**TU AMIGO UN**

**Juan Camilo López Bustos, Alejandro Mendivelso Torres**

**No. de equipo de trabajo: {2do}**

# I. INTRODUCCIÓN

En esta entrega final veremos el esquema del

programa “Tu amigo UN”, que espera proveer una solución al problema de socialización y aislamiento presentado por una gran parte de los estudiantes de la universidad nacional.

También veremos la construcción del programa, sus parámetros y las diferentes opciones que presentara para esta entrega final, los resultados esperados y una descripción muy precisa de las estructuras de datos implementadas y como se usan dentro del programa.

Además de un análisis muy completo de los casos de prueba y con las diferentes estructuras de datos y cómo se comportan en los diferentes requerimientos como añadir, borrar y buscar un dato.

# II. DESCRIPCIÓN DEL PROBLEMA A RESOLVER

I. Es bien sabido que el aislamiento social a causa del covid19 ha dejado muchos problemas por resolver, uno de estos es el foco de este proyecto, y es el causado por el mismo aislamiento y la falta de interacción por parte de los jóvenes pertenecientes a la universidad nacional de Colombia, ya que se ha demostrado que el aislamiento prolongado puede causar problemas de interacción social y física[1], problema que ha tratado de ser mitigado con diferentes actividades culturales y espacios de reunión que ofrece la universidad y la sección de bienestar, pero que a pesar de esto muchos estudiantes se han quejado constantemente del hecho de no poder socializar o simplemente no saber cómo hacerlo ya que carecen tanto de habilidades como de herramientas que les faciliten dicha interacción.

# III. USUARIOS DEL PRODUCTO DE SOFTWARE

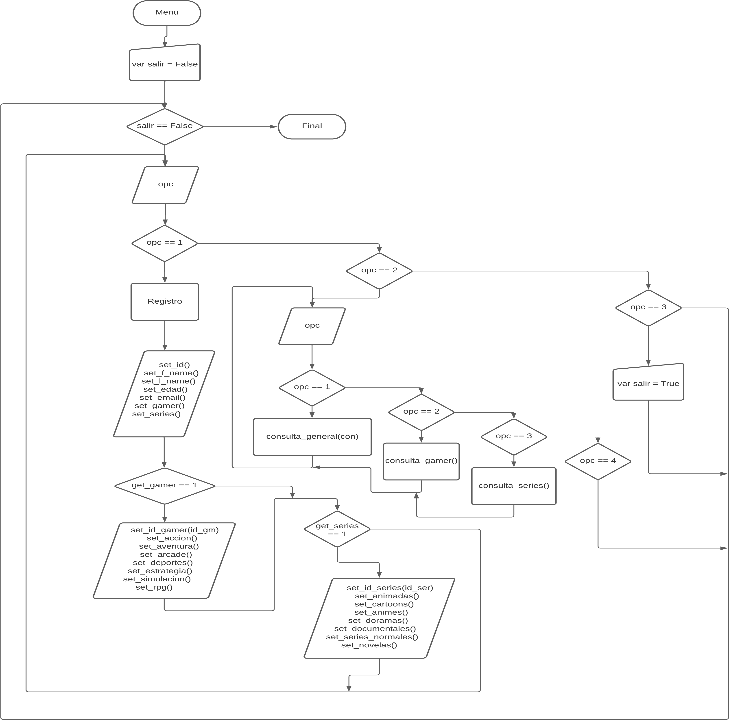
Este producto está destinado a cualquier estudiante de la universidad nacional de Colombia o graduado de la misma, que quiera conocer personas nuevas con sus mismos o diferentes gustos en cuanto a actividades de ocio, no es necesario tener algún conocimiento avanzado en computación ya que el programa está pensado como una plataforma de búsqueda y registro para crear nuevas amistades y fomentar espacios de ocio y esparcimiento.

