IC2001 Estructuras de Datos

Grupo 2

Jose Pablo Agüero Mora 2021126372

Katerine Guzmán Flores 2019390523

Manual Técnico – Proyecto 1B

I Semestre 2022

Índice

Manual técnico

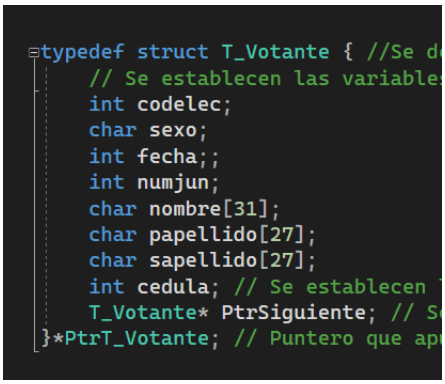
Descripción:

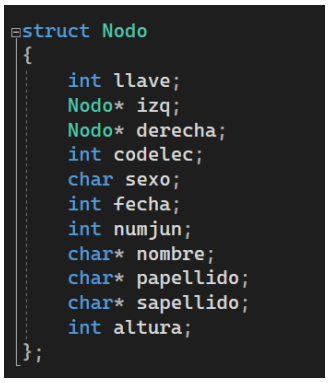
Este proyecto constituye un programa que recibe un archivo de texto que corresponde al padrón electoral de Costa Rica y carga todos los datos en una lista enlazada maestra. El objetivo de este proyecto es introducir los datos del archivo en cuatro diferentes estructuras: Lista Enlazada, Árbol Binario de Búsqueda, Árbol Btree y Árbol AVL. Una vez que se han cargado se prosigue a determinar el tiempo que dura insertando todos los nodos.

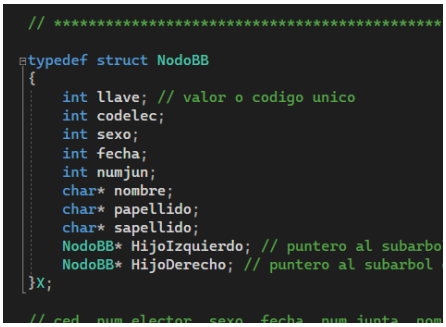
Posteriormente a que se ingresan los nodos, el programa también puede listar todos los votantes en consola que se encuentran en el archivo, según la función correspondiente. Además, otra de las funciones de este programa es buscar un votante y desplegar toda la información personal si es que esta dentro del archivo. Una vez introducidos todos los datos en cada una de las estructuras también el programa tiene la opción de liberar todas las estructuras eliminando uno a uno los nodos. Finalmente, el programa muestra en su menú una cuarta función que despliega todos los tiempos de ejecución de las operaciones básicas de las estructuras.

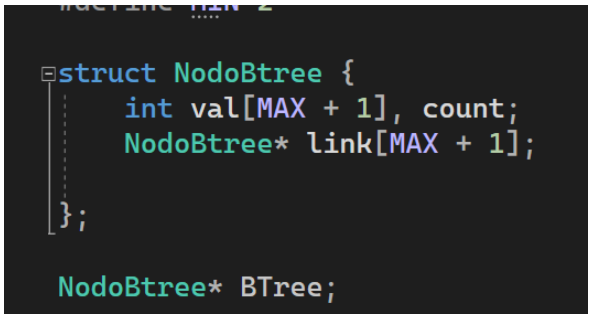
Estructuras utilizadas:

Se utilizan las 4 estructuras correspondientes a: Lista Enlazada, Árbol Binario de Búsqueda, Árbol Btree y árbol AVL. Los campos de estas estructuras vana contener los mismos campos excepto el Btree que solo posee la cedula.









Menú:

Al inicio en el main, se van a inicializar la mayoría de las estructuras para poder luego ser utilizadas. Se carga el padrón y además se ingresan los datos en cada una de las estructuras.

