EIE
Escuela de
Ingeniería Eléctrica

UNIVERSIDAD DE COSTA RICA

UNIVERSIDAD DE COSTA RICA

FACULTAD DE INGENIERIA

ESCUELA DE INGENIERIA ELÉCTRICA

ESTRUCTURAS ABSTRACTAS DE DATOS Y ALGORITMOS PARA INGENIERÍA

LABORATORIO 5: ÁRBOLES BST

ESTUDIANTES:
JESÚS ZÚÑIGA MÉNDEZ (B59084)
DENNIS CHAVARRÍA SOTO (B82097)

PROFESOR:
RICARDO ROMÁN BRENES; M. SC.

II CICLO 2019

Índice

Índice de figuras

1. Introduccion

Uno de los temas del curso de Estructuras de Datos Abstractas y Algoritmos para Ingeniería es el de los árboles, los cuáles permiten organizar datos y acceder a ellos de una forma similar a las listas enlazadas, sin embargo, estos se ordenan dependiendo del valor de los datos o alguna características que discrimine entre elementos de mayor o menor tamaño, pues, los valores menores a uno central, llamado raíz, van a su izquierda, mientras que los mayores van a la derecha.

Los árboles deben mantener la relación de menor y mayor que, entre las raíces y sus hijos, así pues, cuando la raíz de todo tiene un valor numérico y a su derecha tiene un hijo mayor, los descendientes menores de este deberán ser más grandes que la primera raíz, mas no así para su padre; esto debe considerarse con mucha cautela al construir una estructura de datos de este tipo, de lo contrario, se podría producir incongruencias con la teoría así como de funcionamiento de un programa que opere con ellas.

2. Solución

Para la solucion se implementaron tres clases con las cuales se logra crear y estructurar el árbol binari3o estas tres clases son

2.1. DatoNoPrimitivo

Es una clase emplantillada con la que se implementa un dato para usar, esta clase simplemente se compone de una variable llamada dato en la que se almacena el dato ademas de la sobrecarga del operador -"que simplemente permite devolver el valor del dato y el operador + que compara entre dos datos

2.2. ClassNode

Es una clase emplantillada que se compone de una variable de tipo DatoNoPrimitivo, y dos de tipo ClassNode, uno para el Hijo izquierdo y otro para el derecho, en esta clase lo uno que se implementa es el metodo constructor que asigna el valor recibido al valor de la variable

2.3. BinarySearchTree

En esta clase se implementa la estructuración del arbol valiendose de las dos clases anteriores y de varios metodos los cuales se explican a continuacion

2.3.1. Insert

En este metodo se crea un nodo para un dato nuevo, se compone de un while que se recorre hasta encontrar el campo donde se debe crear el nodo y de un if que permite comparar si el valor del nodo es mayor o menor para poder moverse entre ramas izquierdas o derechas

2.3.2. LargesToTheLeft y SmallesToTheRight

Son dos metodos aparte, pero se explican juntos ya que realizan el mismo procedimiento, en estos metodos se devuelve el nodo mayor a la izquierda o menor a la derecha de un nodo recibido como parametro, esto se logra mediante un if que se mueve a alguno de los lados si existe el nodo y despues se mueve hasta el lado contrario hasta que existan nodos

2.3.3. NodeOf y DataIn

En este caso estos metodos devuelven el nodo de un dato o el dato de un él, basicamente se mueve entre izquierda y derecha hasta encontrar que el valor de el dato o el puntero de la memoria sea el mismo y devuelve el dato solicitado

2.3.4. remove

En este metodo se borra un nodo, esto se hace realizando una búsqueda sobre el árbol comparando el dato que se quiere borrar, al encontrarlo se corren los metodos largesToTheLeft y smallesToTheRight, se decide entre alguno de estos valores para sustituirlo por el borrado escogiendo el valor que exista en cada caso

2.3.5. Algoritmos para recorrer el árbol

Todos los métodos que pueden recorrer el árbol en diferentes órdenes, son recursivos, además, su estructura es muy similar entre ellos. El funcionamiento que tienen es el que sigue:

- **PreOrden:** Este orden consiste en *Raíz, Izquierda, Derecha*, es decir, cuando se detecta una raíz se imprime el valor que este nodo contiene, luego, si a la izquierda existe un nodo, se invoca la misma función de forma recursiva y si este otro es una raíz para dos o un hijo, imprime su valor, sin embargo, si no tiene nodo izquierdo ni derecho, se imprime. Cuando detecta que existe un hijo derecho, invoca nuevamente la función recursiva. Es de destacar que se pasa como parámetro el puntero que apunta al hijo específico que se encuentra.
- **InOrden:** Este orden de búsqueda consiste en *Izquierda, Raíz, Derecha*, por lo que, viaja por cada rama-hijo izquierdo del árbol, una vez que este no tiene hijos izquierdos, imprime el valor numérico que tal nodo contiene, luego, imprime el de su raíz y sigue lo mismo que con los hijos izquierdos pero, esta vez, con la derecha, una vez llegado a tal hijo se dirige a la izquierda y busca los de este otro nodo, cuando no encuentra, vuelve a imprimir el izquierdo y luego la raíz, de último el valor que tiene el nodo derecho. Destaca que se usa un condicional para raíz impresa, esto hace que, si el nodo ya imprimió su valor como raíz, no lo vuelva a hacer cuando se le identifique como el último hijo izquierdo; del mismo modo, cuenta con los condicionales de PreOrden para determinar si existen o no hijos izquierdo y derecho. La impresión del valor de nodo como raíz se realiza *al final*
- **PostOrden:** Este orden es muy similar al de InOrden, pero con la particularidad de que imprime el valor de cada nodo siguiendo el orden *Izquierda, Derecha, Raíz*, así, el valor de la raíz que tiene los dos hijos se imprime de última, pasando primero por todos los izquierdos, luego por los derechos (menores y luego mayores, respectivamente) para imprimir, finalmente, la raíz. También cuenta con el identificador de impresión del valor que el nodo contiene como una raíz, osea, si ya lo imprimió con anterioridad, no lo volverá a hacer;

para que esto funcione en PostOrden, se debió colocar el condicional para mostrar el valor contenido, a la mitad del método, antes de los condicionales para identificar la existencia de un hijo derecho.

