Árboles Binarios

Organización de Archivos

1[do] Angely Mendez
Escuela de Informática
Universidad Nacional de Trujillo
Trujillo, Perú
t052701020@unitru.edu.pe

2[ero] Ciara Mendez
Escuela de Informática
Universidad Nacional de Trujillo
Trujillo, Perú
t022700920@unitru.edu.pe

***Resumen*—Este documento es una investigación e implementación en lenguaje C++ sobre árboles binarios que impone una estructura jerárquica sobre una colección de objetos. Ejemplos claros de utilización de árboles se presentan tanto dentro como fuera del área de computación (índices de libros, árboles genealógicos, etc.). En Informática constituyen una de las estructuras más utilizadas, con aplicaciones que van desde los árboles sintácticos, hasta la representación de datos que se desea mantener ordenados con un tiempo de acceso relativamente bajo. En general, se usarán árboles siempre que se quiera representar información jerarquizada, cuando esta converja en un solo punto. En este articulo se implementó las funciones que permiten la creación de registros a través de árboles binarios, en los que se puede escribir, mostrar, buscar, modificar, eliminar registros, ver los registros eliminados y hallar el máximo izquierdo de un registro.**

***Palabras claves:*—árboles binarios, árbol, informática, C++.**

I. Introducción

El árbol es una estructura de datos muy importante en informática y en ciencias de la computación. Los árboles son estructuras no lineales, al contrario que los arrays y las listas enlazadas que constituyen estructuras lineales. La estructura de datos árbol generaliza las estructuras lineales vistas en capítulos anteriores. Los árboles se utilizan para representar fórmulas algebraicas, para organizar objetos en orden de tal forma que las búsquedas son muy eficientes, y en aplicaciones diversas tales como inteligencia artificial o algoritmos de cifrado. Casi todos los sistemas operativos almacenan sus archivos en árboles o estructuras similares a árboles. Además de las aplicaciones citadas, los árboles se utilizan en diseño de compiladores, proceso de texto y algoritmos de búsqueda. Entonces en este informe se presenta información al respecto, el cual está organizado de la siguiente manera: en primer lugar, se explican los conceptos teóricos: árboles generales, definiciones y terminología, luego se da énfasis en la implementación de las funciones en el lenguaje C++, para finalizar las conclusiones más relevantes.

II. Árboles Binarios

II-A. Árboles Generales

Intuitivamente el concepto de árbol implica una estructura en la que los datos se organizan de modo que los elementos de información están relacionados entre sí a través de ramas. El árbol genealógico es el ejemplo típico más representativo del concepto de árbol general. La Figura 1 se representa un ejemplo de árbol general, gráficamente puede verse cómo un árbol invertido, la raíz en la parte más alta de la que salen ramas que llegan a las hojas, que están en la parte baja. Un árbol consta de un conjunto finito de elementos, denominados nodos y un conjunto finito de líneas dirigidas, denominadas

ramas, que conectan los nodos. El número de ramas asociado con un nodo es el grado del nodo.

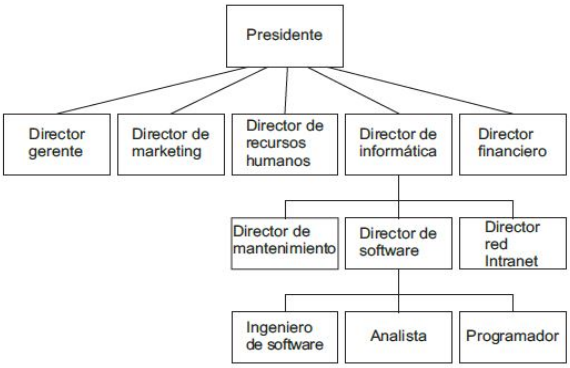

Figura 1. Estructura Jerárquica tipo árbol

II-B. ***Definición***

Según a [5],Un árbol binario es un árbol en el que ningún nodo puede tener más de dos subárboles. En un árbol binario, cada nodo puede tener, cero, uno o dos hijos (subárboles). Se conoce el nodo de la izquierda como hijo izquierdo y el nodo de la derecha como hijo derecho.

De acuerdo a [6], un árbol consiste en un conjunto finito de elementos llamados nodos, o vértices y un conjunto finito de arcos dirigidos que conectan pares de nodos. Si el árbol no es vacío, entonces uno de los nodos, llamado la raíz, no tiene arcos entrantes, pero cualquier otro nodo en el árbol puede ser alcanzado desde él siguiendo un camino único, que es una secuencia de arcos consecutivos.