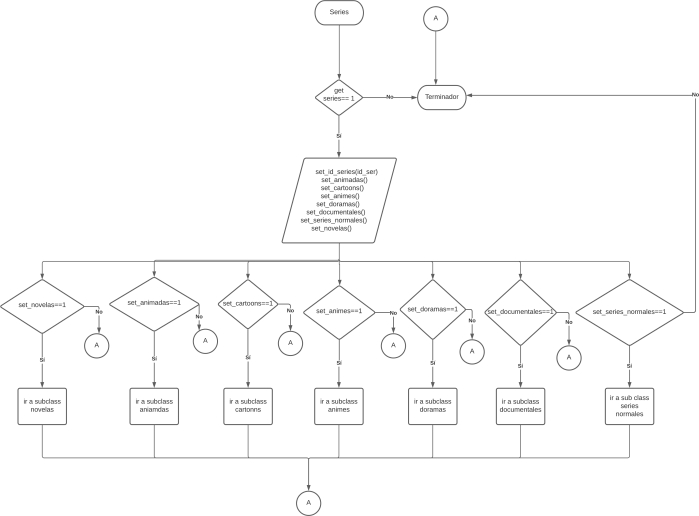
IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE En esta sección veremos los requerimientos mínimos de funcionamiento, y cómo se comporta cada una de las funcionalidades del programa.

* Registro: En esta parte del programa se realiza el registro de los datos del usuario y la selección de sus preferencias, para posteriormente ser almacenadas en una base de datos. El usuario deberá dar clic en la opción de registro, posteriormente diligenciar los datos para ser capturados y guardados temporalmente y dependiendo de su selección de gustos serán guardadas sus preferencias para finalmente ser enviadas a la base de datos.

* Búsqueda: En esta parte del programa el usuario podrá realizar diferentes tipos de búsqueda dependiendo de las opciones que el elija, desde una búsqueda avanzada, hasta una búsqueda especifica por cada una de las categorías disponibles. El usuario puede seleccionar diferentes opciones de búsqueda que le enviarán y mostrarán múltiples datos en su pantalla dependiendo de las opciones que este elija, además esta información se guarda de manera temporal para en versiones futuras del programa ser ordenada de manera alfabética, tales como categoría de gamer o series donde hay subcategorías de búsqueda más avanzadas.

A continuación, se presentan dos diagramas de flujo que muestran la funcionalidad de registro y búsqueda de manera general, se espera para versiones futuras poder implementar una actualización de datos de usuario, y el ordenamiento de datos de manera alfabética.





# V. DESCRIPCIÓN DE LA INTERFAZ DE USUARIO PRELIMINAR

Se hizo esta estructura grafica sencilla para que sea mas intuitiva al usuario, juntos con el registro de los usuarios.

Interfaz de usuario gráfica

Descripción generada automáticamente

# VI. ENTORNOS DE DESARROLLO Y DE OPERACIÓN

El programa es desarrollado en Python, y se espera que se pueda manejar atreves de una plataforma web, para todos los sistemas operativos comerciales actuales (Windows, macOS, Linux, Android) y para equipos con características de hardware de bajo desempeño (desde 2gb de RAM, procesador de 2 núcleos 4 hilos) , ya que se espera que también se pueda acceder desde dispositivos móviles.

# VII. PROTOTIPO DE SOFTWARE INICIAL

A continuación, se deja el enlace de la carpeta de GitHub que contiene el programa y todos los archivos necesarios para que funcione.

<https://github.com/Zecro-Darkness/Entrega-final-estructura-de-datos.git>

# VIII. DISEÑO, IMPLEMENTACIÓN Y APLICACIÓN DE LAS ESTRUCTURAS DE DATOS

Para diseñar este programa se hizo uso de una lista simple enlazada, ya que en esta se podían guardar los datos de manera lineal y crear una lista de listas para almacenar la información de consulta.

Para poder implementar está lista enlazada se definió una clase nodo con algunos atributos simples, que son valor (value) y valor siguiente (Next) que ayudan a enlazar los datos almacenados, también se definieron métodos para obtener los datos como caracteres y como enteros para compararlos, luego se definió como tal la clase lista enlazada (Linkedlist) que tiene también atributos simples que son el primer ítem (First) y el tamaño (Size), también se definieron métodos para la inserción de datos de manera individual (append) donde el dato se inserta al final de la lista, también uno de eliminación (remove) que recorre la lista hasta encontrar el dato a remover y uno de limpiar (limpiar) que deja vacía la lista, y un método tamaño(len) que retorna el tamaño de la lista y un método de caracteres (str) que convierte la información guardada en los nodos en caracteres.