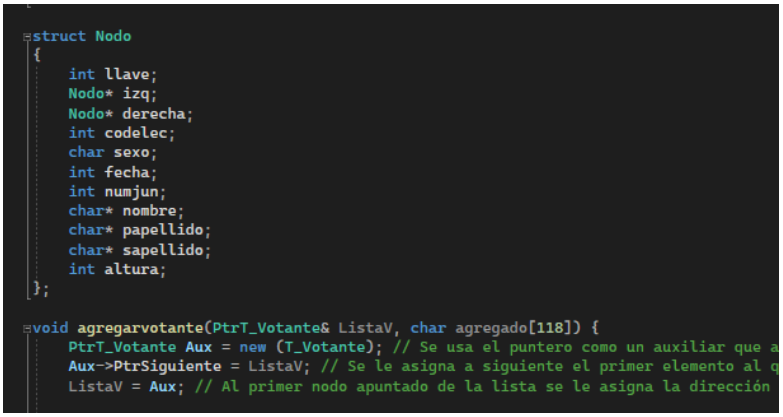
```
int main()
{// Inicializacion de los arboles
    PtrT_Votante ListaV = NULL;
    PtrT_Votante Encontrado = NULL;
    NodoBB* ArbolBinario = NULL;

    // ************************************************* Cargar Padron de Votantes ************************************************************
    cout << "Por favor espere mientras se carga el Padron" << endl;
    time_t inicioC, finC; // Se declaran variables de tipo tiempo para medir la duración del proceso
    inicioC = time(NULL); // Declaración de variable para medir el tiempo inicial
    CargarVotantes(ListaV);
    finC = time(NULL);
    cout << "-------------------------------------------------------------Padron Cargado-------------------------------------------------------------" << endl;
    printf("\nCargar el padron ha tardado : %f segundos.\n", difftime(finC, inicioC));
    tiempo1 = difftime(finC, inicioC);
    printf("\n ");
    printf("\n ");

    // ***************************** Insertar ABB *********************************
    PtrT_Votante Aux1 = ListaV;
    time_t ABIinicio, ABIfin;
    ABIinicio = time(NULL);
    InsertarTodos(Aux1, ArbolBinario);
    ABIfin = time(NULL);
    cout << "-------------------------------------------------Padron Insertado en Arbol ABB-------------------------------------------------" << endl;
    printf("\nLa insercion del padron en Arbol ABB ha tardado : %f segundos.\n", difftime(ABIfin, ABIinicio));
    tiempo2 = difftime(ABIfin, ABIinicio);
    printf("\n ");
    printf("\n ");


    // ******************************** Insertar BTree ************************
    PtrT_Votante Aux2 = ListaV;
    time_t inicioBBT, finBBT;
    inicioBBT = time(NULL);
    BTreeInsertarTodos(Aux2, BTree);
    finBBT = time(NULL);
    cout << "-----------------------------------------------Padron Insertado en Arbol BTree-----------------------------------------------" << endl;
    printf("\nLa insercion del padron en Arbol BTree ha tardado : %f segundos.\n", difftime(finBBT, inicioBBT));
    tiempo3 = difftime(finBBT, inicioBBT);
    printf("\n ");
    printf("\n ");

    // ***************************** Insertar en Arbol AVL ************************
```

Para el menú se utiliza un switch con respecto a cada case que corresponde a la opción que el usuario ingresa de 1 a 5. De esta forma se van llamando cada una de las operaciones según el numero ingresado y mediante un break permite volver al menú para solicitar otra opción una vez terminado ese case.

```
int val, opt, codi;
char cedula[10];

while (true) {
    system("CLS");

    printf(" Padron Electoral ");
    printf("\n");

    cout << " 1. Buscar\n";
    cout << " 2. Listar\n";
    cout << " 3. Liberar\n";
    cout << " 4. Reporte de Tiempos \n";
    cout << " 5. Salir \n\n";
    cout << " Ingrese una opcion: ";

    cin >> opt;
    cout << endl;
```