Algo similar menciona [1] que, en ciencias de la computación, un árbol binario es una estructura de datos en la cual cada nodo puede tener un hijo izquierdo y un hijo derecho. No pueden tener más de dos hijos (de ahí el nombre ”binario”). Si algún hijo tiene como referencia a null, es decir que no almacena ningún dato, entonces este es llamado un nodo externo. En el caso contrario el hijo es llamado un nodo interno. Usos comunes de los árboles binarios son los árboles binarios de búsqueda, los montículos binarios y Codificación de Huffman.

También [2] menciona que, los árboles son las estructuras de datos no lineales y dinámicas de datos más importantes del área de computación. Se define un árbol binario tipo T como una estructura homogenea, resultado de la concatenación de un elemento de tipo T, llamado raíz, con unos árboles binarios disjuntos, llamados subárbol izquierdo y subárbol derecho.

II-C. ***Terminología***

Además de la raíz, existen muchos términos utilizados en la descripción de los atributos de un árbol. En la Figura 2, el nodo A es el raíz. Utilizando el concepto de árboles genealógicos, un nodo puede ser considerado como padre si tiene nodos sucesores. Estos nodos sucesores se llaman hijos. Por ejemplo, el nodo B es el padre de los hijos E y F. El padre de H es el nodo D. Un árbol puede representar diversas generaciones en la familia. Los hijos de un nodo y los hijos de estos hijos se llaman descendientes y el padre y abuelos de un nodo son sus ascendientes. Por ejemplo, los nodos E, F, I y J son descendientes de B. Cada nodo no raíz tiene un único padre y cada padre tiene cero o más nodos hijos. Dos o más nodos con el mismo padre se llaman hermanos. Un nodo sin hijos, tales como E, I, J, G y H se llaman nodo hoja.

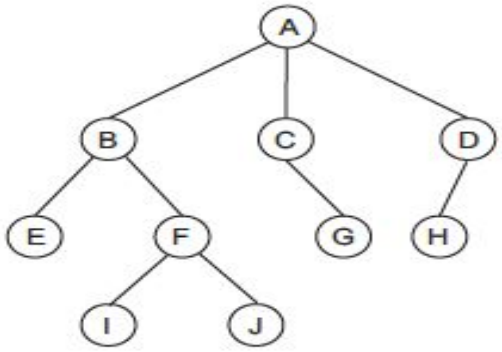

Figura 2. Árbol general

1. **Nivel:**

   De acuerdo a [3], es la distancia al nodo raíz. La raíz tiene una distancia cero de sí misma, por ello se dice que la raíz está en el nivel 0. Los hijos del raíz están en el nivel 1, sus hijos están en el nivel 2 y así sucesivamente. Una cosa importante que se aprecia

entre los niveles de nodos es la relación entre niveles y hermanos. Los hermanos están siempre al mismo nivel, pero no todos los nodos de un mismo nivel son necesariamente hermanos. Por ejemplo, en el nivel 2 (Figura 3), C y D son hermanos, al igual que lo son G, H, e I, pero D y G no son hermanos ya que ellos tienen diferentes padres.

2. **Camino:**

   [4] Jorge Santiago menciona que, es una secuencia de nodos en los que cada nodo es adyacente al siguiente. Cada nodo del árbol puede ser alcanzado (se llega a él) siguiendo un único camino que comienza en el raíz. En la Figura 3, el camino desde el raíz a la hoja I, se representa por AFI. Incluye dos ramas distintas AF y FI.

3. **Altura o profundidad:**

   Según [7], es el nivel de la hoja del camino más largo desde la raíz más uno. Por definición1 la altura de un árbol vacío es 0. La Figura 3 contiene nodos en tres niveles: 0, 1 y 2. Su altura es 3.

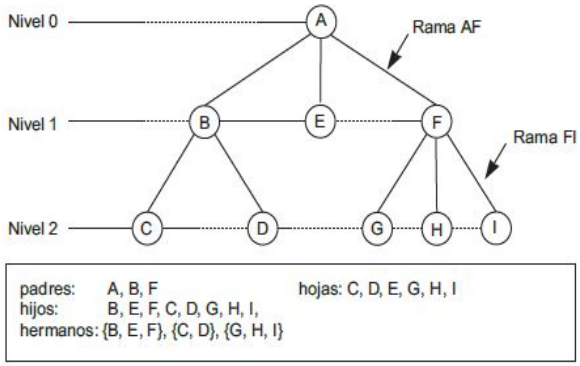

Figura 3. Terminología de árboles

III. Implementación de las Funciones para el Árbol Binario en C++

Para la implementación del Árbol Binario hemos considerado realizar un registro de datos personales de una persona, así que los siguientes códigos se basan en ello.

