

IE-0217 Estructuras abstractas de datos y algoritmos para ingeniería

Lab 5: Binary Search Tree C++

Timna Belinda Brown Ramírez

B61254

`timna.brown@ucr.ac.cr`

`belindabrownr@gmail.com`

I-2019

Tabla de contenidos

1. Consideraciones	1
2. Abordaje y conclusiones	2
3. Apéndice	2
3.1. Código fuente	2

1. Consideraciones

- Laboratorio individual[1]
- Genere un reporte en \LaTeX que incluya su código, su abordaje para la solución y sus conclusiones.[2]

- Suba su código y su documentación (doxygen, README, INSTALL) al git respectivo de su grupo y el directorio del laboratorio. Use su número de carné a diferenciar los trabajos de su grupo.
- Cada estudiante debe subir el reporte a *Schoology*.
- Recuerde que por cada día tardía de entrega se le rebajaran puntos de acuerdo con la formula: 3^d donde $d > 1$ es la cantidad de día tardíos.

2. Abordaje y conclusiones

Para la resolución del laboratorio presentado, se realizaron una serie de clases y funciones que cumplen con los objetivos del laboratorio

Como conclusión se puso en práctica el uso del lenguaje C++, además, del uso de la lógica para cumplir el objetivo planteado.

Los documentos con el código que permite la resolución de este laboratorio se encuentran en el git del grupo 7 en la subdivisión "Lab5-B61254".[4]

3. Apéndice

3.1. Código fuente

[3]