Debido a que la lista enlazada simple logró satisfacer la totalidad de las funciones del programa de manera alterna se empleó una cola (Queue) para comparar los tiempos de ejecución y funcionalidad, para esta clase cola se implementó un nodo (Nodo\_q) en una estructura de lista doblemente enlazada, por lo que el nodo se compone de las propiedades de valor (data) el nodo siguiente (next) y el anterior (prev). Para la clase cola (Queue) se definió el atributo de cabeza (head) y cola (last) que hacen referencia al primer elemento en la cola y el último respectivamente, luego están los métodos de encolar (enqueue) donde se inserta en la información en un nodo que irá al final de la cola y será referenciado por el anterior y el siguiente, luego está el método de desencolar (dequeue) que saca el nodo de la cabeza y lo reemplaza con el anterior guardando su respectiva referencia, después está el método primero (first) que retorna el primer elemento de la cola (head), y el método último (last) que retorna el ultimo elemento de la cola o encolado, seguido de este está el método tamaño (size) que devuelve la cantidad de elementos encolados y el método vació (is\_empty) para comprobar si hay elementos en la cola, y el método de imprimir que recorre la cola elemento a elemento e imprime su valor.

Para la segunda versión del programa, se hizo uso de un árbol AVL que hace uso de una clase nodo exclusiva llamada nodo\_avl que consta de cinco atributos, que son label, parent, left, rigth y height. El primer atributo guarda la información dentro del nodo el segundo es un apuntador al nodo padre de este, el cuarto y quinto atributo son apuntadores a los posibles nodos hijos y el último guarda la altura del nodo. De igual manera se definieron funciones setter y getter para cada atributo de la clase nodo.

Luego tenemos la clase AVL que contiene dos atributos muy simples, root y size, root hace referencia a la raíz o el nodo principal al que apuntan todos, y size hacer referencia los niveles del árbol. Luego tenemos los métodos insert, rebalance, rotate left, rotate rigth, double rotate left, double rotate right, empty, preshow, pre order y get root. El primer método inserta los datos dentro del arbol de manera que cumpla con las reglas de un Avl ingresando los datos mayores a la raíz a la derecha y los menores a la izquierda y de la misma manera para cada nodo padre, también verifica al finalizar la inserción si la altura de las hojas difiere en más de uno y hace el respectivo rebalance. Seguido de esto tenemos el método de rebalance que toma el nodo que se acaba de ingresar y procede a comparar la altura de los nodos hoja para hacer el respectivo rebalanceo si la altura entre estos difiere en más de uno, si el nodo está a la izquierda hará una rotación a la derecha y si difiere en más de dos la altura, procederá a hacer una doble rotación y de la misma manera para los nodos en la derecha si estos tienen altura mayor de uno habrá una rotación a la izquierda doble o sencilla dependiendo de la cantidad de niveles en que difiera, este método se ejecutará las veces necesarias hasta que el árbol quede completamente balanceado. Seguido a este método tenemos las rotaciones antes mencionadas que también se implementan como métodos, la rotación simple tanto a derecha como izquierda funciona de la misma manera, se guarda la información del nodo a rotar en un auxiliar y se convierte en padre y el que era padre pasa a ser hijo bien sea en rotación a la derecha, o rotación a la izquierda. En la rotación doble sucede algo similar, con la diferencia de que la rotación sucede dos veces ya que la altura difiere en más de dos. Seguido a esto tenemos los métodos preshow y preorder que realizan la búsqueda en orden descendente y en orden “pseudo-aleatorio” respectivamente en la BD, para así mostrar la información o datos al usuario. Y por último el método get root que devuelve el nodo raíz del árbol.

En la segunda versión del programa se hizo uso del AVL ingresando el número de identificación de cada usuario, por categoría y luego consultando este en la BD para mostrar la información de consulta.