1. **Estructura Encabezado:**

   Para definir el encabezado, ver Figura 4, se usó **struct** para poder almacenar varios campos, en este caso **NRS** que es la variable que almacena el número total de registros, **RZ** que significa la raíz del árbol y **URE** que es la variable que almacena el último registro eliminado, además la variable **e**.

```
struct encabezado
{
    int NRS; //número de registros total
    int RZ; //raíz
    int URE; //último registro eliminado
}e;
```

Figura 4. Estructura Encabezado

2. **Estructura Person:**

   Para definir el registro, obsérvese en la Figura 5, se usó **struct** para poder almacenar varios campos, en este caso los datos personales de una persona y usamos: **NR** que es la variable que almacena el número de registro, el campo **nombre** que es la variable que almacena el nombre, el campo **apellido** que es la variable que almacena el apellido, el campo **celular** que es la variable que almacena el celular, el campo **email** que es la variable que almacena el correo electrónico, el campo **PI** que es la variable que almacena el puntero a la izquierda o rama izquierda del árbol, , el campo **PD** que es la variable que almacena el puntero a la derecha o rama derecha del árbol, el campo **ARE** que es la variable que almacena el anterior registro eliminado y además de los punteros **r**, **s** y **re max izq**.

```
struct person
{
    int NR; //número de registro
    char nombre[20];
    char apellido[20];
    char celular[10];
    char email[20];
    int PI; //Puntero izquierdo
    int PD; //Puntero derecho
    int ARE; //Anterior registro eliminado
}r,s,reg_max_izq;
```

Figura 5. Estructura person

3. **Escribir:**

```
void escribir()
{
    char rpta, lado; int sgte;

    if((fd=fopen("personas.txt","w+t")) != NULL)
    {
        e.NRS =0; e.RZ=-1; e.URE=-1;

        do{
            r.NR = ++e.NRS; //Actualiza la cant. de registros

            cin.ignore();

            cout<<" Nombre"<<endl;
                cin.getline(r.nombre,20);
            cout<<" Apellido"<<endl;
                cin.getline(r.apellido,20);
            cout<<" Celular"<<endl;
                cin.getline(r.celular,10);
            cout<<" Email"<<endl;
                cin.getline(r.email,20);

            r.ARE=0; r.PI=-1; r.PD=-1; //inicializa ARE, PI y PD

            pos=(r.NR-1)*lr + le;
            fseek(fd,pos,0);
            fwrite(&r,lr,1,fd);

            if(e.RZ==-1){
                e.RZ = r.NR;
            }
            else{
                //Inicia algoritmo buscar al registro
                sgte= e.RZ;

                while(sgte != -1){
                    pos=(sgte-1)*lr + le;
                    fseek(fd,pos,0);
                    fread(&s,lr,1,fd);

                    if(strcmp(r.nombre,s.nombre)>0)
                        {lado='d'; sgte=s.PD; continue;}
                    if(strcmp(r.nombre,s.nombre)<0)
                        {lado='i';sgte=s.PI; continue;}
                }
                //fin de algoritmo buscar

                if(lado=='d'){s.PD = r.NR;}
                if(lado=='i'){s.PI = r.NR;}

                fseek(fd,pos,0);
                fwrite(&s,lr,1,fd);
            }

            cout<<" - Desea guardar otro registro?"<<endl;
            cin>>rpta;
        } while(rpta == 's' || rpta =='S');

        fseek(fd,0,0); //actualiza cabecera
        fwrite(&e,le,1,fd);
        fclose(fd);
    }
    else
        cout<<" No se pudo crear el archivo!"<<endl;
}
```

Figura 6. Función Escribir

En la función escribir, obsérvese en la Figura 6, inicialmente se abre el archivo, es decir para ser escrito; además se inicializa la variable e en sus campos NRS, RZ y URE; que representa numero total de registros existentes, la rzia y el último registro eliminado respectivamente en el archivo.

Dentro de un do while, incrementa en uno el NR del registro con el valor de NRS; se solicita ingresar al usuario la información de los campos para cada registro. Luego inicializa la variable r en sus campos ARE, PI, PD con 0 y los dos últimos con -1, que representa que no apuntan a ningún registro. Posteriormente posiciona y escribe en el registro con **fseek y fwrite**.