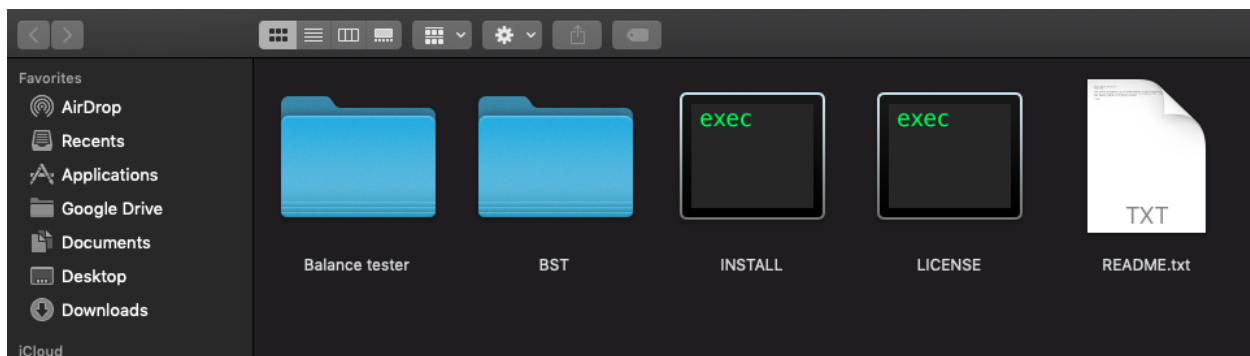


Figura 1: Dentro de Lab5

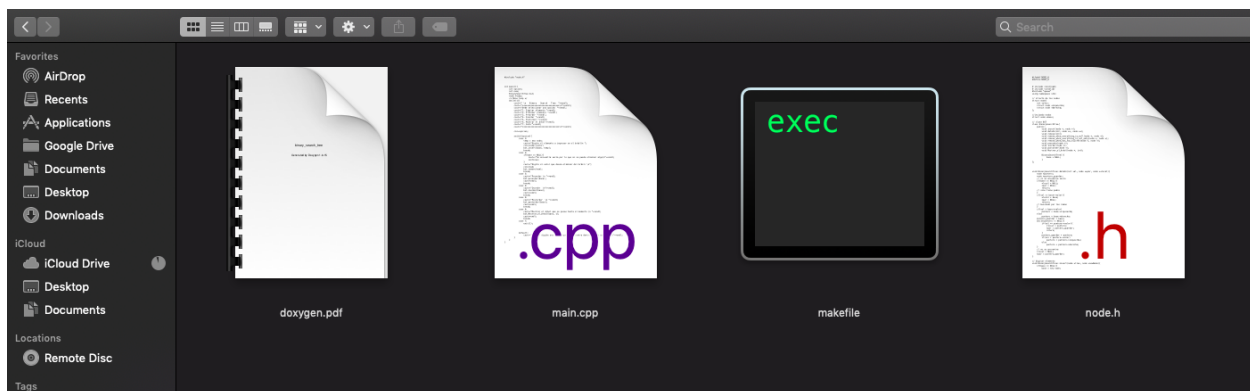


Figura 2: Dentro de BST

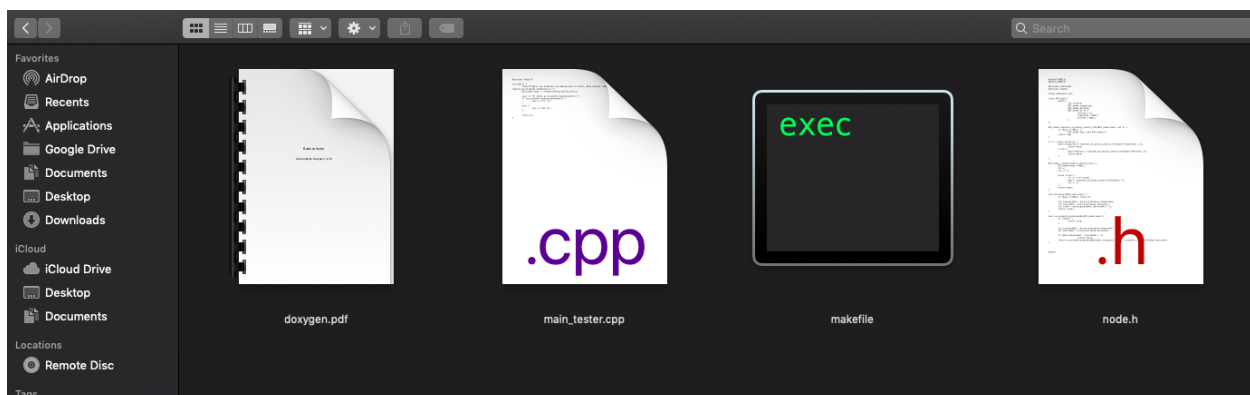


Figura 3: Dentro de Balance Tester

Todos tienen en común lo siguiente:

```
Binary Search Tree en C++  
Junio 2019  
  
Para ejecutar el programa es ir a la carpeta mediante su consola o terminal al folder  
en el que se encuentra el programa, ya sea "List" o la carpeta de "BST" o "Balance tester".  
Debe ingresar y digitar en la consola o terminal:  
  
$ make
```

Figura 4: Readme

```
Implementación de la estructura de datos BinarySearchTree mediante plantillas en C++  
  
Belinda Brown – timna.brown@ucr.ac.cr  
License Apache 2.0  
  
Se distribuye un Makefile con 3 reglas:  
  
* build: compila los fuentes.  
* clean: borra los binarios.  
* run: ejecuta un corrida de ejemplo.
```

Figura 5: Install

```
all: build run clean

build:
    g++ -g --std=c++11 -Wall *.cpp -o a.exe

run:
    ./a.exe

clean:
    rm a.exe
    rm -rf a.exe.dSYM
```

Figura 6: Makefile

```

main.cpp
26 switch(option){
27     case 1:
28         temp = new node;
29         cout<<"Digite el elemento a ingresar en el árbol\n ";
30         cin>>temp->valor;
31         bst.insert(base, temp);
32         break;
33     case 2:
34         if(base == NULL){
35             cout<<"Se encuentra vacía por lo que no se puede eliminar algo\n"<<endl;
36             continue;
37         }
38         cout<<"Digite el valor que desea eliminar del árbol: \n";
39         cin>>num;
40         bst.remove(num);
41         break;
42     case 3:
43         cout<<"Preorder \n "<<endl;
44         bst.preorder(base);
45         cout<<endl;
46         break;
47     case 4:
48         cout<<"Inorder \n"<<endl;
49         bst.inorder(base);
50         cout<<endl;
51         break;
52     case 5:
53         cout<<"Postorder \n "<<endl;
54         bst.postorder(base);
55         cout<<endl;
56         break;
57     case 6:
58         cout<<"Mostrar el árbol que se posee hasta el momento \n "<<endl;
59         bst.Mostrar_el_Arbol(base, 1);
60         cout<<endl;
61         break;
62     case 7:
63         exit(1);
64
65
66     default:
67         cout<<"\nFavor digite una opción que se encuentre dentro de las opciones \n"<<endl;
68 }
69 }
70 }
71 }
72

