INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO

REPORTE DE EXAMEN 3

TERCERA EVALUACION

NOMBRE DEL ALUMNO: GARCÍA QUIROZ GUSTAVO IVAN

GRUPO: 2CV3

MATERIA: ALGORITMOS Y ESTRUCTURA DE DATOS

FECHA: 19/01/2023

Índice

Introducción

Un árbol binario de búsqueda (ABB) es un tipo especial de árbol binario en el que cada nodo tiene un valor y cada sub-árbol tiene un orden específico. El valor en el nodo raíz es mayor que los valores en cualquier nodo en su sub-árbol izquierdo y menor que los valores en cualquier nodo en su sub-árbol derecho. Los valores en el sub-árbol izquierdo son menores que los valores en el nodo raíz y los valores en el sub-árbol derecho son mayores que los valores en el nodo raíz.

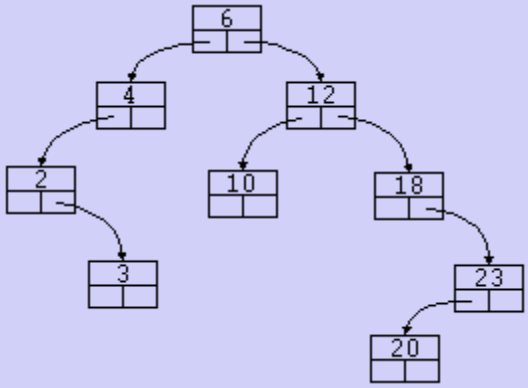
Los ABB se utilizan comúnmente para implementar estructuras de datos como diccionarios, conjuntos y mapas, ya que ofrecen una búsqueda rápida y eficiente. La complejidad del tiempo de búsqueda, inserción y eliminación es de O(log n) en promedio, donde n es el número de nodos en el árbol.

Además de la búsqueda, los ABB también soportan operaciones como recorridos in-order, pre-order y post-order, así como operaciones para encontrar el predecesor y el sucesor de un nodo dado. Sin embargo, los ABB no son adecuados para algoritmos que requieren un recorrido secuencial de todos los elementos, ya que el orden en que los elementos aparecen en un ABB depende de la estructura del árbol.

Desarrollo

Definición

Se trata de árboles de orden 2 en los que se cumple que para cada nodo, el valor de la clave de la raíz del subárbol izquierdo es menor que el valor de la clave del nodo y que el valor de la clave raíz del subárbol derecho es mayor que el valor de la clave del nodo.



Árbol binario de búsqueda

Operaciones en ABB

El repertorio de operaciones que se pueden realizar sobre un ABB es parecido al que realizábamos sobre otras estructuras de datos, más alguna otra propia de árboles:

- Buscar un elemento.
- Insertar un elemento.
- Borrar un elemento.
- Movimientos a través del árbol:
  - Izquierda.
  - Derecha.
  - Raiz.
- Información:
  - Comprobar si un árbol está vacío.

- Calcular el número de nodos.
- Comprobar si el nodo es hoja.
- Calcular la altura de un nodo.
- Calcular la altura de un árbol.

Buscar un elemento

Partiendo siempre del nodo raíz, el modo de buscar un elemento se define de forma recursiva.

- Si el árbol está vacío, terminamos la búsqueda: el elemento no está en el árbol.
- Si el valor del nodo raíz es igual que el del elemento que buscamos, terminamos la búsqueda con éxito.
- Si el valor del nodo raíz es mayor que el elemento que buscamos, continuaremos la búsqueda en el árbol izquierdo.
- Si el valor del nodo raíz es menor que el elemento que buscamos, continuaremos la búsqueda en el árbol derecho.

El valor de retorno de una función de búsqueda en un ABB puede ser un puntero al nodo encontrado, o NULL, si no se ha encontrado.

Insertar un elemento

Para insertar un elemento nos basamos en el algoritmo de búsqueda. Si el elemento está en el árbol no lo insertaremos. Si no lo está, lo insertaremos a continuación del último nodo visitado.

Necesitamos un puntero auxiliar para conservar una referencia al padre del nodo raíz actual. El valor inicial para ese puntero es NULL.

- Padre = NULL
- nodo = Raiz
- Bucle: mientras actual no sea un árbol vacío o hasta que se encuentre el elemento.