- **print:** Usa los metodos anteriores para imprimir un arbol de forma gráfica

2.3.6. Adicionales

Además de los metodos solicitados en el laboratorio se implementaron otros metodos adicionales para poder imprimir el árbol, estos son:

- **calcularNiveles:** Calcula la cantidad de niveles de un arbol
- **calcularNodos:** Calcula la cantidad de nodos en un nivel
- **devolverStringNivel:** Devuelve la representacion a string de un nivel
- **print:** Usa los metodos anteriores para imprimir un arbol de forma gráfica
- **Existe** Funcion que verifica que un dato existe o no

3. Conclusiones

- Los arboles binarios son una buena forma de organizar datos numericos no repetidos a partir de un ordenamiento que discrimine entre mayores y menores.
- Gracias a las destrezas desarrolladas con la implementación de listas enlazadas, se comprendió y logró la implementación de un árbol BST con el lenguaje de programación C++.
- Se aprendió sobre la existencia de tres algoritmos para recorrer un árbol BST, además, se logró implementar para este laboratorio.
- Fue posible crear, implementar e instanciar un árbol con métodos para insertar nodos, reordenarse en caso de eliminar una raíz, remover raíces, encontrar datos o posiciones, así como recorrerlos con diferentes algoritmos.

4. Anexos

```
/**
 * @file main.c
 * @author Jesus Zu iga Mendez
 * @author Dennis Chavarria Soto
 * @brief Archivo pricipal, Laboratorio sobre herencia representada con geometria
 * @version 1.0
 * @date 24 de setiembre de 2019
 * @copyright Copyleft (l) 2019
 */

#include "./include/Includes.h"

using namespace std;
//using namespace std::chrono;



int numeroRandom(int minimo, int tope){
    //codigo tomado de https://es.stackoverflow.com/questions/148661/por-qu%C3%A9-el-n%C3%BAmero-que-
    // Tenemos control sobre el algoritmo y distribuci n a usar.
    random_device device;
    // Se usa el algoritmo Mersenne twister
    // https://es.wikipedia.org/wiki/Mersenne_twister
    mt19937 generador(device());
    // Escogemos una distribucion uniforme entre 0 y 100
    uniform_int_distribution<> distribucion(minimo, tope);
    /* Generamos un n mero pseudo-aleatorio con el algoritmo
    mt19937 distribuido uniformemente entre 0 y 100 */
    int a = distribucion(generador);
    return a;
}

/**
 * @brief Imprime Un menu
 */
int Menu(){
    int respuesta;
    cout << "Escoja la opcion\n\n" << endl;
    cout << "1:  Insertar Elemento" << endl;
    cout << "2:  LargeToTheLefth" << endl;
    cout << "3:  SmallesToTheRigth" << endl;
    cout << "4:  NodeOf" << endl;
    cout << "5:  DataIn" << endl;
    cout << "6:  Remove" << endl;
    cout << "7:  PreOrden" << endl;
    cout << "8:  InOrden" << endl;
    cout << "9:  PosOrden" << endl;
    cout << "10: Imprimir" << endl;
    cout << "0:  Salir" << endl;
    cout << endl;
    cin >> respuesta;
    return (respuesta);
}

/**
 * @brief Funcion main del codigo
```

```
 */
int main(int argc, char** argv){

    int dimensionArregloMasUno = argc;
    int arregloParaElArbol[dimensionArregloMasUno - 1];
    BinarySearchTree<DatoNoPrimitivo<int>, ClassNode<DatoNoPrimitivo<int>>> arbol;
    if(dimensionArregloMasUno > 1){
        for (int i = 1; i< dimensionArregloMasUno ; i++){
            //cout << argv[i] << endl;
            arregloParaElArbol[i-1] = stoi(argv[i]);
            //cout << arregloParaElArbol[i-1] << endl;
        }
        for (int i = 0; i < (dimensionArregloMasUno - 1); i++){
            DatoNoPrimitivo<int> dato(arregloParaElArbol[i]);
            arbol.insert(dato);
        }
    }
    arbol.print();
    int seguir = 1;
    int datoLeido;
    do{
        seguir = Menu();
        if (seguir == 1){
            cout << "Digite el dato que quiere insertar" << endl;
            cin >> datoLeido;
            DatoNoPrimitivo<int> dato(datoLeido);
            if (arbol.Existe(dato) == 1){
                cout << "El dato ya existe en el arbol" << endl;
            }else{
                arbol.insert(dato);
            }
        }
        if (seguir == 2){
            cout << "A partir de la raiz el dato es" << endl;
            ClassNode<DatoNoPrimitivo<int>> *elNodo = arbol.largesToTheLeft(*arbol.raiz);
            cout << ~ *elNodo->valor << endl;
        }
        if (seguir == 3){
            cout << "A partir de la raiz el dato es" << endl;
            ClassNode<DatoNoPrimitivo<int>> *elOtroNodo = arbol.smallesToTheRight(*arbol.raiz);
            cout <<  ~ *elOtroNodo->valor << endl;
        }
        if (seguir == 4){
            cout << "Digite el dato que quiere buscar" << endl;
            cin >> datoLeido;
            DatoNoPrimitivo<int> dato(datoLeido);
            if (arbol.Existe(dato) == 0){
                cout << "El dato no existe en el arbol" << endl;
            }else{
                ClassNode<DatoNoPrimitivo<int>> *elNodoDelDato = arbol.NodeOf(dato);
                cout << "la memoria del nodo es " << elNodoDelDato << endl;
            }
        }
        if (seguir == 5){
            DatoNoPrimitivo<int> *eldato =  arbol.dataIn(*arbol.raiz);
            cout << "el dato en la raiz es " << ~ *eldato << endl;
        }
        if (seguir == 6){
            cout << "Digite el dato que quiere borrar" << endl;
```

```
            cin >> datoLeido;
            DatoNoPrimitivo<int> dato(datoLeido);
            if (arbol.Existe(dato) == 0){
                cout << "El dato no existe en el arbol" << endl;
            }else{
                arbol.remove(dato);
            }
        }
        if (seguir == 7){
            arbol.preorden(arbol.raiz);
            cout << endl;
        }
        if (seguir == 8){
            arbol.InOrden(arbol.raiz);
            cout << endl;
        }
        if (seguir == 9){
            arbol.PostOrden(arbol.raiz);
            cout << endl;
        }
        if (seguir == 10){
            arbol.print();
        }
        /*switch (seguir)
        {
        case 1:
            break;
        case 2:
            cout << "Large" << endl;
            break;
        case 3:
            cout << "Small" << endl;
            break;
        case 4:
            cout << "Node" << endl;
            break;
        case 5:
            cout << "Data" << endl;
            break;
        case 6:
            cout << "Remove" << endl;
            break;
        case 7:
            cout << "Pre" << endl;
            break;
        case 8:
            cout << "In" << endl;
            break;
        case 9:
            cout << "Pos" << endl;
            break;
        case 10:
            cout << "imprimir" << endl;
            break;
        default:
            break;
        }*/

    }while (seguir != 0);
```

```
175
176
177
178     /*int arreglodatos[14] = {20,8,25,2,15,10,17,9,11,100,50,75,40,30};
179     BinarySearchTree<DatoNoPrimitivo<int>, ClassNode<DatoNoPrimitivo<int>>> arbol;
180     cout << "Llenando el arbol" << endl;
181     for (int i = 0; i < 14; i++){
182         DatoNoPrimitivo<int> dato(arreglodatos[i]);
183         arbol.insert(dato);
184     }*/
185 //     cout << "Imprimiendp el arbol" << endl;
186 //     arbol.print();//espaciado inicial para aegurarse de que quede bien impreso, el niver que se des
187 //     cout << "finalizando" << endl;
188 //     cout << "larges to the left de la raiz" << endl;
189 //     ClassNode<DatoNoPrimitivo<int>> *elNodo = arbol.largesToTheLeft(*arbol.raiz);
190 //     cout << "el mayor a la izquierda es " << ~ *elNodo->valor << endl;
191 //     ClassNode<DatoNoPrimitivo<int>> *elOtroNodo = arbol.smallesToTheRight(*arbol.raiz);
192 //     cout << "el menor a la derecha es " << ~ *elOtroNodo->valor << endl;
193     // DatoNoPrimitivo<int> *eldato =  arbol.dataIn(*arbol.raiz);
194     // cout << "el dato en el nodo es " << ~ *eldato << endl;
195     // DatoNoPrimitivo<int> datoABuscar(17);
196     // ClassNode<DatoNoPrimitivo<int>> *elNodoDelDato = arbol.NodeOf(datoABuscar);
197     // eldato = arbol.dataIn(*elNodoDelDato);
198     // cout << "la memoria del nodo es " << elNodoDelDato << endl;
199     //cout << "el dato en el nodo es " << ~ *eldato << endl;
200     // cout << "posorden" << endl;
201     // arbol.PostOrden(arbol.raiz);
202     // cout << "inorden" << endl;
203     // arbol.InOrden(arbol.raiz);
204     // cout << "preorden" << endl;
205     // arbol.preorden(arbol.raiz);
206
207
208     //for (int i = 0; i < 14; i++){
209         // cout << "borrare" << endl << endl << endl << endl;
210         // DatoNoPrimitivo<int> datoABorrar(20);
211         // arbol.remove(datoABorrar);
212         // cout << endl << endl;
213     //}
214     // cout << "voy a imprmir" << endl;
215     // arbol.print();
216     return 0;
217 }
```