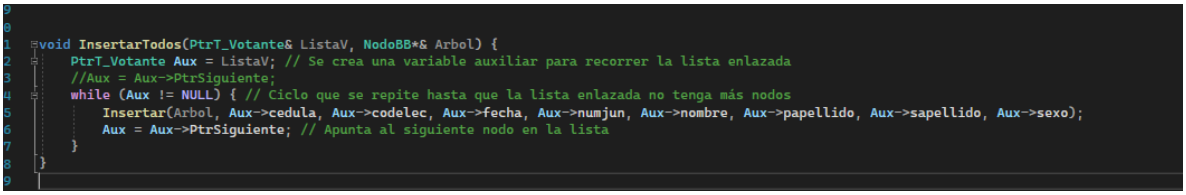
Insertar

Para el caso de insertar en la lista enlazada se hace un llamado desde la función insertar para que vaya dividiendo los char del archivo original y los vaya introduciendo en la estructura de tipo votante.



```
struct Nodo
{
    int llave;
    Nodo* izq;
    Nodo* derecha;
    int codelec;
    char sexo;
    int fecha;
    int numjun;
    char* nombre;
    char* papellido;
    char* sapellido;
    int altura;
};

void agregarvotante(PtrT_Votante& ListaV, char agregado[118]) {
    PtrT_Votante Aux = new (T_Votante); // Se usa el puntero como un auxiliar que a
    Aux->PtrSiguiente = ListaV; // Se le asigna a siguiente el primer elemento al q
    ListaV = Aux; // Al primer nodo apuntado de la lista se le asigna la dirección
```

Insertar en el caso del árbol binario de búsqueda, funciona como un insertar masivo ya que va tomando cada uno de los nodos de la lista enlazada y los va metiendo en el árbol con la función de insertar común de esta estructura, donde sino esta lo inserta ya sea si es menor a la izquierda o mayor a la derecha.



```
void InsertarTodos(PtrT_Votante& ListaV, NodoBB*& Arbol) {
    PtrT_Votante Aux = ListaV; // Se crea una variable auxiliar para recorrer la lista enlazada
    //Aux = Aux->PtrSiguiente;
    while (Aux != NULL) { // Ciclo que se repite hasta que la lista enlazada no tenga más nodos
        Insertar(Arbol, Aux->cedula, Aux->codelec, Aux->fecha, Aux->numjun, Aux->nombre, Aux->papellido, Aux->sapellido, Aux->sexo);
        Aux = Aux->PtrSiguiente; // Apunta al siguiente nodo en la lista
    }
}
```

En el caso del insertar en el Btree funciona parecido en relaciona al insertar masivo que se hace en el caso del árbol binario de búsqueda para que de esta forma vaya insertando cada uno de los nodos. Sin embargo, en esta estructura se llaman otras funciones ya que este paso es más complejo ya que se debe ingresar

en una función aparte el valor de la llave del nodo a ingresar, y además dentro de esta función se debe tomar en cuenta donde colocar el nodo si no ha alcanzado el máximo, o el valor de m y sino debe llamar a otra función para ir partiendo los nodos.

```
//inserta valores en el btree
void BTreeinsertar(PtrT_Votante& ListaV, int val) {
    int flag, i;
    NodoBtree* hijo;

    flag = SetValorNodo(val, &i, BTree, &hijo);
    if (flag)
        BTree = crearnodo(i, hijo);
}

void BTreeInsertarTodos(PtrT_Votante& ListaV, NodoBtree*& Arbol) {
    PtrT_Votante Aux = ListaV; // Se crea una variable auxiliar para recorrer la lista enlazada

    Aux = Aux->PtrSiguiente;
    while (Aux != NULL) { // Ciclo que se repite hasta que la lista enlazada no tenga más nodos
        BTreeinsertar(ListaV, Aux->cedula);
        Aux = Aux->PtrSiguiente; // Apunta al siguiente nodo en la lista
    }
}
```

```
//Crear nodo
NodoBtree* crearnodo(int val, NodoBtree* hijo) {
    NodoBtree* NuevoNodo = new NodoBtree;
    NuevoNodo->val[1] = val;
    NuevoNodo->count = 1;
    NuevoNodo->link[0] = BTree;
    NuevoNodo->link[1] = hijo;
    return NuevoNodo;
}

//coloca el nodo en la posicion adecuada, de acuerdo a su valor
void Colocarnodo(int val, int pos, NodoBtree* node, NodoBtree* hijo) {
    int j = node->count;
    while (j > pos) {
        node->val[j + 1] = node->val[j];
        node->link[j + 1] = node->link[j];
        j--;
    }
    node->val[j + 1] = val;
    node->link[j + 1] = hijo;
    node->count++;
}
```