    - Si el valor del nodo raíz es mayor que el elemento que buscamos, continuaremos la búsqueda en el árbol izquierdo: Padre=nodo, nodo=nodo->izquierdo.
    - Si el valor del nodo raíz es menor que el elemento que buscamos, continuaremos la búsqueda en el árbol derecho: Padre=nodo, nodo=nodo->derecho.
- Si nodo no es NULL, el elemento está en el árbol, por lo tanto salimos.
- Si Padre es NULL, el árbol estaba vacío, por lo tanto, el nuevo árbol sólo contendrá el nuevo elemento, que será la raíz del árbol.
- Si el elemento es menor que el Padre, entonces insertamos el nuevo elemento como un nuevo árbol izquierdo de Padre.
- Si el elemento es mayor que el Padre, entonces insertamos el nuevo elemento como un nuevo árbol derecho de Padre.

Este modo de actuar asegura que el árbol sigue siendo ABB.

Borrar un elemento

Para borrar un elemento también nos basamos en el algoritmo de búsqueda. Si el elemento no está en el árbol no lo podremos borrar. Si está, hay dos casos posibles:

1. Se trata de un nodo hoja: en ese caso lo borraremos directamente.
2. Se trata de un nodo rama: en ese caso no podemos eliminarlo, puesto que perderíamos todos los elementos del árbol de que el nodo actual es padre. En su lugar buscamos el nodo más a la izquierda del subárbol derecho, o el más a la derecha del subárbol izquierdo e intercambiamos sus valores. A continuación eliminamos el nodo hoja.

Necesitamos un puntero auxiliar para conservar una referencia al padre del nodo raíz actual. El valor inicial para ese puntero es NULL.

- Padre = NULL
- Si el árbol está vacío: el elemento no está en el árbol, por lo tanto salimos sin eliminar ningún elemento.

- Si el valor del nodo raíz es igual que el del elemento que buscamos, estamos ante uno de los siguientes casos:
    - El nodo raíz es un nodo hoja:
        - Si 'Padre' es NULL, el nodo raíz es el único del árbol, por lo tanto el puntero al árbol debe ser NULL.
        - Si raíz es la rama derecha de 'Padre', hacemos que esa rama apunte a NULL.
        - Si raíz es la rama izquierda de 'Padre', hacemos que esa rama apunte a NULL.
        - Eliminamos el nodo, y salimos.
    - El nodo no es un nodo hoja:
        - Buscamos el 'nodo' más a la izquierda del árbol derecho de raíz o el más a la derecha del árbol izquierdo. Hay que tener en cuenta que puede que sólo exista uno de esos árboles. Al mismo tiempo, actualizamos 'Padre' para que apunte al padre de 'nodo'.
        - Intercambiamos los elementos de los nodos raíz y 'nodo'.
        - Borramos el nodo 'nodo'. Esto significa volver a (1), ya que puede suceder que 'nodo' no sea un nodo hoja. (Ver ejemplo 3)
- Si el valor del nodo raíz es mayor que el elemento que buscamos, continuaremos la búsqueda en el árbol izquierdo.
- Si el valor del nodo raíz es menor que el elemento que buscamos, continuaremos la búsqueda en el árbol derecho.

Programa

Primero declaro los 50 atletas los cuales estaran almacenados en un arreglo de 50 renglones y 3 columnas ya que cada atleta tendra su nombre, nacionalidad, y medallas. En la imagen tenemos un codigo y este código define una estructura de datos llamada "Node" que representa un nodo en un árbol binario de búsqueda. La estructura tiene tres campos: "name", "nacionalidad" y "medalla", que almacenan los datos de un atleta (nombre, nacionalidad y medalla obtenida, respectivamente). También tiene dos punteros, "left" y "right", que apuntan a los nodos hijo izquierdo y derecho, respectivamente.