Listing 1: main.cpp

```cpp
#ifndef BINARYSEARCHTREE_H
#define BINARYSEARCHTREE_H

#include "./Includes.h"

using namespace std;
// ...
template <typename Data , typename TypeNodo >
    class BinarySearchTree{
        public :
            BinarySearchTree(){

            };
            ~BinarySearchTree(){

            };

            TypeNodo &insert (const Data &dato)
            {
                //cout << "dato recibido " << dato.dato << endl;
                //cout << "cantidad items " << this->items << endl;
                if (this->items == 0)
                {
                    raiz = new ClassNode<Data>(new Data(dato));
                    this->items++;
                    //cout << "sume un item " << this->items << endl;
                }
                else
                {
                    ClassNode<Data> *last = raiz;
                    int continuar = 0;
                    while (continuar == 0)
                    {
                        if (dato.dato > ~ *last->valor){
                            if (last->HijoDerecho == 0x0){
                                last->HijoDerecho = new ClassNode<Data>(new Data(dato));
                                this->items++;
                                continuar++;
                                return *(last->HijoDerecho);
                            }else{
                                last = last->HijoDerecho;
                            }
                        }else{
                            if (last->HijoIzquierdo == 0x0){
                                last->HijoIzquierdo = new ClassNode<Data>(new Data(dato));
                                this->items++;
                                continuar++;
                                return *(last->HijoIzquierdo);
                            }else{
                                last = last->HijoIzquierdo;
                            }
                        }
                    }
                }
                //cout << "la memoria de la raiz " << this->raiz << endl;
                //cout << "el valor de la raiz " << ~ *this->raiz->valor << endl;
                //cout << "sali con esta cantidad de items " << this->items << endl;
                return *raiz;
            };
```

```
60
61
62
63          /**
64           * @brief Funcion que devuelve el hijo mas grande a la izquierda
65           */
66          TypeNodo *largesToTheLeft ( TypeNodo &nodoInicial){
67              ClassNode<Data> *actual;
68              actual = &nodoInicial;
69              //cout << "el valor de actual es " << ~  *actual->valor << endl;
70              if (actual -> HijoIzquierdo != 0x0){
71                  actual = actual -> HijoIzquierdo;
72                  int continuar = 0;
73                  while (continuar == 0){
74                      if (actual -> HijoDerecho != 0x0){
75                          actual = actual ->HijoDerecho;
76                      }else{
77                          continuar = 1;
78                      }
79                  }
80              }
81              return actual;
82          };
83
84          /**
85           * @brief Funcion que devuelve el hijo mas peque o a la derecha
86           */
87          TypeNodo *smallesToTheRight ( TypeNodo &nodoInicial){
88              ClassNode<Data> *actual;
89              actual = &nodoInicial;
90              //cout << "el valor de actual es " << ~  *actual->valor << endl;
91              if (actual -> HijoDerecho != 0x0){
92                  actual = actual -> HijoDerecho;
93                  int continuar = 0;
94                  while (continuar == 0){
95                      if (actual -> HijoIzquierdo != 0x0){
96                          actual = actual ->HijoIzquierdo;
97                      }else{
98                          continuar = 1;
99                      }
100                 }
101             }
102             return actual;
103         };
104
105
106         /**
107          * @brief funcion que devuelve el nodo de un dato
108          */
109         TypeNodo * NodeOf(Data &elDato){
110             ClassNode<Data> *actual = this->raiz;
111             TypeNodo * resultado;
112             resultado = actual;
113             /*cout << "\n\n\nmemoria de el dato" << &elDato << endl;
114             cout << "dato de el dato " << elDato.dato << endl;
115             cout << "memoria de la raiz" << this->raiz << endl;
116             cout << "dato de la raiz " <<~ *this->raiz->valor << endl;
117             cout << "memoria de la actual" << actual << endl;
118             cout << "dato de actual " << ~ *actual->valor << endl;*/
```

```
119                 int parar = 0;
120                 while (parar == 0)
121                 {
122                     if (elDato.dato == ~ *actual->valor){
123                         resultado = actual;
124                         //cout << "el nodo es igual al actual" << endl;
125                         parar  = 1;
126                     }else if (elDato.dato > ~*actual->valor){
127                         if (actual->HijoDerecho != 0x0){
128                             actual = actual->HijoDerecho;
129                         }else{
130                             //cout << "El nodo no existe" << endl;
131                         }
132                     }else if (elDato.dato < ~*actual->valor){
133                         if (actual->HijoIzquierdo != 0x0){
134                             actual = actual->HijoIzquierdo;
135                         }else{
136                             //cout << "El nodo no existe" << endl;
137                         }
138                     }
139                 }
140                 return resultado;
141 
142             };
143 
144 
145 
146 
147 
148 
149             /**
150              * @brief funcion que devuelve el dato de un nodo
151              */
152             Data * dataIn(TypeNodo &elNodo){
153                 //cout << "\n\n\ndata in" << endl;
154                 ClassNode<Data> *actual = this->raiz;
155                 Data * resultado;
156                 resultado = actual->valor;
157                 /*cout << "memoria de el nodo" << &elNodo << endl;
158                 cout << "dato de el nodo " << ~ *elNodo.valor << endl;
159                 cout << "memoria de la raiz" << this->raiz << endl;