Para el caso de insertar en el árbol AVL, se hace un insertar masivo nada más que la función insertar devuelve un nodo que se vuelve a llamar en cada interacción hasta que un auxiliar de tipo ListaV haya leído por completo la lista enlazada y de esa forma ir ingresando cada nodo al mismo árbol.

```
// *************************** Insertar en Arbol AVL ************************

time_t inicio, fin;
inicio = time(NULL);
struct Nodo* root = NULL;
PtrT_Votante Aux3 = ListaV;
Aux3 = Aux3->PtrSiguiente;


while (Aux3 != NULL) {
    root = insertarAVL(root, Aux3->cedula, Aux3->codelec, Aux3->fecha, Aux3->numjun, Aux3->nombre, Aux3->papellido, Aux3->sapellido, Aux3->sexo);
    Aux3 = Aux3->PtrSiguiente;

}
```

Buscar:

En el caso de la lista enlazada, se realiza una función buscar que lo permite es ir buscando el nodo en todo el árbol y si lo encuentra me devuelve el nodo.

```
PtrT_Votante BuscarVotante(PtrT_Votante& Lista, int cual)
{
    bool encontro = false; // Se establece un flag booleano en false
    PtrT_Votante Aux; // Se inicializa un puntero
    Aux = Lista; // Se le asigna la dirección del primer elemento en la lista

    while ((!encontro == true) && (Aux != NULL)) // Ciclo que recorre toda la lista hasta que encuentre el elemento o termine la lista
    {
        if (Aux->cedula == cual) // Condición en donde se encuentra el elemento
            encontro = true; // Cambia la variable flag a true
        else
            Aux = Aux->PtrSiguiente; // El otro caso donde no lo encuentra en el nodo actual entonces pasa a buscar al siguiente
    }
    return Aux; // Retorna la dirección del último nodo analizado, es decir, donde se encontró el elemento buscado
}
```

Se utilizan pasos auxiliares para poder desplegar la información del votante que se ha encontrado o imprimir que no se ha encontrado en caso contrario.

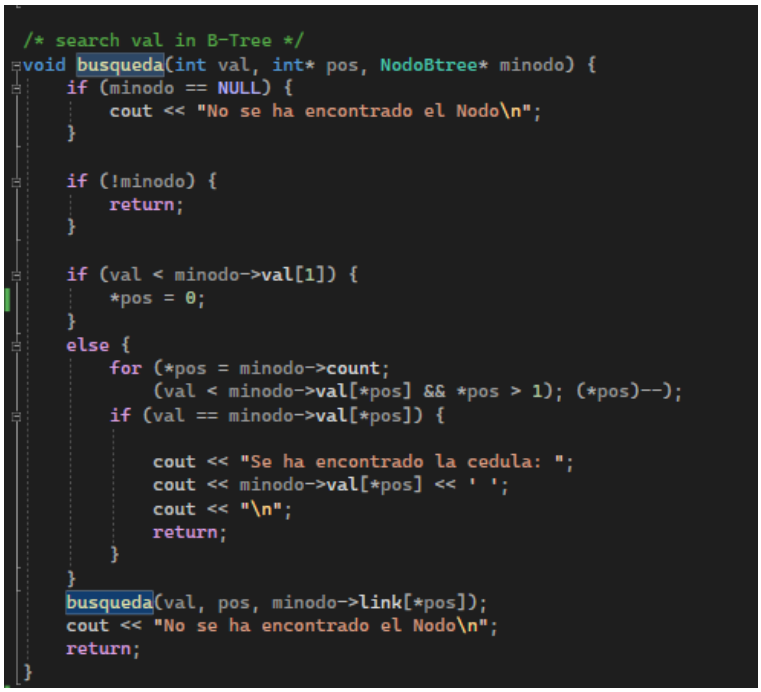
```
Encontrado = BuscarVotante(ListaV, val);
if (Encontrado != NULL) {
    cout << endl << "Encontre al Votante!! " << endl << endl;

    cout << "Cedula: " << Encontrado->cedula << endl;
    cout << "Nombre: " << Encontrado->nombre << endl;
    cout << "Primer Apellido: " << Encontrado->papellido << endl;
    cout << "Segundo Apellido: " << Encontrado->sapellido << endl;
    cout << "Sexo: " << Encontrado->sexo << endl;
    cout << "Fecha: " << Encontrado->fecha << endl;
    cout << "Codigo Electoral:  " << Encontrado->codelec << endl;
    cout << "Numero Junta: " << Encontrado->numjun << endl;
}
else
    cout << " No se encontro al votante !!! " << endl;
printf("\n ");
```