Para el condicional, evalúa si la raíz es igual a -1, entonces la raiz es igual a número de registro. Por el contrario para el else, se emplea el algoritmo buscar al registro donde a la variable $sgte$ se le asigna el valor de la raiz, dentro del while calcula la posición de la variable sgte la posiciona y lee con fread. Además, evalúa donde irá almacenado, si al lado derecho o izquierdo. Por lo que, si el nombre ingresado es mayor irá al lado izquierdo de lo contrario al lado derecho. Luego actualiza los PD y PI de la variable s; según lo que la variable lado almacenó. Posiciona y escribe en el archivo.

Para el do while mencionado inicialmente, seguirá ejecutándose si rpta es S o s, de no cumplirse se termina. Finalmente se actualiza el encabezado y cierra del archivo.

4. **Mostrar:**

```
void mostrar()
{
    if((fd=fopen("personas.txt","rt")) != NULL)
    {
        fread(&e,le,1,fd);

        cout << "| MOSTRAR REGISTROS |"<<endl;
        cout<<endl<<"-----------------------------------------------"<<endl;
        cout<<" NRS:"<<e.NRS<<"\t"<<"Raiz:"<<e.RZ<<"\t"<<"URE:"<< e.URE<<endl;

        cout<<" NR"<<"\t"<<"Nombre"<<"\t"<<"Apellido"<<"  "<<"Celular"<<"\t"
            <<"Email"<<"\t\t"<<"PI"<<"\t"<<"PD"<<"\t"<<"ARE"<<endl;

        while(fread(&s,lr,1,fd))
        {
            cout<<endl<<" "<<s.NR<<"\t"<<s.nombre<<"\t"<<s.apellido<<"\t"
            <<s.celular<<"\t"<<s.email<<"\t\t"<<s.PI<<"\t"<<s.PD<<"\t"<<s.ARE;
        }
        cout<<endl<<"-----------------------------------------------"<<endl;
        fclose(fd);
    }
    else
        cout<<" No se pudo crear el archivo!"<<endl;
}
```

Figura 7. Función Mostrar

Para definir la función mostrar, ver Figura 7, primero, se abre el archivo $personas.txt$, luego, a través de **fread** se lee el encabezado para mostrar datos convenientemente y mediante un $while$, también se lee los datos a mostrar en pantalla, finalmente se cierra el archivo.

5. **Buscar:**

   En la función buscar, obsérvese en la Figura 8, inicialmente se abre el archivo $personas.txt$, luego, a través de **fread**, se lee el encabezado; luego se solicita digitar al usuario el nombre que desea buscar. Después asignamos a la variable sgte, la raíz del árbol. Para recorrer todos los registros e ir buscando el nombre solicitado usamos un $while$, se calcula la posición de $sgte$, se lee el archivo y a través de if, comparamos hasta que encontrar el nombre solicitado, mientras se va actualizando la variable $sgte$ a los punteros: izquierda o derecha, cuando se encuentra el nombre ingresado, se muestra los datos en el registro del árbol de esa persona. También se actualiza la variable $band$, de lo contrario se mostraría el mensaje: El nombre no existe.

```
void buscar()
{
    char nom[20], band='F'; int sgte;

    if((fd=fopen("personas.txt","rt")) != NULL)
    {
        fread(&e,le,1,fd);

        cout << "| BUSCAR |"<<endl;
        cin.ignore();
        cout<<" Nombre:"; cin.getline(nom,20);

        //algoritmo buscar al registro que apunta al nuevo
        sgte= e.RZ;

        while(sgte != -1)
        {
            pos=(sgte-1)*lr + le;
            fseek(fd,pos,0);
            fread(&s,lr,1,fd);

            if(strcmp(nom,s.nombre)>0)
                {sgte=s.PD; continue;}
            if(strcmp(nom,s.nombre)<0)
                {sgte=s.PI; continue;}

            cout<<endl<<"-----------------------------------------------"<<endl;
            cout<<endl<<" "<<s.NR<<"\t"<<s.nombre<<"\t"<<s.apellido<<"\t"
            <<s.celular<<"\t"<<s.email<<"\t\t"<<s.PI<<"\t"<<s.PD<<"\t"<<s.ARE;
            cout<<endl<<"-----------------------------------------------"<<endl;

            band='V'; break;
        }
        //fin de algoritmo buscar

        if(band=='F'){cout<<" El Nombre no existe!"<<endl;}
    }
    else
        cout<<" No se pudo crear el archivo!"<<endl;
    fclose(fd);
}
```