160                 cout << "dato de la raiz " <<~ *this->raiz->valor << endl;
161                 cout << "memoria de la actual" << actual << endl;
162                 cout << "dato de actual " << ~ *actual->valor << endl;
163                 system ("sleep 1");*/
164 
165                 int parar = 0;
166                 while (parar == 0)
167                 {
168                     if (&elNodo == actual){
169                         resultado = actual->valor;
170                         //cout << "el nodo es igual al actual" << endl;
171                         parar  = 1;
172                     }else if (~ *elNodo.valor > ~*actual->valor){
173                         if (actual->HijoDerecho != 0x0){
174                             actual = actual->HijoDerecho;
175                         }else{
176                             //cout << "El nodo no existe" << endl;
177                         }
```

```
178                 }else if (~ *elNodo.valor < ~*actual->valor){
179                     if (actual->HijoIzquierdo != 0x0){
180                         actual = actual->HijoIzquierdo;
181                     }else{
182                         //cout << "El nodo no existe" << endl;
183                     }
184                 }
185             }
186             return resultado;
187 
188 
189             /*if (&elNodo == actual){
190                 cout << "el nodo es igual al actual" << endl;
191                 resultado = actual->valor;
192                 return resultado;
193             }else if (~ *elNodo.valor > ~*actual->valor){
194                 cout << "es mayor" << endl;
195                 if (actual->HijoDerecho != 0x0){
196                     actual = actual->HijoDerecho;
197                     resultado = dataIn(*actual);
198                 }else{
199                     cout << "El nodo no existe" << endl;
200                 }
201             }else if (~ *elNodo.valor < ~*actual->valor){
202                 cout << "es menor" << endl;
203                 if (actual->HijoIzquierdo != 0x0){
204                     actual = actual->HijoIzquierdo;
205                     resultado = dataIn(*actual);
206                 }else{
207                     cout << "El nodo no existe" << endl;
208                 }
209             }
210 
211             return resultado;*/
212 
213 
214         };
215 
216 
217         /**
218          * @brief funcion que borra el nodo de un arbol
219          */
220         void remove(Data &datoABorrar){
221             ClassNode<Data> *actual = this->raiz;
222             ClassNode<Data> *anterior = this->raiz;
223             ClassNode<Data> * elNodoBorrado = NodeOf(datoABorrar);
224             Data * elvalor = dataIn(*elNodoBorrado);
225             ClassNode<Data> * resultado = actual;
226             resultado = resultado;
227             ClassNode<Data> * mayorALaIzquierda = actual;
228             ClassNode<Data> * menorALaDerecha = actual;
229 
230 
231             int identificadorSoyRaiz = 0;
232             if (this->raiz == elNodoBorrado){
233                 cout << "el if del identificador" << endl;
234                 identificadorSoyRaiz = 1;
235             }
236             cout << "el dato es " << ~ datoABorrar << endl;
```

```
237                 cout << "la direccion de la raiz  " << this->raiz << endl;
238                 cout << "la direccion del dato es " << elNodoBorrado << endl;
239                 mayorALaIzquierda = this->largesToTheLeft(*elNodoBorrado);
240                 cout << "el mayor a la izquierda de " << datoABorrar.dato << " es "<< ~ *mayorALaIzq
241                 menorALaDerecha = this->smallesToTheRight(*elNodoBorrado);
242                 cout << "el menor a la derecha de " << datoABorrar.dato << " es "<< ~ *menorALaDerech
243                 //en esta parte se borra un nodo sin hijos
244                 if (~ *mayorALaIzquierda->valor == ~ *menorALaDerecha->valor){
245                     cout << "el if de iguales " << endl;
246                     if (identificadorSoyRaiz == 1){
247                         this->raiz = 0x0;
248                     }else{
249                         int continuar = 0;
250                         while (continuar == 0){
251                             if (elNodoBorrado == actual){
252                                 if (datoABorrar.dato > ~ * anterior->valor ){
253                                     anterior->HijoDerecho = 0x0;
254                                     delete(anterior->HijoDerecho);
255                                 }else{
256                                     anterior->HijoIzquierdo = 0x0;
257                                     delete(anterior->HijoIzquierdo);
258                                 }
259                                 continuar = 1;
260                             }else{
261                                 if (datoABorrar.dato > ~ *actual->valor){
262                                     anterior = actual;
263                                     actual  = actual ->HijoDerecho;
264                                 }else{
265                                     anterior = actual;
266                                     actual = actual ->HijoIzquierdo;
267                                 }
268                             }
269
270                         }
271                     }
272                 }else if (~ *mayorALaIzquierda->valor != datoABorrar.dato){//se borra un nodo con hij
273                     cout << "el if de mayor izquierda " << endl;
274                     cout << "este es el valor " << ~ * elvalor << endl;
275                     elvalor = dataIn(*mayorALaIzquierda);
276                     cout << " ahora este es el valor " << ~ * elvalor << endl;
277                     int continuar = 0;
278                     while (continuar == 0){
279                         if (elNodoBorrado == actual){
280                             if (identificadorSoyRaiz == 1){
281                                 this->remove(*elvalor);//se aplica el mismo proceso al nodo que se va
282                                 this->raiz->valor = elvalor;
283                             }else{
284                                 if (datoABorrar.dato > ~ * anterior->valor ){
285                                     this->remove(*elvalor);//se aplica el mismo proceso al nodo que s