En el caso de buscar de un árbol AVL, lo que se hace es llamar a una función llamada buscarCedulaAVL quien recibe el árbol ya inicializado y la llave a buscar. Si al recorrer todo el árbol no lo encuentra retorna null y un mensaje de no encontrado. Si por lo contrario si lo encuentra al recorrer todo el árbol envía el mensaje correspondiente.

```
Nodo* BuscaCedulaAVL(Nodo* Raiz, int llave)
{
    time_t inicio, fin; // Se declaran variables de tipo tiempo para medir la duración del proceso
    inicio = time(NULL); // Declaración de variable para medir el tiempo inicial
    if (Raiz == NULL)
    {
        cout << "No se ha encontrado al votante " << endl;
        return NULL;
    }
    else
    {
        if (llave == Raiz->llave) {
            cout << "Cedula: " << Raiz->llave << endl; // Imprime los valores de los nodos
            cout << "Nombre: " << Raiz->nombre << endl;
            cout << "Primer apellido: " << Raiz->papellido << endl;
            cout << "Segundo apellido: " << Raiz->sapellido << endl;
            cout << "Sexo: " << Raiz->sexo << endl;
            cout << "Fecha: " << Raiz->fecha << endl;
            cout << "Codigo electorial: " << Raiz->codelec << endl;
            cout << "Numero junta: " << Raiz->numjun << endl;
            return Raiz;
        }
        else
            if (llave < Raiz->llave)
                return BuscaCedulaAVL(Raiz->izq, llave);
            else
                return BuscaCedulaAVL(Raiz->derecha, llave);
    }
    cout << "-----------------------------------------------------Votante Encontrado en Arbol AVL-------------
    printf("\nSe ha tardado : %f segundos.\n", difftime(fin, inicio));
}
```

Para buscar en el árbol Btree lo que se hace es insertar un valor que representa nuevamente la llave buscar, si el nodo retorna null se envía el mensaje a consola

que no se encontró y si lo encuentra me retorna la llave del nodo y un mensaje de encontrado.



```
/* search val in B-Tree */
void busqueda(int val, int* pos, NodoBtree* minodo) {
    if (minodo == NULL) {
        cout << "No se ha encontrado el Nodo\n";
    }

    if (!minodo) {
        return;
    }

    if (val < minodo->val[1]) {
        *pos = 0;
    }
    else {
        for (*pos = minodo->count;
            (val < minodo->val[*pos] && *pos > 1); (*pos)--);
        if (val == minodo->val[*pos]) {

            cout << "Se ha encontrado la cedula: ";
            cout << minodo->val[*pos] << ' ';
            cout << "\n";
            return;
        }
    }
    busqueda(val, pos, minodo->link[*pos]);
    cout << "No se ha encontrado el Nodo\n";
    return;
}
```

Listar:

En el caso de listar para la lista enlazada lo que va haciendo es un recorrido mediante un auxiliar hasta que llegue antes de ser null y de esta forma va imprimiendo la información de cada votante hasta terminar la lista. Si la lista ya se ha liberado antes mediante la opción 3 de menú, esta envía un mensaje de que la lista ya se ha liberado.

```
void ListarInventario(PtrT_Votante& ListaV)
{
    int Contador = 1;
    PtrT_Votante Aux; // variable local que servira para ir saltando de nodo en nodo
    Aux = ListaV; // Apunta donde apunta lista(primer elemento)
    if (Aux != NULL) {
        Aux = Aux->PtrSiguiente;
        while (Aux != NULL) // pregunta si se salio de la lista o esta vacia
        {
            // se imprime la informacion de cada pieza
            printf("\nCedula %i", Aux->cedula);
            printf("\nNombre %s", Aux->nombre);
            printf("\nPrimer Apellidoe %s", Aux->papellido);
            printf("\nSegundo Apellido %s", Aux->sapellido);
            printf("\nSexo");
            printf("\nCodigo elector %i", Aux->codelec);
            printf("\nFecha %i", Aux->fecha);
            printf("\nNumero de Junta %i", Aux->numjun);
            printf("\n");
            Aux = Aux->PtrSiguiente; // se pasa al siguiente nodo
            Contador++;
        }
    }

    else
    {
        printf("La lista se ha liberado");
        exit(0);
    }
}
```

