado y así simular sugerencia de amigos de amigos. Además, se planteo utilizar Jaccard para calcular la similitud entre los usuarios, para poder calcular cuántos usuarios comparten un interes común con otro.

Para simular las publicaciones, se planteo que primero se mostrarían las publicaciones de los usuarios a los que seguimos, seguidos de estas publicaciones, vendrán aquellas publicaciones recomendadas en base a los usuarios con los que tenemos intereses comunes (jaccard).

4. Implementación

La implementación del proyecto fue realizada en lenguaje C, utilizando librerías estándar. Donde el código fue escrito de manera modular, o sea, las funciones fueron separadas en diferentes archivos según su funcionalidad. Esto permitió una mayor manera de organización y mantenimiento del código, así como una mejor comprensión del funcionamiento del programa.

4.1. Estructura de directorios

En el directorio principal se encuentran los archivos `Makefile`, con el cual es posible compilar el programa de manera automática, y el archivo `README.MD`, que contiene información sobre el proyecto y su funcionamiento. Además, se encuentra el archivo `subtopics`, que contiene una lista de subtópicos modificable, que se utilizará para el funcionamiento de la aplicación.

Dentro del directorio `src` se encuentran los archivos fuente del programa, que son los siguientes:

- `main.c`: Contiene el flujo principal del programa.
- `database.c`: Contiene las funciones para la gestión y guardado de la base de datos.
- `graph.c`: Contiene las funciones para gestionar los grafos y relaciones entre usuarios.
- `hash_table.c`: Contiene las funciones para gestionar tablas hash.
- `users.c`: Contiene las funciones para gestionar usuarios y publicaciones.
- `utilities.c`: Contiene las funciones de utilidad y testing para el programa.
- `heaps.c`: Contiene las funciones para gestionar las colas de prioridad (heaps) para publicaciones y sugerencias de amistades.

Por otro lado, en el directorio `incs` se encuentran los archivos de cabeceras correspondiente a cada uno de los archivos fuente.

4.2. Colas de prioridad (heaps)

La cola de prioridad es una estructura de datos que se utilizó para almacenar sugerencias de amistades con respecto a sus intereses y amigos de amigos, y publicaciones en base a nuestros intereses y/o amigos.

Para las sugerencias de amigos de amigos se utilizó el algoritmo de Dijkstra para calcular la distancia de un nodo a otro en un grafo, y de ésta manera lograr encontrar usuarios cercanos en la red con amigos en común. Una vez calculada la distancia, ésta se utilizará como prioridad para la sugerencia de amistad, para que los usuarios con amigos en común se ordenen de menor a mayor distancia.

Para la sugerencia de amistades según los intereses de los usuarios, se utilizó el algoritmo de similaridad de Jaccard, que mide la proporción entre la intersección y la unión de los conjuntos de intereses o conexiones compartidas entre dos usuarios. Este algoritmo se utilizó para calcular la similitud entre los usuarios donde mientras menor sea el resultado, más similar serán sus intereses.

Para la visualización de publicaciones se considerarán prioridad las publicaciones de nuestros seguidos, y luego de éstas las publicaciones de aquellos usuarios con los que tengamos intereses comunes calculado previamente con jaccard como lo hicimos con la sugerencia de amistades según los intereses.

Esta cola de prioridad (heap), es una estructura de datos que organiza elementos de manera que siempre se pueda tener acceso rápido al elemento de menor o mayor prioridad, dependiendo del tipo de heap (mínimo o máximo). En nuestro caso trabajamos con el máximo, es decir que el elemento con mayor prioridad será el primero en la cola (se encuentra en la raiz). Para mantener las propiedades de ésta estructura, se implementaron funciones para insertar, extraer máximos y mínimos, funciones para ordenar la cola de prioridad.

Las principales funciones que fueron implementadas para mantener estas propiedades de la cola de prioridad (heap) fueron las de heapify_up y heapify_down.

- **heapify_up:** Esta función ordena la cola prioridad de manera que, al insertar un nuevo elemento al final de la cola, este es comparado con su padre, intercambiándolos si la prioridad del nuevo elemento es mayor. Esto se repite hasta que se cumpla la propiedad o el elemento llegue a la raiz de la cola.
- **heapify_down:** Esta función ordena la cola de prioridad de manera que, después de eliminar o extraer el máximo elemento de la cola, el elemento al final de la cola es ubicado en la raiz de la cola y se compara con sus hijos, se intercambia si la prioridad de del hijo mayor es mayor a la del elemento en la raiz (padre). Esto continúa hasta que se cumpla la propiedad o el elemento no tenga más hijos.

5. Gestión del equipo de trabajo

El equipo de trabajo constó de 5 personas, uno de ellos designado por el profesor como líder del grupo, con la responsabilidad de coordinar el trabajo y la gestión del proyecto. Debido a la cantidad de personas involucradas, fue fundamental establecer mecanismos de comunicación, coordinación y orden para asegurar la colaboración y la coordinación de los diferentes miembros del equipo. Algunos de estos mecanismos fueron:

5.1. Normas de codificación

Las normas de codificación fueron establecidas para el proyecto, con el objetivo de garantizar la coherencia y la consistencia en la codificación. Estas normas incluían:

- **Constantes**: Para las constantes se utilizó la convención de `SCREAMING_SNAKE_CASE`, con el objetivo de diferenciarlas de las variables y funciones.
- **Variables:** Para las variables comunes se utilizó la convención `camelCase`
- **Funciones:** Para las funciones se utilizó la convención `snake_case`, así diferenciándose de las variables y constantes.
- **Tipos de datos:** Para los tipos de datos creados con `typedef` se utilizó la convención `PascalCase`, con el objetivo de diferenciarlos de los tipos de datos nativos de C.
- **Llaves de apertura:** Se estableció la convención de que las llaves de apertura se escribieran en la misma línea que el código que las contiene

```
#define MAX_CHAR 256 // Constante
typedef _User *User; // Tipo de dato
Graph newGraph; // Variable
void print_all_users(Graph graph); // Funcion

```

Código 1. Ejemplo de código, utilizando las normas de codificación

5.2. Comunicación

Un aspecto de gran importancia para el proyecto es la comunicación entre los miembros del equipo, ya que sin ella, no sería posible garantizar la colaboración y la coordinación en el desarrollo del proyecto. Debido al tamaño del equipo y la complejidad del proyecto esto fue un desafío, debido a las diferencias en horarios y responsabilidades académicas de cada miembro. Para resolver este problema fue escencial establecer un medio de comunicación remoto, que permitiera a cada uno comunicarse de manera rápida y en el lugar en que estuviese.

Se creó un chat en línea con cada integrante del equipo, el cual se utilizó para discutir sobre el desarrollo del proyecto y su organización, así como para compartir información y recursos.

Esto resultó exitoso, ya que las reuniones presenciales resultaron ser extremadamente dificiles de coordinar. La comunicación remota nos permitió a los miembros ver y discutir los detalles del proyecto en el momento del día en que estuviesen disponibles. Sin embargo, esto también implicó consecuencias, ya que dio paso a que fuera fácil que algún miembro del equipo "desapareciese"; sin embargo, estas situaciones fueron manejadas a través de la comunicación por privado con éste para consultar su situación.

5.3. División del trabajo

Al inicio del proyecto se decidió establecer **objetivos a corto plazo** para cada miembro del equipo, según las estructuras de datos que se deseaban implementar en el proyecto. A cada uno se le asignaron según afinidad, capacidad y gusto propio.

Una vez las estructuras de datos hechas, se decidió crear una **lista de tareas** a realizar, cada miembro con al menos una tarea a realizar. Esta lista fue creada en base a las ideas discutidas por los integrantes del equipo, y se fue actualizando constantemente en función de las necesidades del proyecto.

A continuación, se presenta una descripción detallada de las tareas asignadas y realizadas por cada miembro del equipo:

- **Duvan Figueroa:**
  Duvan se encargó de la implementación de las **tablas hash** que permiten el almacenamiento eficiente de datos en la red social. Además, trabajó en la **organización de publicaciones**, gestionando cómo se almacenan y visualizan las publicaciones en la plataforma.
- **Diego Sanhueza:**
  Diego fue responsable de la implementación de la **estructura de grafos** en conjunto con Miguel, lo que facilita la representación de las conexiones entre usuarios. También desarrolló el **algoritmo de Dijkstra** para encontrar usuarios cercanos basados en la ubicación o afinidades. Además, trabajó en la creación de perfiles psicológicos de los usuarios y en la implementación de **usuarios aleatorios** para probar el sistema.
- **Iván Mansilla:**
  Iván implementó las **listas enlazadas** para manejar las publicaciones, así como la creación de **estructuras de usuarios**, las cuales incluyen la lista enlazada de publicaciones. También se encargó de agregar **temas/tópicos** en la estructura de usuarios y publicaciones, y de implementar el índice **Jaccard** para medir las similitudes entre usuarios y publicaciones. Además, fue responsable de la asignación de intereses a los usuarios y de la implementación de funcionalidades para **iniciar sesión** y **publicar** según el usuario iniciado. Por último, implementó los comandos para publicar, borrar, ver usuarios, perfil, publicaciones, entre otros.

- **Franco Aguilar:**
  Franco fue responsable de la implementación de **colas de prioridad**, que permiten gestionar las tareas de manera eficiente y priorizar las acciones dentro de la red social.
- **Miguel Maripillan:**
  Miguel trabajó en conjunto con Diego en la implementación de la **estructura de grafos** y los **algoritmos BFS/DFS** para explorar las conexiones entre usuarios de manera eficiente. Además, se encargó del **almacenamiento de la información de los usuarios** en la nube y de la **carga de usuarios** a través de un archivo `txt` de entrada.

6. Posibles mejoras

En este punto se presentarán las posibles mejoras que se podrían realizar en este programa, las cuales pueden ser varias dado el tiempo y complejidad asignado a este proyecto.

- Mejoras en la interacción con el usuario: al visualizar las publicaciones recomendadas y sugerencias de amigos se podría incluir una función para guardar publicaciones que le gustaron al usuario y además agregar una opción para seguir a otro usuario.
- Inclusión de más topicos: En este proyecto para los intereses se utilizaron intereses comunes con respecto a la programación, pero existen muchos otros temas de interes generales que podrían ser agregados para simular una red social más amplia.
- Implementación de un sistema de mensajería: En este proyecto se utilizó una interfaz de línea de comandos para interactuar con el programa, pero se podría implementar un sistema de mensajería para permitir a los usuarios comunicarse entre sí y con otros usuarios.
- La creación de grupos o comunidades: Para este tipo de programas es posible crear grupos o comunidades en la red social con respecto a un tema o usuario en particular, para que los usuarios puedan conectarse con otros usuarios de la misma comunidad.

7. Ejemplo de uso

POR HACER

8. Conclusiones

POR HACER

9. Referencias