286                                     anterior->HijoDerecho->valor = elvalor;
287                                 }else{
288                                     this->remove(*elvalor);//se aplica el mismo proceso al nodo que s
289                                     anterior->HijoIzquierdo->valor = elvalor;
290                                     //delete(anterior->HijoIzquierdo);
291                                 }
292                             }
293                             continuar = 1;
294                         }else{
295                             if (datoABorrar.dato > ~ *actual->valor){
```

```
296                             anterior = actual;
297                             actual  = actual ->HijoDerecho;
298                         }else{
299                             anterior = actual;
300                             actual = actual ->HijoIzquierdo;
301                         }
302                     }
303 
304                 }
305             }else if (~ *menorALaDerecha->valor != ~datoABorrar.dato){
306                 cout << "el if de menor derecha " << endl;
307                 cout << "este es el valor " << ~ * elvalor << endl;
308                 elvalor = dataIn(*menorALaDerecha);
309                 cout << " ahora este es el valor " << ~ * elvalor << endl;
310                 int continuar = 0;
311                 while (continuar == 0){
312                     if (elNodoBorrado == actual){
313                         if (identificadorSoyRaiz == 1){
314                             this->remove(*elvalor);//se aplica el mismo proceso al nodo que se va
315                             this->raiz->valor = elvalor;
316                         }else{
317                             if (datoABorrar.dato > ~ * anterior->valor ){
318                                 this->remove(*elvalor);//se aplica el mismo proceso al nodo que s
319                                 anterior->HijoDerecho->valor = elvalor;
320                             }else{
321                                 this->remove(*elvalor);//se aplica el mismo proceso al nodo que s
322                                 anterior->HijoIzquierdo->valor = elvalor;
323                                 //delete(anterior->HijoIzquierdo);
324                             }
325                         }
326                         continuar = 1;
327                     }else{
328                         if (datoABorrar.dato > ~ *actual->valor){
329                             anterior = actual;
330                             actual  = actual ->HijoDerecho;
331                         }else{
332                             anterior = actual;
333                             actual = actual ->HijoIzquierdo;
334                         }
335                     }
336 
337                 }
338             }else{
339                 cout << "el else " << endl;
340             }
341             /*cout << "\n\n\nmemoria de el dato" << &elDato << endl;
342             cout << "dato de el dato " << elDato.dato << endl;
343             cout << "memoria de la raiz" << this->raiz << endl;
344             cout << "dato de la raiz " <<~ *this->raiz->valor << endl;
345             cout << "memoria de la actual" << actual << endl;
346             cout << "dato de actual " << ~ *actual->valor << endl;*/
347             /*int parar = 0;
348             while (parar == 0)
349             {
350                 if (elDato.dato == ~ *actual->valor){
351                     resultado = actual;
352                     //cout << "el nodo es igual al actual" << endl;
353                     parar  = 1;
354                 }else if (elDato.dato > ~*actual->valor){
```

```
                    if (actual->HijoDerecho != 0x0){
                        actual = actual->HijoDerecho;
                    }else{
                        //cout << "El nodo no existe" << endl;
                    }
                }else if (elDato.dato < ~*actual->valor){
                    if (actual->HijoIzquierdo != 0x0){
                        actual = actual->HijoIzquierdo;
                    }else{
                        //cout << "El nodo no existe" << endl;
                    }
                }
            }
            return resultado;*/

        };

        ClassNode<Data> *raiz;
        //private :
        int items =0;



        /**
         * @brief funcion que devuelve si el dato existe o no
         * return bandera 1 si existe 0 si no
         */
        int Existe(Data &elDato){
            ClassNode<Data> *actual = this->raiz;
            int parar = 0;
            int bandera = 0;
            while (parar == 0)
            {
                if (elDato.dato == ~ *actual->valor){
                    bandera = 1;
                    parar  = 1;
                }else if (elDato.dato > ~*actual->valor){
                    if (actual->HijoDerecho != 0x0){
                        actual = actual->HijoDerecho;
                    }else{
                        parar = 1;
                    }
                }else if (elDato.dato < ~*actual->valor){
                    if (actual->HijoIzquierdo != 0x0){
                        actual = actual->HijoIzquierdo;
                    }else{
                        parar = 1;
                    }
                }
            }
            return bandera;

        };
```

```
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428         void preorden(ClassNode<Data> *n){
429             cout<<~*n->valor<<"-";
430
431             if (n->HijoIzquierdo != 0x0){
432                 preorden(n->HijoIzquierdo);
433             }
434             if (n->HijoDerecho != 0x0){
435                 //cout<<~*n->HijoDerecho->valor;
436                 preorden(n->HijoDerecho);
437             }
438         }
439
440         void InOrden(ClassNode<Data> *n){
441             int RaizImpresa=0;
442
443             if (n->HijoIzquierdo != 0x0){
444                 InOrden(n->HijoIzquierdo);
445             }
446             if (n->HijoIzquierdo == 0x0){
447                 cout<<~*n->valor<<"-";
448                 RaizImpresa=1;
449
450             }
451             if (RaizImpresa==0){
452                 cout<<~*n->valor<<"-";
453             }
454             if (n->HijoDerecho != 0x0){
455                 InOrden(n->HijoDerecho);
456             }
457
458
459         }