Además, se declaran cuatro punteros de nodos: "root", "Oro", "Plata" y "Bronce", que apuntan a la raíz del árbol, a la raiz del subarbol de los atletas con medallas de oro, a la raiz del subarbol de los atletas con medallas de plata y a la raiz del subarbol de los atletas con medallas de bronce, respectivamente. Estos punteros se utilizarán para navegar y realizar operaciones en el árbol. Sin embargo, se debe tener en cuenta que en este código no se estan realizando acciones con los punteros, solo se estan declarando.

```
//50 datos
std::string atleta[55][3]={{"Aaron","argentina","oro"   },{ "Barbara"   ,"brazil",  "plata" },{ "Charles"   ,"rusia",   "bronce"

// arbol
struct Node {
    std::string name;
    std::string nacionalidad;
    std::string medalla;
    Node *left;
    Node *right;
};

Node * root = NULL;
Node * Oro = NULL;
Node * Plata = NULL;
Node * Bronce = NULL;
```

Este código define una función llamada "insertNode" que toma un argumento entero "i" y utiliza ese valor para insertar un nuevo nodo en un árbol binario de búsqueda. La función hace lo siguiente:

Crea un puntero de nodo llamado "newNode" y asigna un nuevo nodo en memoria dinámica.

Asigna los valores del atleta en las posiciones [i][0], [i][1] y [i][2] del arreglo 'atleta' al nuevo nodo en los campos "name", "nacionalidad" y "medalla", respectivamente.

Asigna NULL a los punteros "left" y "right" del nuevo nodo.

Verifica si el árbol está vacío. Si es así, el nuevo nodo se convierte en la raíz del árbol.

En caso contrario, se utiliza un ciclo while para buscar un lugar adecuado para el nuevo nodo en el árbol. Se utiliza un puntero "current" para recorrer el árbol y un puntero "parent" para mantener una referencia al nodo actual. Si el valor del atleta en [i][0] es menor que el valor del nodo actual, se mueve hacia el subárbol izquierdo. Si es mayor, se mueve hacia el subárbol derecho. Si se encuentra un espacio vacío, se inserta el nuevo nodo allí.

En resumen, esta función se encarga de insertar un nuevo atleta en el arbol binario de busqueda.

```
23  void insertNode(int i) {
24      Node *newNode = new Node;
25      newNode->name = atleta[i][0];
26      newNode->nacionalidad = atleta[i][1];
27      newNode->medalla=atleta[i][2];
28      newNode->left = NULL;
29      newNode->right = NULL;
30
31      if (root == NULL) {
32          root = newNode;
33      } else {
34          Node *current = root;
35          Node *parent;
36          while (true) {
37              parent = current;
38              if (atleta[i][0] < current->name) {
39                  current = current->left;
40                  if (current == NULL) {
41                      parent->left = newNode;
42                      return;
43                  }
44              } else {
45                  current = current->right;
46                  if (current == NULL) {
47                      parent->right = newNode;
48                      return;
49                  }
50              }
51          }
```

Tambien el codigo tiene otras 2 funciones que se encargan de hacer lo mismo pero se diferencia de los demas cada una en que se ordenan por cada medalla, en otras palabras, se ordena en 3 arboles binarios de nombres oro, plata y bronce.

```
void insertNodeOro(int i) {
    Node *newNode = new Node;
    newNode->name = atleta[i][0];
    newNode->nacionalidad = atleta[i][1];
    newNode->medalla=atleta[i][2];
    newNode->left = NULL;
    newNode->right = NULL;

    if (Oro == NULL) {
        Oro = newNode;
    } else {
        Node *currentOro = Oro;
        Node *parentOro;
        while (true) {
            parentOro = currentOro;
            if (atleta[i][0] < currentOro->name)
                currentOro = currentOro->left;
                if (currentOro == NULL) {
                    parentOro->left = newNode;
                    return;
                }
            } else {
                currentOro = currentOro->right;
                if (currentOro == NULL) {
                    parentOro->right = newNode;
                    return;
                }
            }
        }
    }
```

```
void insertNodePlata(int i) {
    Node *newNode = new Node;
    newNode->name = atleta[i][0];
    newNode->nacionalidad = atleta[i][1];
    newNode->medalla=atleta[i][2];
    newNode->left = NULL;
    newNode->right = NULL;

    if (Plata == NULL) {
        Plata = newNode;
    } else {
        Node *currentPlata=Plata;
        Node *parentPlata;
        while (true) {
            parentPlata = currentPlata;
            if (atleta[i][0] < currentPlata->name) {
                currentPlata = currentPlata->left;
                if (currentPlata== NULL) {
                    parentPlata->left = newNode;
                    return;
                }
            } else {
                currentPlata= currentPlata->right;
                if (currentPlata== NULL) {
                    parentPlata->right = newNode;
                    return;
                }
            }
        }
    }
```

Despues existen tres funciones llamadas preorden, postorden e inorden que reciben como parámetro un puntero del tipo Node llamado root. Cada una de estas funciones está diseñada para recorrer un árbol binario de búsqueda en un orden específico.