Figura 8. Función Buscar

6. **Insertar:**

   En esta función, ver Figura 9, es de manera similar que la función escribir, solo que aquí se **evita la inicialización del campo NRS, RZ, URE** en la variable e; porque esos son campos globales y puede existir la posibilidad que ya existan registros, por lo que si las inicializamos nuestro programa no funcionará y no almacenara los registros de manera adecuada, debido a que en ese campo se inicializa en 0. Luego todo es exactamente igual que la función escribir.

```
void insertar()
{
    char rpta, lado; int sgte;

    if((fd=fopen("personas.txt","r+t")) != NULL)
    {
        //e.NRS =0; e.RZ=-1; e.URE=-1;

        do{
            r.NR = ++e.NRS; //Actualiza la cant. de registros

            cout << "| INSERTAR |"<<endl;
            cin.ignore();

            cout<<" Nombre"<<endl;
                cin.getline(r.nombre,20);
            cout<<" Apellido"<<endl;
                cin.getline(r.apellido,20);
            cout<<" Celular"<<endl;
                cin.getline(r.celular,10);
            cout<<" Email"<<endl;
                cin.getline(r.email,20);

            r.ARE=0; r.PI=-1; r.PD=-1; //inicializa ARE, PI y PD

            pos=(r.NR-1)*lr + le;
            fseek(fd,pos,0);
            fwrite(&r,lr,1,fd);

            if(e.RZ==-1){
                e.RZ = r.NR;
            }
            else{
                //Inicia algoritmo buscar al registro
                sgte= e.RZ;

                while(sgte != -1){
                    pos=(sgte-1)*lr + le;
                    fseek(fd,pos,0);
                    fread(&s,lr,1,fd);

                    if(strcmp(r.nombre,s.nombre)>0)
                        {lado='d'; sgte=s.PD; continue;}
                    if(strcmp(r.nombre,s.nombre)<0)
                        {lado='i';sgte=s.PI; continue;}
                }
                //fin de algoritmo buscar

                if(lado=='d'){s.PD = r.NR;}
                if(lado=='i'){s.PI = r.NR;}

                fseek(fd,pos,0);
                fwrite(&s,lr,1,fd);
            }

            cout<<" - Desea guardar otro registro?"<<endl;
            cin>>rpta;
        } while(rpta == 's' || rpta =='S');

        fseek(fd,0,0);  //actualiza cabecera
        fwrite(&e,le,1,fd);
        fclose(fd);
    }
    else
        cout<<" No se pudo crear el archivo!"<<endl;
}
```

Figura 9. Función Insertar

7. **Modificar Celular:**

   En la función modificar, ver Figura 10, inicialmente se abre el archivo para ser modificado $r + t$, es decir para poder leer y escribir, **personas.txt**, luego con **fread**, se lee el encabezado y se solicita digitar al usuario el nombre que desea modificar.

   Después, a través del while como primer paso, se **busca en los registros** realizándose comparaciones, actualizan-

do a su vez los punteros: izquierda y derecha, de encontrarse se realiza el segundo paso donde **solicita digitar el nuevo Celular**, actualizando dicha información a través de posicionar y escribir, con **fseek y fwrite** en el registro, por otro lado, de no encontrarse el nombre muestra un mensaje. Finalmente actualiza el encabezado, y cierra el archivo.

```
void modificar_celular()
{
    char nom[20], cel[10], band='F'; int sgte, pos1=0;

    if((fd=fopen("personas.txt","r+t")) != NULL)
    {
        fread(&e,le,1,fd);

        cout << "| MODIFICAR CELULAR |"<<endl;
        cin.ignore();
        cout<<" Nombre a modificar:"; cin.getline(nom,20);

        //algoritmo busca al registro
        sgte= e.RZ;

        while(sgte != -1)
        {
            pos=(sgte-1)*lr + le;
            fseek(fd,pos,0);
            fread(&s,lr,1,fd);

            if(strcmp(nom,s.nombre)>0)
                {sgte=s.PD; continue;}
            if(strcmp(nom,s.nombre)<0)
                {sgte=s.PI; continue;}

            cout<<" Digite el Nuevo celular:"; //Modifica el nuevo celular
            cin.getline(s.celular,10);

            fseek(fd,pos,0); //Posiciona y escribe en el archivo
            fwrite(&s,lr,1,fd);

            cout<<" ¡El Registro fue modificado! "<<endl;
            band='V'; break;
        }
        //fin de algoritmo buscar

        if(band=='F'){cout<<" El Nombre no existe!"<<endl;}
    }
    else
        cout<<" No se pudo crear el archivo!"<<endl;

    fseek(fd,0,0); //actualiza cabecera
    fwrite(&e,le,1,fd);
    fclose(fd);
}
```

Figura 10. Función Modificar

8. **Eliminar:**

   Para la función eliminar, obsérvese en la Figura 11, se inicializan las variables locales que usará esta función; inicialmente se abre el archivo **personas.txt**, luego con fread, se lee el encabezado y se solicita digitar al usuario el nombre que desea eliminar.