460
461         void PostOrden(ClassNode<Data> *n){
462             int RaizImpresa=0;
463
464             if (n->HijoIzquierdo != 0x0){
465                 PostOrden(n->HijoIzquierdo);
466             }
467             if (n->HijoIzquierdo == 0x0){
468                 cout<<~*n->valor<<"-";
469                 RaizImpresa=1;
470
471             }
472             if (n->HijoDerecho != 0x0){
```

```
473                     PostOrden(n->HijoDerecho);
474                 }
475 
476                 if (RaizImpresa==0){
477                     cout<<~*n->valor<<"-";
478                 }
479 
480 
481             }
482 
483 
484 
485 
486             /**
487              * @brief funcion recursiva que calcula la cantidad de niveles a partir de un nodo
488              * @param inicio es un puntero a la direccion de memoria
489              */
490             int calcularNiveles(ClassNode<Data> *inicio){
491                 int cantidadNivel = 0;
492                 int numeroHijos = 0;
493                 int nivelesIzquierda = 0;
494                 int nivelesDerecha =0;
495                 //calcula el numero de hijos
496                 if ((inicio->HijoIzquierdo !=0x0)&&(inicio->HijoDerecho!= 0x0)){
497                     numeroHijos = 2;
498                 }else if ((inicio->HijoIzquierdo !=0x0)||(inicio->HijoDerecho!= 0x0)){
499                     numeroHijos =1;
500                 }else if ((inicio->HijoIzquierdo ==0x0)&&(inicio->HijoDerecho== 0x0)){
501                     numeroHijos = 0;
502                 }
503 
504                 //devuelve el numero de niveles
505                 if (numeroHijos == 0){
506                     cantidadNivel=1;
507                     return cantidadNivel;
508                 }else if(numeroHijos == 1){
509                     cantidadNivel++;
510                     if (inicio->HijoDerecho != 0x0){
511                         inicio = inicio->HijoDerecho;
512                     }else{
513                         inicio = inicio->HijoIzquierdo;
514                     }
515                     cantidadNivel = cantidadNivel + calcularNiveles(inicio);
516                     return cantidadNivel;
517                 }else if(numeroHijos == 2){
518                     cantidadNivel++;
519                     nivelesIzquierda = calcularNiveles(inicio->HijoIzquierdo);
520                     nivelesDerecha = calcularNiveles(inicio->HijoDerecho);
521                     if ((nivelesIzquierda == 1)&&(nivelesDerecha == 1)){
522                         cantidadNivel = cantidadNivel +1;
523                     }else if (nivelesIzquierda > nivelesDerecha){
524                         cantidadNivel = cantidadNivel + nivelesIzquierda;
525                     }else{
526                         cantidadNivel = cantidadNivel + nivelesDerecha;
527                     }
528                     return cantidadNivel;
529                 }else{
530                     cout << "no deberia estar aqui" << endl;
531                     return cantidadNivel;
```

```
532                 }
533             }
534 
535 
536             /**
537              * @brief funcion recursiva que calcula la cantidad de nodos en un nivel
538              * @param inicio es un puntero a la direccion de memoria
539              * @param nivel es el nivel que se quiere
540              */
541             int calcularNodos(ClassNode<Data> *inicio , int nivel){
542                 int sumaNodos = 0;
543                 int numeroHijos = 0;
544                 //calcula el numero de hijos
545                 if ((inicio->HijoIzquierdo !=0x0)&&(inicio->HijoDerecho!= 0x0)){
546                     numeroHijos = 2;
547                 }else if ((inicio->HijoIzquierdo !=0x0)||(inicio->HijoDerecho!= 0x0)){
548                     numeroHijos =1;
549                 }else if ((inicio->HijoIzquierdo ==0x0)&&(inicio->HijoDerecho== 0x0)){
550                     numeroHijos = 0;
551                 }
552 
553                 //devuelve los nodos de una raiz
554                 if (nivel == 1){
555                     return 1;
556                 }else{
557                     nivel--;
558                     if (numeroHijos == 0){
559                         return 0;
560                     }else if (numeroHijos == 1){
561                         if (inicio->HijoDerecho != 0x0){
562                             inicio = inicio->HijoDerecho;
563                         }else{
564                             inicio = inicio->HijoIzquierdo;
565                         }
566                         sumaNodos = sumaNodos + calcularNodos(inicio , nivel);
567                         return sumaNodos;
568                     }else if (numeroHijos == 2){
569                         sumaNodos = sumaNodos + calcularNodos(inicio->HijoIzquierdo,nivel);
570                         sumaNodos = sumaNodos + calcularNodos(inicio->HijoDerecho,nivel);
571                         return sumaNodos;
572                         }else{
573                             cout << "no deberia estar aqui" << endl;
574                             return 0;
575                         }
576                 }
577             }
578             /**
579              * @brief funcion recursiva que devuelve la representacion de texto de un nivel
580              * @param inicio es un puntero a la direccion de memoria
581              * @param nivel es el nivel que se quiere
582              */
583             string devolverStringNivel(ClassNode<Data> *inicio , int nivel ,  int cantidadDatos){
584                 string linea = "";
585                 int numeroHijos = 0;
586 /*                if (nivel == 1){
587                     linea = linea + to_string(~ *inicio->valor);
588                     return linea;
589                 }else if(nivel == 2){
590                     if ((inicio->HijoIzquierdo !=0x0)&&(inicio->HijoDerecho!= 0x0)){
```

```
591                         linea = linea + to_string(~ *inicio->HijoIzquierdo->valor) + this->tabMI+this
592                     }else if ((inicio->HijoIzquierdo !=0x0)||(inicio->HijoDerecho!= 0x0)){
593                         if ((inicio->HijoIzquierdo == 0x0 )){
594                             linea = linea + "++" + this->tabMI+this->tabMD + to_string(~ *inicio->Hij
595                         }else{