La función preorden imprime el valor del nodo raíz, luego recorre el sub-árbol izquierdo y finalmente el sub-árbol derecho.

La función postorden recorre primero el sub-árbol izquierdo, luego el sub-árbol derecho y finalmente imprime el valor del nodo raíz.

La función inorden recorre primero el sub-árbol izquierdo, luego imprime el valor del nodo raíz y finalmente recorre el sub-árbol derecho.

Cada función utiliza una estructura de control de tipo recursión para recorrer el árbol. Si el puntero root es nulo, la función regresa y termina la recursión.

```
117  //Funcion rrecorrido en profundidad preorden
118  void preorden(Node *root){
119      if(root==NULL){
120          return ;
121      }else{
122          std::cout << root->name << "-";
123          preorden(root->left);
124          preorden(root->right);
125      }
126  }
127
128  //Funcion rrecorrido en profundidad postorden
129  void postorden(Node *root){
130      if(root==NULL){
131          return ;
132      }else{
133          postorden(root->left);
134          postorden(root->right);
135          std::cout << root->name << "-";
136      }
137  }
138
139  void inorden(Node * root){
140      if(root==NULL){
141          return ;
142      }else{
143          inorden(root->left);
144          std::cout << root->name << "-";
145          inorden(root->right);
146      }
147  }
```

Luego el siguiente código es una función que busca un nodo específico en un árbol binario de búsqueda. La función toma dos parámetros: el nodo raíz del árbol y el nombre del nodo que se está buscando. Utiliza el algoritmo de búsqueda recursivo para recorrer el árbol y buscar el nodo que tenga el nombre especificado.

La función comienza verificando si el nodo raíz es nulo, en caso de serlo devuelve null. Luego verifica si el nombre del nodo actual es igual al nombre buscado, si es así devuelve el nodo actual. En caso contrario, si el nombre buscado es menor al nombre del nodo actual, se llama a la función recursivamente con el nodo hijo izquierdo. Si no, se llama a la función recursivamente con el nodo hijo derecho. Si el nodo buscado no se encuentra en el árbol, la función devuelve null.

```
148 Node* searchNode(Node* root,std::string name) {
149     if (root == NULL) {
150         return NULL;
151     }
152     if (root->name == name) {
153         return root;
154     }
155     if (name < root->name) {
156         return searchNode(root->left, name);
157     }
158     return searchNode(root->right, name);
159 }
160
```

Por ultimo el código main tiene como objetivo crear un árbol binario de búsqueda (ABB) para almacenar atletas con información como su nombre, nacionalidad y medalla obtenida en unos juegos deportivos. El árbol se crea a partir de un arreglo bidimensional llamado "atleta" que tiene información de 50 atletas.

La función "insertNode" es utilizada para insertar un nuevo nodo en el árbol. Recibe un entero "i" que es utilizado para acceder a la información del atleta en el arreglo "atleta". Se crea un nuevo nodo y se le asigna la información del atleta correspondiente. Si el árbol está vacío, el nuevo nodo se convierte en la raíz del árbol. Si no está vacío, se recorre el árbol buscando el lugar adecuado para insertar el nuevo nodo.

Hay tres funciones para hacer recorridos en profundidad del árbol: "preorden", "inorden" y "postorden". Cada función recibe como parámetro la raíz del árbol y realiza un recorrido en profundidad del árbol, imprimiendo el nombre de cada nodo visitado.

La función "searchNode" es utilizada para buscar un nodo específico en el árbol. Recibe como parámetros la raíz del árbol y un nombre a buscar. Si el nodo es encontrado, se imprime la información del nodo y se retorna un puntero al nodo. Si no es encontrado, se retorna un puntero nulo.