   Después, a través del while como paso 0, se **busca en los registros** realizándose comparaciones con el if, de encontrarse, actualiza la band con true.

   Y se evalúa para dos opciones si tiene máximo izquierdo y si no tiene máximo izquierdo.

```
void eliminar()
{
    int sgte, e_PI, e_PD, pos_3,pos_2, pos_1, number, max_izq;
    bool band_1=false, band_2=false, band,flag=false;
    char name [20], ape[20], cel[10], em[20], nom_eliminado[20];

    cout << "| ELIMINAR |";

    if((fd=fopen("personas.txt","r+t"))==NULL) //abre archivo
    {
        cout<<" No se pudo abrir el archivo!"<<endl;
        return;
    }

    fread(&e,le,1,fd); //Lee cabecera

    cin.ignore();
    cout<<endl<<" Digite el nombre a eliminar: ";
    cin.getline(nom_eliminado,20);

    //0.Buscar el registro a eliminar
    while(fread(&r,lr,1,fd))
    {
        if(strcmp(nom_eliminado,r.nombre)==0)
        {
            flag=true;

            ///Si tiene máx izquierdo
            if(r.PI != -1)
            {
                band_1=false, band_2=false; max_izq=r.PI;

                while(max_izq != - 1)
                {
                    pos = (max_izq -1)*lr + le;
                    fseek(fd,pos,0);
                    fread(&reg_max_izq,lr,1,fd);
                    max_izq = reg_max_izq.PD;
                }

                cout<<" Máximo izquierdo:"<<reg_max_izq.nombre<<endl;

                //Intercambia claves - campos del eliminado a su max.izq
                strcpy(name, r.nombre);
                strcpy(r.nombre,reg_max_izq.nombre);
                strcpy(ape, r.apellido);
                strcpy(r.apellido,reg_max_izq.apellido);
                strcpy(cel, r.celular);
                strcpy(r.celular,reg_max_izq.celular);
                strcpy(em, r.email);
                strcpy(r.email,reg_max_izq.email);

                //actualiza, pos y escribe en el archivo
                pos_1=(r.NR-1)*lr + le;
                fseek(fd,pos_1,0);
                fwrite(&r,lr,1,fd);
```

Figura 11. Función Eliminar - Primera parte

Para el primer caso realiza la búsqueda con el while del máximo izquierdo, cuando lo encuentra intercambia las claves.

Obsérvese en la Figura 12, aquí como se consideraron varios campos, se almacena en las variables name, ape, cel y em, la información del que será eliminado.

Luego reemplaza los campos del eliminado con los del max.izq. Actualiza y escribe en el archivo lo realizado. Continua el intercambio de claves pero ahora los campos del máximo izquierdo a los campos del eliminado. Mediante un while recorre todo el registro y actualiza los campos. Además, actualiza el URE y ARE, almacenando los valores de los punteros PD y PI.

En el siguiente while según el NR a eliminar se actualizan mediante condicionales los valores anteriormente almacenados de PI y PD.

```
        //Intercambia claves - campos del max.izq al eliminado
        sgte= e.RZ;
        while(sgte <= e.NRS)
        {
            pos_2=(sgte-1)*lr + le;
            fseek(fd,pos_2,0);
            fread(&r,lr,1,fd);

            if (reg_max_izq.NR == r.NR)
            {
             strcpy(r.nombre,name);
             strcpy(r.apellido,ape);
             strcpy(r.celular,cel);
             strcpy(r.email,em);
             //2. actualizar URE/ARE
             r.ARE = e.URE;
             e.URE = r.NR;

             //almecena var. auxi punteros PI y PD
             number= r.NR;
             e_PD = r.PD;
             e_PI = r.PI;

             fseek(fd,pos_2,0);
             fwrite(&r,lr,1,fd);
             break;
            }

            sgte++;
        }

        //---------------------------------------

        //Busca el NR en el PI y PD
        sgte= e.RZ;
        while(sgte <= e.NRS)
        {
            pos_3=(sgte-1)*lr + le;
            fseek(fd,pos_3,0);
            fread(&r,lr,1,fd);

            if (number == r.PI && band_1==false) //Actualiza PI
            {
                r.PI = e_PD;
                fseek(fd,pos_3,0);
                fwrite(&r,lr,1,fd);

                band_1 = true;
            }

            if (number == r.PD && band_2==false) //Actualiza PD
            {
                r.PD = e_PI;
                fseek(fd,pos_3,0);
                fwrite(&r,lr,1,fd);

                band_2=true;
            }

            if (band_1==true && band_2==true) {break;}
            sgte++;
        }
    }
```