```

Figura 7: Código de main.cpp

```

node.h
1  #ifndef NODE_H
2  #define NODE_H
3
4
5  # include <iostream>
6  # include <cstdlib>
7  #include "queue"
8  using namespace std;
9
10 // structs de los nodos
11 struct node{
12     int valor;
13     struct node *izquierda;
14     struct node *derecha;
15 };
16
17 //inciando nodos
18 struct node *base;
19
20 // clase BST
21 class BinarySearchTree{
22     public:
23         void insert(node *, node *);
24         void dataIn(int, node **, node **);
25         void remove(int);
26         void remove_when_everything_is_null(node *, node *);
27         void remove_when_everything_is_not_empty(node *, node *);
28         void remove_when_has_two_hijoren(node *, node *);
29         void preorder(node *);
30         void inorder(node *);
31         void postorder(node *);
32         void Mostrar_el_Arbol(node *, int);
33
34         BinarySearchTree(){
35             base = NULL;
36         }
37 };
38
39
40
41 void BinarySearchTree::dataIn(int val, node **par, node **local){
42     node *puntero;
43     node *puntero_guardar;
44     // si se encuentra vacio
45     if(base == NULL){
46         *local = NULL;

```

Figura 8: Código de node.cpp

```

node.h
46     *local = NULL;
47     *par = NULL;
48     return;
49     // sino tiene padre
50 }
51 if(val == base->valor){
52     *local = base;
53     *par = NULL;
54     return;
55     // buscando por las ramas
56 }
57 if(val < base->valor)
58     puntero = base->izquierda;
59 else
60     puntero = base->derecha;
61 puntero_guardar = base;
62 while(puntero != NULL){
63     if(val == puntero->valor){
64         *local = puntero;
65         *par = puntero_guardar;
66         return;
67     }
68     puntero_guardar = puntero;
69     if(val < puntero->valor)
70         puntero = puntero->izquierda;
71     else
72         puntero = puntero->derecha;
73 }
74 // no se encuentra
75 *local = NULL;
76 *par = puntero_guardar;
77 }
78
79 // Digitar elemento
80 void BinarySearchTree::insert(node *tree, node *newNode){
81     if(base == NULL){
82         base = new node;
83         base->valor = newNode->valor;
84         base->izquierda = NULL;
85         base->derecha = NULL;
86         cout<<"Se creó la base del árbol"<<endl;
87         return;
88     }
89     if(tree->valor == newNode->valor){
90         cout<<"Ya se encuentra en el árbol, favor revisar si desea otro valor"<<endl;
91         return;

```

Figura 9: Código de node.cpp


```

node.h
90     cout<<"Ya se encuentra en el árbol, favor revisar si desea otro valor"<<endl;
91     return;
92 }
93 if(tree->valor > newNode->valor){
94     if(tree->izquierda != NULL){
95         insert(tree->izquierda, newNode);
96     }
97     else{
98         tree->izquierda = newNode;
99         (tree->izquierda)->izquierda = NULL;
100         (tree->izquierda)->derecha = NULL;
101         cout<<"Agregando valor a la izquierda \n "<<endl;
102         return;
103     }
104 }
105 else{
106     if(tree->derecha != NULL){
107         insert(tree->derecha, newNode);
108     }
109     else{
110         tree->derecha = newNode;
111         (tree->derecha)->izquierda = NULL;
112         (tree->derecha)->derecha = NULL;
113         cout<<"Agregando valor a la derecha \n"<<endl;
114         return;
115     }
116 }
117 }
118
119 void BinarySearchTree::remove(int val){
120     node *padre;
121     node *localation;
122     if(base == NULL){
123         cout<<"Está vacío, debe ingresar algo si desea eliminarlo \n"<<endl;
124         return;
125     }
126     dataIn(val, &padre, &localation);
127     if(localation == NULL){
128         cout<<"No se encuentra en el árbol \n"<<endl;
129         return;
130     }
131     if(localation->izquierda == NULL && localation->derecha == NULL)
132         remove_when_everything_is_null(padre, localation);
133     if(localation->izquierda == NULL && localation->derecha != NULL)
134         remove_when_everything_is_not_emty(padre, localation);
135     if(localation->izquierda != NULL && localation->derecha == NULL)
136         remove_when_everything_is_not_emty(padre, localation);