Para la entrega final volvimos a crear un nodo y una lista enlazada llamada lista hash donde guardaremos los datos de los usuarios, para poder implementar la estructura de datos HASH, Luego de crear la el nodo y la lista procedimos a crear cada una de las operaciones, las cuales fueron, la de insertar búsqueda y eliminar, todo conectado al archivo DB donde tenemos los usuarios guardados, Luego creamos el HASH y sacamos el numero hash de cada id de los usuarios para tener la llave hashing con la cual podemos identificarlos, para asignarles una posición en la tabla, utilizamos un numero primo en la función hash para evitar el mayor número de colisiones posibles, así creando una matriz donde cada dato tiene su propia casilla y es una lista enlazada, para llenar todos los datos en una sola posición,

Creamos la función hash, y creamos una matriz [] y para el numero de casillas de la matriz ponemos el mismo número primo que usamos para sacar la función hash el cual es 2147483647 que es un numero mersenne el cual es 2^31-1 que es un numero primo, ya con el tamaño de la matriz (M) y el tamaño de la lista de los usuarios (N) y ya que M>>N el factor de carga será menor, haciendo un uso optimo de la tabla Hash.

En la versión final del programa se hizo uso del Hash ingresando el número de identificación de cada usuario, por categoría y luego consultando este en la BD para mostrar la información de consulta.

# IX. PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

De acuerdo las pruebas realizadas en la función de búsqueda que es la que toma más tiempo, ya que el programa debe tomar los datos de la BD y ponerlos en una lista de listas, para luego imprimirlos, para ello se hicieron pruebas tanto en la lista enlazada simple como en la cola.

Para la primera prueba de inserción de n datos se obtuvo un comportamiento de algoritmo de O(1) para ambas estructuras, con muestras de datos máxima de 250.000, los tiempos de inserción correspondientes a la lista enlazada y la cola se muestran a continuación en la tabla 1

-1

,

-0

5

0

0

,

5

1

0

50000

100000

150000

200000

250000

300000

Inserción Lista Enlazada

-1

-0

,

5

0

0

,

5

1

0

50000

100000

150000

200000

250000

300000

Inserción en Cola (enqueue)

Tabla 1

Tabla 2

Para la segunda versión se corrió la prueba de inserción para la misma cantidad de datos obteniendo un comportamiento logarítmico de la velocidad de inserción para esta estructura de datos como se observa en la tabla 3.

Para la segunda prueba de la impresión de n datos se obtuvo tanto para la cola como para la lista enlazada un comportamiento de O(N) ya que el tiempo de impresión depende de la cantidad de datos a imprimir, pues recorre uno a uno los datos guardados dentro de la estructura, se usó de igual manera una cantidad máxima de 250.000 datos obteniendo las siguientes gráficas (Tabla 4) en diferentes pruebas:

0

5

10

0

50000

100000

150000

200000

250000

300000

Impresión Linked List

0

5

10

15

0

50000

100000

150000

200000

250000

300000

Impresión Queue

Tabla 3

0

5

10

0

50000

100000

150000

200000

250000

300000

Impresión AVL

Tabla 4

Para la versión 2 también se realizaron pruebas de impresión o consulta de datos dando como resultado un comportamiento de O(N)

Para la versión final se corrió la prueba de inserción para la misma cantidad de datos obteniendo un comportamiento logarítmico de la velocidad de inserción para esta estructura de datos como se observa en la tabla 6

-1

,

-0

5

0

0

,

5

1

0

50000

100000

150000

250000

300000

Inserción Lista Enlazada

-1

-0

,

5

0

0

,

5

1

0

50000

100000

150000

200000

250000

300000

Inserción en Lista Hash

*Tabla 6*

Se realizaron las pruebas de impresión o consultas de datos dando como resultado un comportamiento O(1) en la tabla hash ya que la matriz hash es mucho mas grande que los datos (tabla7)

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Gráfico, Escala de tiempo

Descripción generada automáticamente

*Tabla 7*

X. INFORMACIÓN DE ACCESO AL VIDEO DEMOSTRATIVO DEL PROTOTIPO DE SOFTWARE

El siguiente link muestra un video sobre el software funcionando: <https://youtu.be/osZrHZyKTDk>

# XI. ROLES Y ACTIVIDADES

Se deben describir de manera concisa los roles asignados y las respectivas actividades realizadas por cada uno los integrantes del equipo durante el desarrollo de esta entrega. Para esto, tenga en cuenta los siguientes roles positivos, que puede aplicar durante el desarrollo del proyecto; los mismos deberán ser ‘rotados’ entre todos los integrantes del equipo en las diferentes entregas.