En la función main se realiza un menú con diferentes opciones para el usuario, entre ellas buscar un atleta por nombre y nacionalidad, y sumar todos los puntos del equipo. También se pueden hacer recorridos en profundidad del árbol.

```
int main() {
    int dato, opcion, contador=0, i=0, j=0, suma=0;
        std::string nombre;
        std::string nacionalidad;
    for(i=0;i<50;i++){
    if(atleta[i][2]=="oro"){
    insertNodeOro(i);
    } else if(atleta[i][2]== "plata"){
    insertNodePlata(i);
    } else{
    insertNode(i);
    }
    }

    do{
        printf("\n\t \t\t\t Menu\n");
        printf("1.-Buscar un nodo por su nombre y nacionalidad y sumar todos los puntos del equipo\n");
//      printf("1. insertar nodos\n");
        printf("2.-Recorrido en Preorden\n");
        printf("3.-Recorrido en Inorden\n");
        printf("4.-Recorrido en Postorden\n");
        printf("5.-Salir \n");
        printf("\n Elije una opcion:\n");
        scanf("%d",&opcion);

        switch(opcion){
            case 1:{
        std::cout << "Ingresa el nombre: ";
        std::cin >> nombre;
        std::cout << "Ingresa la nacionalidad: ";
        std::cin >> nacionalidad;
//      std::cout << "Hola " << nombre << "!" << std::endl;
```

```
    Node* result = searchNode(root, nombre);
    if (result != NULL) {
        std::cout << "Nodo encontrado: " << result->name<<" tiene la medalla de:"<<result->medalla << std::endl;
        std::cout << std::endl;
    } else {
        std::cout << "Nodo no encontrado en las medallas de Bronce" << std::endl;
        std::cout << std::endl;
    }

    Node* resultO = searchNode(Oro, nombre);
    if (resultO != NULL) {
        std::cout << "Nodo encontrado: " << resultO->name<<" tiene la medalla de:"<<resultO->medalla << std::endl;
    } else {
        std::cout << "Nodo no encontrado en las medallas de Oro" << std::endl;
        std::cout << std::endl;
    }

    Node* resultP = searchNode(Plata, nombre);
    if (resultP != NULL) {
        std::cout << "Nodo encontrado: " << resultP->name<<"tiene la medalla de:"<<resultP->medalla << std::endl;
        std::cout << std::endl;
    } else {
        std::cout << "Nodo no encontrado  en las medallas de Plata" << std::endl;
        std::cout << std::endl;
    }
        std::cout << "Equipo de nacionalidad: " << nacionalidad << std::endl;
        std::cout << std::endl;

for(int i = 0; i < 50; i++){
    if(atleta[i][1]==nacionalidad){
        if(atleta[i][2]== "oro"){
            std::cout<<"nombre: \t "<< atleta[i][0]<< " \t nacionalidad: \t"<< atleta[i][1]<<" \t medalla: \t "<< atleta[i][2]<< std::endl;
        suma = suma + 10;
```