En el caso de listar o desplegar para un árbol AVL, lo que se hace es un proceso similar al de la lista enlazada, pero se usa recursividad para ir llamando la función y busque ya sea en los hijos de la izquierda o de la derecha.

```
//recorido
void preOrder(struct Nodo* Raiz)
{
    if (Raiz != NULL)
    {
        cout << "Cedula: " << Raiz->llave << endl; // Imprime los valores de los nodos
        cout << "Nombre: " << Raiz->nombre << endl;
        cout << "Primer apellido: " << Raiz->papellido << endl;
        cout << "Segundo apellido: " << Raiz->sapellido << endl;
        cout << "Sexo: " << endl;
        cout << "Fecha: " << Raiz->fecha << endl;
        cout << "Codigo electorial: " << Raiz->codelec << endl;
        cout << "Numero junta: " << Raiz->numjun << endl;
        printf("\n ");
        preOrder(Raiz->izq);
        preOrder(Raiz->derecha);
    }

}
```

Para listar el padrón en un árbol binario de búsqueda lo que hace es utilizar el recorrido de en orden, es decir, ir primero a la izquierda, la raíz y a la derecha para irlos imprimiendo en este orden cada uno de los nodos con la información

correspondiente y nuevamente ir llamando de forma recursiva la función para que se vaya a buscar por ambos posibles hijos ya sea si es menor o mayor.

```
void EnOrdenIRD(NodoBB* Raiz) // I R D
{ // si sale del rango ya empieza a retornar

    if (Raiz != NULL) // caso continuidad
    {
        EnOrdenIRD(Raiz->HijoIzquierdo);
        cout << "Cedula: " << Raiz->llave << endl; // Imprime los valores de los nodos
        cout << "Nombre: " << Raiz->nombre << endl;
        cout << "Primer apellido: " << Raiz->papellido << endl;
        cout << "Segundo apellido: " << Raiz->sapellido << endl;
        cout << "Sexo: " << endl;
        cout << "Fecha: " << Raiz->fecha << endl;
        cout << "Codigo electorial: " << Raiz->codelec << endl;
        cout << "Numero junta: " << Raiz->numjun << endl;
        printf("\n ");
        EnOrdenIRD(Raiz->HijoDerecho);
    }
}
```

Para el caso de listar en un Btree, se van recorriendo todos los nodos en un for hasta que llegue al limite que es count, y llama recursivamente a desplegar para ir imprimiendo la cedula del votante según el valor que posea el nodo.

```
// B-Tree desplegar
void desplegar(NodoBtree* minodo) {
    int i;
    if (minodo) {
        for (i = 0; i < minodo->count; i++) {
            desplegar(minodo->link[i]);
            cout << minodo->val[i + 1] << ' ';
            cout << "\n";
        }
        desplegar(minodo->link[i]);
    }
}
```

Liberar:

Liberar en el caso de la lista enlazada, funciona como listar, pero en vez de imprimir lo que hace es ir eliminando el nodo en memoria y además apuntar a null cada nodo para que se libere la referencia.

```
//Funcion para liberar los votantes en estructuras de memoria dinamica de la lista enlazada hasta dejar la lista en NULL

void LiberarVotantes(PtrT_Votante& ListaV) {
    PtrT_Votante Aux = ListaV;
    ListaV = ListaV->PtrSiguiente;

    while (Aux != NULL) {
        ListaV = ListaV->PtrSiguiente;
        printf("\nBorro votante con cedula %i", Aux->cedula);
        delete(Aux);
        Aux = ListaV;
    }
}
```