596                             linea = linea + to_string(~ *inicio->HijoIzquierdo->valor) + this->tabMI-
597                         }
598                     }else if ((inicio->HijoIzquierdo ==0x0)&&(inicio->HijoDerecho== 0x0)){
599                     }
600 
601                 }else{
602                     if ((inicio->HijoIzquierdo !=0x0)&&(inicio->HijoDerecho!= 0x0)){
603                         numeroHijos = 2;
604                     }else if ((inicio->HijoIzquierdo !=0x0)||(inicio->HijoDerecho!= 0x0)){
605                         numeroHijos =1;
606                     }else if ((inicio->HijoIzquierdo ==0x0)&&(inicio->HijoDerecho== 0x0)){
607                         numeroHijos = 0;
608                     }
609 
610                     if (numeroHijos == 0){
611                         //linea = linea + this->tabMI;
612                         return linea;
613                     }else if (numeroHijos == 1){
614                         if (inicio->HijoDerecho != 0x0){
615                             inicio = inicio->HijoDerecho;
616                             //linea = linea + this->tabMI;
617                         }else{
618                             //inicio = inicio->HijoIzquierdo;
619                         }
620                         linea = linea + devolverStringNivel(inicio , nivel,cantidadDatos);
621                        // cout << "tengo un hijo devuelvo " << linea << endl;
622                         return linea;
623                     }else if (numeroHijos == 2){
624                         linea = linea + devolverStringNivel(inicio->HijoIzquierdo,nivel,cantidadDatos
625                         //linea = linea + this->tabMI + this->tabMD + this->tabMI+ this->tabMD;
626                         linea = linea + devolverStringNivel(inicio->HijoDerecho,nivel,cantidadDatos)
627                        // cout << "tengo dos hijos devuelvo " << linea << endl;
628                         return linea;
629                     }else{
630                         cout << "no deberia estar aqui" << endl;
631                         return linea;
632                     }
633                 }
634 */
635                 if ((inicio->HijoIzquierdo !=0x0)&&(inicio->HijoDerecho!= 0x0)){
636                     numeroHijos = 2;
637                 }else if ((inicio->HijoIzquierdo !=0x0)||(inicio->HijoDerecho!= 0x0)){
638                     numeroHijos =1;
639                 }else if ((inicio->HijoIzquierdo ==0x0)&&(inicio->HijoDerecho== 0x0)){
640                     numeroHijos = 0;
641                 }
642 
643 
644                 //cout << "nodo " << ~ *inicio->valor << endl;
645                 //devuelve los nodos de una raiz
646                 if (nivel == 1){
647                     linea = linea + to_string(~ *inicio->valor);
648                    // cout << "el nivel es uno entonces devuelvo " << linea << endl;
649                     return linea;
```

```
650                 }else{
651                     //cantidadDatos = calcularNodos(inicio,nivel-1);
652                     nivel--;
653                     if (numeroHijos == 0){
654                         linea = linea + this->tabMI;
655                         return linea;
656                     }else if (numeroHijos == 1){
657                         if (inicio->HijoDerecho != 0x0){
658                             inicio = inicio->HijoDerecho;
659                             linea = linea + this->tabMI;
660                         }else{
661                             inicio = inicio->HijoIzquierdo;
662                         }
663                         linea = linea + devolverStringNivel(inicio , nivel,cantidadDatos);
664                        // cout << "tengo un hijo devuelvo " << linea << endl;
665                         return linea;
666                     }else if (numeroHijos == 2){
667                         linea = linea + devolverStringNivel(inicio->HijoIzquierdo,nivel,cantidadDatos
668                         linea = linea + this->tabMI + this->tabMD + this->tabMI+ this->tabMD;
669                         linea = linea + devolverStringNivel(inicio->HijoDerecho,nivel,cantidadDatos)
670                        // cout << "tengo dos hijos devuelvo " << linea << endl;
671                         return linea;
672                     }else{
673                         cout << "no deberia estar aqui" << endl;
674                         return linea;
675                     }
676                 }
677 
678             }
679             /**
680              * @brief funcion que imprime un arbol
681              */
682             void print(){
683                 int cantidadNiveles = 0;
684                 int cantidadNodos = 0;
685                 cantidadNiveles = calcularNiveles(this->raiz);
686                 int contador = 1;
687                 string impresionFinal= "";
688                 int indicadrorEspacios = cantidadNiveles;
689                 while(contador <= cantidadNiveles){
690                     for (int i = 0; i < indicadrorEspacios; i++){
691                         impresionFinal = impresionFinal + this->tabMI + this->tabMD;
692                     }
693                     cantidadNodos = calcularNodos(this->raiz , contador);
694                     // cout << "en el nivel: " << cantidadNiveles << " tengo: " << cantidadNodos << "
695                     impresionFinal = impresionFinal + devolverStringNivel(this->raiz,contador, cantid
696                     cout << impresionFinal << endl;
697                     //cantidadNiveles--;
698                     contador++;
699                     impresionFinal = "";
700                     indicadrorEspacios --;
701 
702                 }
703 
704                 //cout << "fin funcion print" << endl;
705             //     ClassNode<Data> *last = raiz;
706             //     if (numeroDeEspacio != 0){
707             //         for (int i =0; i< numeroDeEspacio; i++){
708             //             cout << "+";
```

```
709             //          }
710             //          cout << ~ * last->valor << endl;
711             //          print((numeroDeEspacio-5) , nivel+1);
712             //      }
713             // }
714 
715             }
716             string tabMI = "    ";
717             string tabMD = "    ";
718     };
719 
720 
721     // ...
722 #endif
```