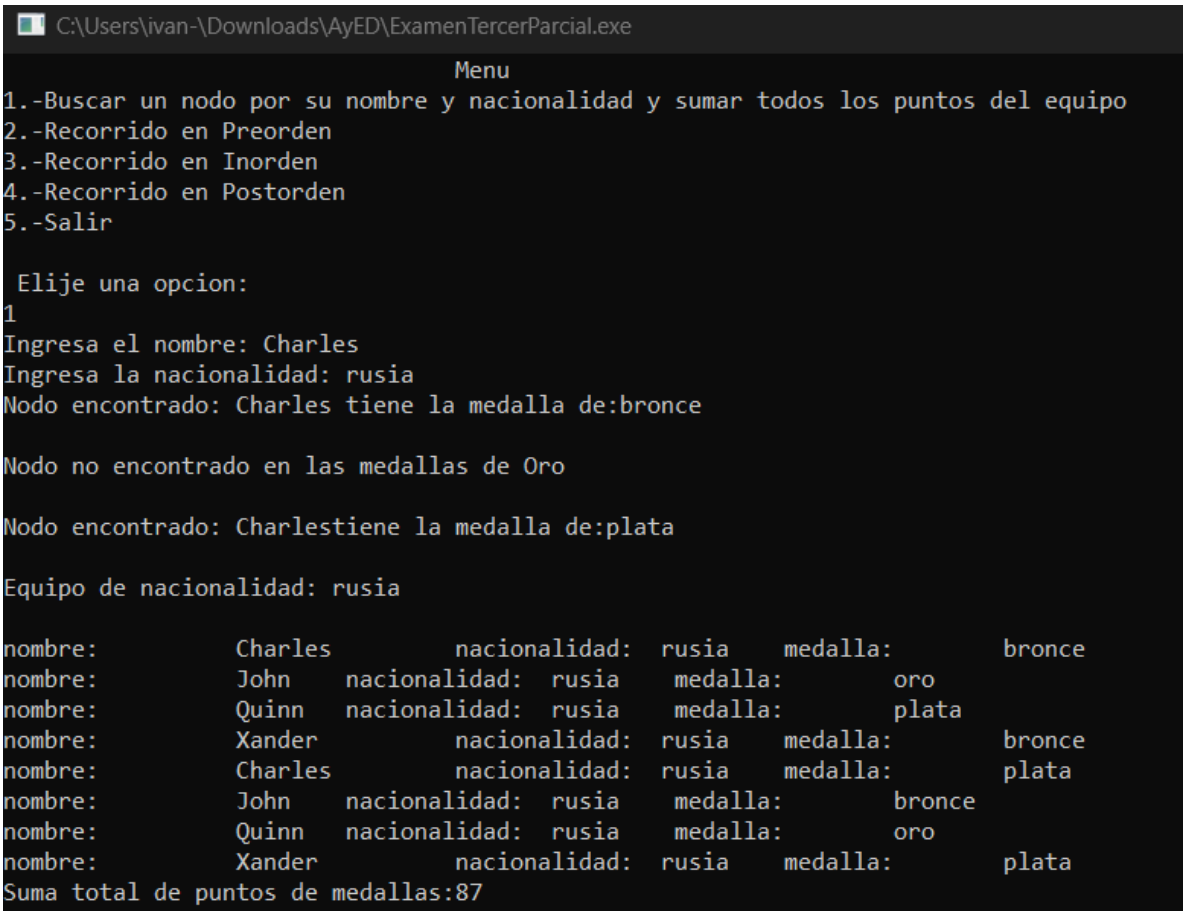
```
        suma = suma + 10;
    } else if(atleta[i][2]== "plata"){
            std::cout<<"nombre: \t "<< atleta[i][0]<< " \t nacionalidad: \t"<< atleta[i][1]<<" \t medalla: \t "<< atleta[i][2]<< std::endl;
        suma = suma + 5;
    } else{
            std::cout<<"nombre: \t "<< atleta[i][0]<< " \t nacionalidad: \t"<< atleta[i][1]<<" \t medalla: \t "<< atleta[i][2]<< std::endl;
        suma = suma + 2;
    }
    }
}

std::cout<<"Suma total de puntos de medallas:"<<suma;


            break;
        }
        case 2:{
            printf("\nDame el arbol que quieres recorrer\n");
            printf("1. Oro\n");
            printf("2. Plata\n");
            printf("3.-Bronce\n");
            scanf("%d",&opcion);
            printf("\n\t \t Recorrer arbol en preorden \n");
            if(opcion==1){
            preorden(Oro);
            }else if(opcion==2){
            preorden(Plata);
            }else if(opcion==3){
            preorden(root);
            }
//          inorden(Oro);
```

```
            }
            case 3:{
                printf("\nDame el arbol que quieres recorrer\n");
            printf("1. Oro\n");
            printf("2. Plata\n");
            printf("3.-Bronce\n");
            scanf("%d",&opcion);
            printf("\n\t \t Recorrer arbol en inorden \n");
                if(opcion==1){
                    inorden(Oro);
                }else if(opcion==2){
                inorden(Plata);
                }else if(opcion==3){
                inorden(root);
                }
                break;
            }
            case 4:{
                printf("\nDame el arbol que quieres recorrer\n");
            printf("1. Oro\n");
            printf("2. Plata\n");
            printf("3.-Bronce\n");
            scanf("%d",&opcion);
                printf("\n\t \t Recorrer arbol en postorden \n");
                if(opcion==1){
                postorden(Oro);
                }else if(opcion==2){
                postorden(Plata);
                }else if(opcion==3){
                postorden(root);
                }
//              postorden(arbol);
                break;
```