Figura 12. Función Eliminar - Segunda parte

```
        ///Si no tiene máx izquierdo
        else
        {
            //1. Asignación a variables auxiliares
            number = r.NR;
            e_PI = r.PI;
            e_PD = r.PD;

            //2. Actualizar URE/ARE y escribe
            r.ARE = e.URE;
            e.URE = r.NR;

            pos_1=(r.NR-1)*lr + le;
            fseek(fd,pos_1,0);
            fwrite(&r,lr,1,fd);

            //Busca el NR
            sgte= e.RZ;
            while(sgte <= e.NRS)
            {
                pos_2=(sgte-1)*lr + le;
                fseek(fd,pos_2,0);
                fread(&r,lr,1,fd);

                if (number == r.PI && band_1==false)  //Actualiza PI
                {
                    r.PI = e_PD;
                    fseek(fd,pos_2,0);
                    fwrite(&r,lr,1,fd);

                    band_1 = true;
                }

                if (number == r.PD && band_2==false)  //Actualiza PD
                {
                    r.PD = e_PI;
                    fseek(fd,pos_2,0);
                    fwrite(&r,lr,1,fd);

                    band_2=true;
                }

                if (band_1==true && band_2==true) {break;}
                sgte++;
            }
        }
        cout<<" ¡El Registro fue eliminado! "<<endl;
        break;
    }
}

if(flag==false){
    cout<<" El nombre no existe!"<<endl;
    return;
}

fseek(fd,0,0);  //actualiza cabecera
fwrite(&e,le,1,fd);
fclose(fd);
}
```

Figura 13. Función Eliminar - Tercera parte

Para el segundo caso, donde no tiene máximo izquierdo, se asigna a variables auxiliares los valores de NR, PI y PD. Asimismo actualiza los valores de URE/ARE. Posicionando y escribiendo en el archivo.

Y el siguiente while según el NR a eliminar se actualizan mediante condicionales los valores anteriormente almacenados de PI y PD.

De cumplirse uno de los dos casos, muestra registro eliminado. Por otro lado, de no encontrarse el nombre muestra un mensaje. Finalmente posiciona y escribe con **fseek y fwrite** actualiza el encabezado, y cierra el archivo.

9. **Ver Eliminados:**

   Para definir la función Ver Eliminados, ver Figura 14, primero, se abre el archivo $personas.txt$, luego, a través de **fread** se lee el encabezado para mostrar datos convenientemente, posteriormente asignamos a la variable $sgte$ el URE y mediante un $while$ se recorre todo el árbol, actualizamos la pos a través de **fseek** y se lee a través de **fread**, finalmente se lee los datos a mostrar en pantalla y se actualiza el ARE en $sgte$, para terminar se cierra el archivo.

```
void ver_eliminados() //Muestra los registros eliminados
{
    int sgte,pos;

    if((fd=fopen("personas.txt","rt"))==NULL)
    {
        cout<<"No se pudo abrir el archivo"<<endl;
        return;
    }

    fread(&e,le,1,fd);
    cout<<endl<<"-------------------------------------------------"<<endl;
    cout<<" NRS:"<<e.NRS<<"\t"<<"Raiz:"<<e.RZ<<"\t"<<"URE:"<< e.URE<<endl;

    cout<<" NR"<<"\t"<<"Nombre"<<"\t"<<"Apellido"<<"  "<<"Celular"<<"\t"
        <<"Email"<<"\t\t"<<"PI"<<"\t"<<"PD"<<"\t"<<"ARE"<<endl;

    sgte = e.URE;

    while(sgte != -1) //aquí se recorre todo el archivo
    {
        pos = (sgte-1)*lr + le; //aquí se actualiza la pos y avanza
        fseek(fd,pos,0);
        fread(&s,lr,1,fd);

        cout<<endl<<" "<<s.NR<<"\t"<<s.nombre<<"\t"<<s.apellido<<"\t"
        <<s.celular<<"\t"<<s.email<<"\t\t"<<s.PI<<"\t"<<s.PD<<"\t"<<s.ARE;

        sgte = s.ARE;
    }
    cout<<endl<<"-------------------------------------------"<<endl;
    fclose(fd);
}
```