Tabla

Descripción generada automáticamente

Puede utilizar la siguiente tabla para definir los integrantes del grupo, los roles asignados y el listado de actividades durante el desarrollo de la entrega:

|  |  |  |
| --- | --- | --- |
| **INTEGRANTE** | **ROL(ES)** | **ACTIVIDADES REALIZADAS (Listado)** |
| **Juan Camilo López Bustos** | Líder | Desarrollador de la idea del proyecto |
| Experto | Crear parte del código y buscar soluciones |
| Investigador | Ayuda a resolver inquietudes sobre las  funciones del  programa |
| Observador | Revisa el progreso del programa cada 2-3  días después de distribuir las responsabilidades |
| **Alejandro**  **Mendivelso Torres** | Coordinador | Habla con todos los integrantes y  coordinar reuniones |
| Experto | Crear parte del código y buscar soluciones |
| Investigador | Ayuda a resolver inquietudes sobre las funciones del  programa |
| Observador | Revisa el progreso del programa cada 2-3  días después de distribuir las  responsabilidades |

# XII. DIFICULTADES Y LECCIONES APRENDIDAS

La dificultad más grande del proyecto fue englobar un interés en común entre las personas a la que está dirigida el proyecto para que puedan relacionarse entre sí de manera más fácil y con más confianza ya que es un tema de interés entre las personas, y también como organizar la información de la persona, los intereses, y sus gustos; y una forma fácil de relacionar los intereses de diversos usuarios para poder integrar a los usuarios.

# XIII. REFERENCIAS BIBLIOGRÁFICAS

1. Yunier Broche-Pérez, “Consecuencias psicológicas de la cuarentena y el aislamiento social durante la pandemia de COVID-19”, *Scielo.sld.cu,*

2020[En línea]. Disponible en:

http://scielo.sld.cu/scielo.php?script=sci\_arttext&pid=S0864-

34662020000500007

1. Weiss, M.A.: *Data Structures and Algorithm Analysis in C++, 4th Edition*, Pearson/Addison Wesley, 2014.
2. Hernández, Z.J. y otros: *Fundamentos de Estructuras de Datos. Soluciones en Ada, Java y C++*, Thomson, 2005.
3. Shaffer, Clifford A.: *Data Structures and Algorithm Analysis in C++, Third Edition*, Dover Publications, 2013. ([En línea.](http://people.cs.vt.edu/shaffer/Book/))
4. Campos Laclaustra, J.: *Apuntes de Estructuras de Datos y Algoritmos*,

segunda edición, 2018. [(En línea.)](https://webdiis.unizar.es/asignaturas/EDA/apuntes_EDA.pdf)

1. Martí Oliet, N., Ortega Mallén, Y., Verdejo López, J.A.: Estructuras de datos y métodos algorítmicos: 213 ejercicios resueltos. 2ª Edición, Ed. Garceta, 2013.
2. Joyanes, L., Zahonero, I., Fernández, M. y Sánchez, L.: Estructura de datos. Libro de problemas, McGraw Hill, 1999.
3. Web site: <https://pythondiario.com/2018/08/implementacion-de-un-arbol-avl.html#:~:text=El%20%C3%A1rbol%20AVL%20tiene%20la,la%20rama%20derecha%20o%20viceversa>
4. Wen site: <https://pythondiario.com/2018/06/tabla-hash-en-python.html#:~:text=La%20forma%20de%20implementar%20en,a%20la%20clave%20del%20elemento>.
5. Web site: <https://es.acervolima.com/implementacion-de-hash-con-encadenamiento-en-python/>