Resultados


C:\Users\ivan-\Downloads\AyED\ExamenTercerParcial.exe
Menu
1.-Buscar un nodo por su nombre y nacionalidad y sumar todos los puntos del equipo
2.-Recorrido en Preorden
3.-Recorrido en Inorden
4.-Recorrido en Postorden
5.-Salir
Elije una opcion:
1
Ingresa el nombre: Charles
Ingresa la nacionalidad: rusia
Nodo encontrado: Charles tiene la medalla de:bronce
Nodo no encontrado en las medallas de Oro
Nodo encontrado: Charlestiene la medalla de:plata
Equipo de nacionalidad: rusia
nombre: Charles nacionalidad: rusia medalla: bronce
nombre: John nacionalidad: rusia medalla: oro
nombre: Quinn nacionalidad: rusia medalla: plata
nombre: Xander nacionalidad: rusia medalla: bronce
nombre: Charles nacionalidad: rusia medalla: plata
nombre: John nacionalidad: rusia medalla: bronce
nombre: Quinn nacionalidad: rusia medalla: oro
nombre: Xander nacionalidad: rusia medalla: plata
Suma total de puntos de medallas:87

```
                                Menu
1.-Buscar un nodo por su nombre y nacionalidad y sumar todos los puntos del equipo
2.-Recorrido en Preorden
3.-Recorrido en Inorden
4.-Recorrido en Postorden
5.-Salir

 Elije una opcion:
1
Ingresa el nombre: Barbara
Ingresa la nacionalidad: brazil
Nodo no encontrado en las medallas de Bronce

Nodo encontrado: Barbara tiene la medalla de:oro
Nodo encontrado: Barbaratiene la medalla de:plata

Equipo de nacionalidad: brazil

nombre:          Barbara         nacionalidad:  brazil   medalla:        plata
nombre:          Isabella        nacionalidad:  brazil   medalla:        bronce
nombre:          Peter   nacionalidad:  brazil   medalla:        oro
nombre:          William         nacionalidad:  brazil   medalla:        plata
nombre:          Barbara         nacionalidad:  brazil   medalla:        oro
nombre:          Isabella        nacionalidad:  brazil   medalla:        plata
nombre:          Peter   nacionalidad:  brazil   medalla:        bronce
nombre:          William         nacionalidad:  brazil   medalla:        oro
Suma total de puntos de medallas:136
```

```
C:\Users\ivan-\Downloads\AyED\ExamenTercerParcial.exe
1

                Recorrer arbol en preorden
Aaron- Susan-David-Barbara-George-Elizabeth-John-Heather-Michael-Katherine-Peter-Nicole-Victoria-Quinn-Thomas-Yolanda-William-
                            Menu
1.-Buscar un nodo por su nombre y nacionalidad y sumar todos los puntos del equipo
2.-Recorrido en Preorden
3.-Recorrido en Inorden
4.-Recorrido en Postorden
5.-Salir

 Elije una opcion:
3

Dame el arbol que quieres recorrer
1. Oro
2. Plata
3.-Bronce
1

                Recorrer arbol en inorden
 Susan-Aaron-Barbara-David-Elizabeth-George-Heather-John-Katherine-Michael-Nicole-Peter-Quinn-Thomas-Victoria-William-Yolanda-
```

```
C:\Users\ivan-\Downloads\AyED\ExamenTercerParcial.exe

                Recorrer arbol en postorden
 Susan-Barbara-Elizabeth-Heather-Katherine-Nicole-Thomas-Quinn-William-Yolanda-Victoria-Peter-Michael-John-George-David-Aaron-
                            Menu
1.-Buscar un nodo por su nombre y nacionalidad y sumar todos los puntos del equipo
2.-Recorrido en Preorden
3.-Recorrido en Inorden
4.-Recorrido en Postorden
5.-Salir

 Elije una opcion:
5

--------------------------------
Process exited after 461.7 seconds with return value 1
Presione una tecla para continuar . . .
```

Conclusión

En conclusión, los árboles binarios de búsqueda (ABB) son una estructura de datos importante en C++ que se utiliza para organizar y buscar información de manera eficiente. Los árboles ABB son similares a los árboles binarios regulares, pero tienen la propiedad adicional de que cada nodo tiene un valor que es mayor que todos los valores en su subárbol izquierdo y menor que todos los valores en su subárbol derecho. Esto permite que los árboles ABB sean muy rápidos al buscar información en ellos. Los árboles ABB también tienen varias operaciones, como insertar, eliminar y buscar nodos, así como diferentes tipos de recorridos, como preorden, inorden y postorden, que pueden ser utilizados para recorrer y procesar todos los nodos en el árbol. Los árboles ABB son ampliamente utilizados en aplicaciones como bases de datos, sistemas de archivos y algoritmos de búsqueda, debido a su capacidad para organizar y buscar información de manera eficiente.

Bibliografía

Pozo, S. (s/f). C Con Clase. Conclase.net. Recuperado el 19 de enero de 2023, de https://conclase.net/c/edd/cap7