Figura 14. Ver eliminados

10. **Máximo Izquierdo:**

Para definir la función Máximo Izquierdo, ver Figura 15, primero, se abre el archivo $personas.txt$, luego, a través de **fread** se lee el encabezado. Luego, se pide al usuario digitar el nombre a buscar si tiene Máximo Izquierdo. Posteriormente asignamos a la variable $sgte$ el RZ y mediante un $while$ se recorre todo el árbol, actualizamos la pos a través de **fseek** y se lee a través de **fread**, para poder realizar las comparaciones usando $strcmp$, a su vez se va actualizando el valor de $sgte$ con los punteros: izquierdo y derecho, aquí se termina la búsqueda del nombre ingresado. Finalmente se calcula el Máximo Izquierdo, a través de un $while$ se recorre todo el árbol, se actualiza la pos y se leen datos. Para terminar, se muestra en pantalla el máximo izquierdo del nombre ingresado, de lo contrario que, el nombre no existe y se cierra el archivo.

```
void max_izq()
{
    char nom[20], band='F'; int sgte, max_izq;

    if((fd=fopen("personas.txt","rt")) != NULL)
    {
        fread(&e,le,1,fd);

        cout << "| MAX. IZQ |"<<endl;
        cin.ignore();
        cout<<" Nombre:";  cin.getline(nom,20);

        //algoritmo buscar al registro
        sgte= e.RZ;

        while(sgte != -1)
        {
            pos=(sgte-1)*lr + le;
            fseek(fd,pos,0);
            fread(&s,lr,1,fd);

            if(strcmp(nom,s.nombre)>0)
                {sgte=s.PD; continue;}
            if(strcmp(nom,s.nombre)<0)
                {sgte=s.PI; continue;}

            band = 'V'; //se encontró la clave, ahora busca su max_izquierdo

            if(s.PI != -1)
            {
                //existe un maximo izquierdo
                max_izq = s.PI;

                while(max_izq != - 1)
                {
                    pos = (max_izq -1)*lr + le;
                    fseek(fd,pos,0);
                    fread(&reg_max_izq,lr,1,fd);
                    max_izq = reg_max_izq.PD;
                }

                cout<<endl<<"-----------------------------------------------"
                <<endl;
                cout<<" Máximo izquierdo:"<<reg_max_izq.nombre;
                cout<<endl<<"-----------------------------------------------"
                <<endl;
            }
            else
                cout<<" No tiene maximo izquierdo"<<endl;
            //fin de busqueda de maximo izquierdo

            break;
        }

        //fin de algoritmo buscar
        if(band=='F'){cout<<" El Nombre no existe!"<<endl;}
    }
    else
        cout<<" No se pudo abrir el archivo!"<<endl;
    fclose(fd);
}
```

Figura 15. Función Máximo Izquierdo

IV. CONCLUSIONES

Este informe presentó información relevante acerca de árboles binarios, se ha explicado los conceptos teóricos relacionados con este, cabe precisar que las ramas del árbol está representada por los campos PI y PD; las cuáles siempre tendrán un valor, pues son la base de la creación del árbol. Además de permitir conocer sobre la implementaciones de árboles binarios, con sus respectivas explicaciones, además de otras funciones que permiten la creación de registros y archivos, con extensión $*.txt.$ en el que se puede escribir, mostrar, buscar, modificar, eliminar y ver registros eliminados, además encontrar el máximo izquierdo de un registro.

REFERENCIAS

[1] Árbol binario - wikipedia, la enciclopedia libre. (Accessed on 01/29/2022).

[2] O. Cairó and S. Guardati. *Estructuras de datos*. MC GRAW HILL INTERAMERICANA, 2006.

[3] A. H. González. Árboles. 2013.

[4] S. Jorge. Estructuras de datos: Arboles binarios de busqueda, monticulos.

[5] L. Joyanes, L. Joyanes, and I. Zahonero. *Estructuras de datos en C++*. McGraw-Hill/Interamericana de España, 2007.

[6] L. R. Nyhoff, M. P. Tarjuelo, C. S. Díaz, J. A. V. López, and N. M. Olliet. *TADs, Estructuras de datos y resolución de problemas con C++*. Pearson Prentice Hall, 2006.

[7] J. Peral Cortés, A. Ferrández, S. Luján-Mora, and A. Requena Jiménez. Programación y estructuras de datos (curso 2006-2007). *Programación y Estructuras de Datos*, 2006.