Listing 2: BinarySearchTree.h

```
#ifndef NODE_H
#define NODE_H


#include "./Includes.h"
using namespace std;
// ...
template <typename Dato>
    class ClassNode {
        public :
            ClassNode (Dato *valor){
                this->valor = valor;
            };
            ~ClassNode(){
                this -> valor = 0x0;
                //cout << "destructor del nodo" << endl;
            };

            ClassNode<Dato> * HijoIzquierdo = 0x0;
            ClassNode<Dato> * HijoDerecho = 0x0;
            Dato *valor;
    };
#endif
```

Listing 3: Node.h

```
#ifndef DATONOPRIMITIVO_H
#define DATONOPRIMITIVO_H

#include "./Includes.h"

template <typename TipoDato>
    class DatoNoPrimitivo{
    public:
        DatoNoPrimitivo(TipoDato valor){
            this->dato = valor;
        };
        ~DatoNoPrimitivo(){
            this->dato = 0;
            //cout << "destructor dato no primitivo" << endl;
        };

        TipoDato operator~ (){
            return this->dato;
        };

        TipoDato operator+(TipoDato elOtro){
            if (dato > elOtro){
                return dato;
            }else{
                return elOtro;
            }
        }

        TipoDato dato;
    };

#endif
```

Listing 4: DatoNoPrimitivo.h

```
/**
 *@file Includes.hpp/
 */
#ifndef INCLUDES_H
#define INCLUDES_H

    #include <iostream>
    #include "./Node.h"
    #include "./BinarySearchTree.h"
    #include "./DatoNoPrimitivo.h"
    #include <random>
    #include <ctime>
    #include <unistd.h>
    #include <chrono>


#endif
```

Listing 5: Includes.h