En el caso del AVL, funciona muy similar ya que se va llamando la misma función de forma recursiva y de igual forma se apunta a null y se libera la memoria como en lista enlazada. De esta forma mata los nodos primero los de la izquierda, derecha y por último la raíz. Además, el liberar del árbol binario de búsqueda funciona de la misma forma, solo que se llama el árbol correspondiente a la estructura.

```
void AVLPodarHojas(Nodo*& Raiz) // mata los de la izquierda, derecha y raiz
{
    if (Raiz != NULL) // caso continuidad
    {
        AVLPodarHojas(Raiz->izq); // I poda los menores
        AVLPodarHojas(Raiz->derecha); // D poda los mayores
        printf("Borro votante con cedula : %i\n", Raiz->llave); //
        delete(Raiz);
        Raiz = NULL; // R borra la raiz y dejo puntero apuntando a null para que arbol se mantenga c
    }
}

// ********************************************************** ARB **********************************
```

```
void PodarHojas(NodoBB*& Raiz) // mata los de la izquierda, derecha y raiz
{
    if (Raiz != NULL) // caso continuidad
    {
        PodarHojas(Raiz->HijoIzquierdo); // I poda los menores
        PodarHojas(Raiz->HijoDerecho); // D poda los mayores
        printf("Borro votante con cedula: %i \n", Raiz->llave); //
        delete(Raiz);
        Raiz = NULL; // R borra la raiz y dejo puntero apuntando a null para que arbol se mantenga consistente
    }
}
```

En el caso de liberar para el Btree se basa en la función desplegar de la misma estructura y solamente que en vez de imprimir va a poner a eliminar el nodo uno por uno hasta recorrer toda la lista llamando recursivamente así misma la función.

```
// B-Tree borrar
void borrarBTree(NodoBtree* minodo) {
    int i;
    if (minodo) {
        for (i = 0; i < minodo->count; i++) {
            borrarBTree(minodo->link[i]);
            cout << "Borro votante con cedula: " << minodo->val[i + 1] << ' ';
            delete(minodo->link[i]);
            cout << "\n";
        }
        borrarBTree(minodo->link[i]);
    }
}
```

Reporte de tiempos:

Para el reporte de tiempos general en cada una d las operaciones basucas se va guardando el tiempo que dura ene ejecutarse y se va guardando es información en variables globales de tipo double que posteriormente se llaman y se imprimen al insertar la opción 4 del menú:

```
double tiempo1, tiempo2, tiempo3, tiempo4, tiempo5, tiempo6, tiempo7, tiempo8, tiempo9,
```

```
case 4: // Reporte tiempos
    printf("\n       TIEMPO DE CARGA: %f segundos. \n", tiempo1);
    printf("\n       ------------------------------------------------Tiempos de Insercion------------------------------------------------ \n");
    printf("\n       Se ha tardado en insertar en Arbol Binario de Busqueda: %f segundos.", tiempo2);
    printf("\n       Se ha tardado en insertar en Arbol BTree: %f segundos.", tiempo3);
    printf("\n       Se ha tardado en insertar en Arbol AVL: %f segundos. \n", tiempo4);

    printf("\n       ------------------------------------------------Tiempos de Busqueda------------------------------------------------ \n");
    printf("\n       Se ha tardado en buscar en Lista Enlazada: %f segundos.", tiempo5);
    printf("\n       Se ha tardado en buscar en Arbol AVL: %f segundos.", tiempo6);
    printf("\n       Se ha tardado en buscar en Arbol Binario de Busqueda: %f segundos. ", tiempo7);
    printf("\n       Se ha tardado en buscar en Arbol BTree: %f segundos. \n", tiempo8);

    printf("\n       ------------------------------------------------Tiempos de Listar------------------------------------------------ \n");
    printf("\n       Se ha tardado en listar en Lista Enlazada: %f segundos.", tiempo9);
    printf("\n       Se ha tardado en listar en Arbol AVL: %f segundos.", tiempo10);
    printf("\n       Se ha tardado en listar en Arbol Binario de Busqueda: %f segundos. ", tiempo11);
    printf("\n       Se ha tardado en listar en Arbol BTree: %f segundos. \n", tiempo12);

    printf("\n       ------------------------------------------------Tiempos de Liberar------------------------------------------------ \n");
    printf("\n       Se ha tardado en liberar en Lista Enlazada: %f segundos.", tiempo13);
    printf("\n       Se ha tardado en liberar en Arbol AVL: %f segundos.", tiempo14);
    printf("\n       Se ha tardado en liberar en Arbol Binario de Busqueda: %f segundos. ", tiempo15);
    printf("\n       Se ha tardado en liberar en Arbol BTree: %f segundos. \n\n", tiempo16);
```

Salir:

Finalmente con la opción 5 se sale del programa con la función exit() ya que se acaba el ciclo while del menú:

```
case 5: // Salir
    exit(0);
}
```