```

Figura 10: Código de node.cpp

```

node.h
134     remove_when_everything_is_not_emty(padre, localation);
135     if(localation->izquierda != NULL && localation->derecha == NULL)
136         remove_when_everything_is_not_emty(padre, localation);
137     if(localation->izquierda != NULL && localation->derecha != NULL)
138         remove_when_has_two_hijoren(padre, localation);
139     free(localation);
140 }
141
142 void BinarySearchTree::remove_when_everything_is_null(node *par, node *local){
143
144     if(par == NULL){
145         base = NULL;
146     }
147     else{
148         if(local == par->izquierda)
149             par->izquierda = NULL;
150         else
151             par->derecha = NULL;
152     }
153 }
154
155 void BinarySearchTree::remove_when_everything_is_not_emty(node *par, node *local){
156     node *hijo;
157     // esta a la izquierda o a la derecha
158     if(local->izquierda != NULL)
159         hijo = local->izquierda;
160     else
161         hijo = local->derecha;
162
163     if(par == NULL){
164         base = hijo;
165     }
166     else{
167
168         if(local == par->izquierda)
169             par->izquierda = hijo;
170         else
171             par->derecha = hijo;
172     }
173 }
174
175 void BinarySearchTree::remove_when_has_two_hijoren(node *par, node *local){
176     node *puntero;
177     node *puntero_guardar;
178     node *suc;
179     node *parsuc;
180

```

Figura 11: Código de node.cpp

```

node.h
179     node *parsuc;
180
181     puntero_guardar = local;
182     puntero = local->derecha;
183     while(puntero->izquierda != NULL){
184         puntero_guardar = puntero;
185         puntero = puntero->izquierda;
186     }
187     suc = puntero;
188     parsuc = puntero_guardar;
189
190     if(suc->izquierda == NULL && suc->derecha == NULL)
191         remove_when_everything_is_null(parsuc, suc);
192     else
193         remove_when_everything_is_not_empty(parsuc, suc);
194     if(par == NULL){
195         base = suc;
196     }
197     else{
198         if(local == par->izquierda)
199             par->izquierda = suc;
200         else
201             par->derecha = suc;
202     }
203     suc->izquierda = local->izquierda;
204     suc->derecha = local->derecha;
205 }
206
207 // Preorder
208 void BinarySearchTree::preorder(node *puntero){
209     if(base == NULL){
210         cout<<"Se encuentra vacio \n"<<endl;
211         return;
212     }
213     if(puntero != NULL){
214         cout<<puntero->valor<<" ";
215         preorder(puntero->izquierda);
216         preorder(puntero->derecha);
217     }
218 }
219
220 // Inorder
221 void BinarySearchTree::inorder(node *puntero){
222     if(base == NULL){
223         cout<<"Se encuentra vacio \n " <<endl;
224         return;
225     }

```

Figura 12: Código de node.cpp

```

node.h
220 // inorder
221 void BinarySearchTree::inorder(node *puntero){
222     if(base == NULL){
223         cout<<"Se encuentra vacio \n " << endl;
224         return;
225     }
226     if(puntero != NULL){
227         inorder(puntero->izquierda);
228         cout<<puntero->valor<<" ";
229         inorder(puntero->derecha);
230     }
231 }
232
233 //Postorder
234 void BinarySearchTree::postorder(node *puntero){
235     if(base == NULL){
236         cout<<"Se encuentra vacio \n" << endl;
237         return;
238     }
239     if(puntero != NULL){
240         postorder(puntero->izquierda);
241         postorder(puntero->derecha);
242         cout<<puntero->valor<<" ";
243     }
244 }
245
246 //se visualiza como arbol
247 void BinarySearchTree::Mostrar_el_Arbol(node *puntero, int level){
248     int i;
249     if(puntero != NULL){
250         Mostrar_el_Arbol(puntero->derecha, level+1);
251         cout<<endl;
252         if(puntero == base)
253             cout<<"Tenemos: \n ";
254         else{
255             // para diferentes niveles
256             for(i = 0; i < level; i++)
257                 cout<<" ";
258         }
259         cout<<puntero->valor;
260         Mostrar_el_Arbol(puntero->izquierda, level+1);
261     }
262 }
263
264
265 #endif
266

```

Figura 13: Código de node.cpp

```

      Binary    Search    Tree
*****

Debe seleccionar una opción:
1. Digitalar elemento
2. Eliminar elemento
3. Preorder
4. Inorder
5. Postorder
6. Mostrar el árbol
7. Salir
*****

1
Digite el elemento a ingresar en el árbol
45
Se creó la base del árbol

      Binary    Search    Tree
*****

Debe seleccionar una opción:
1. Digitalar elemento
2. Eliminar elemento
3. Preorder
4. Inorder
5. Postorder
6. Mostrar el árbol
7. Salir
*****

1
Digite el elemento a ingresar en el árbol
76
Agregando valor a la derecha

      Binary    Search    Tree
*****

```

Figura 14: Resultados de BST

```

6. Mostrar el árbol
7. Salir
*****

6
Mostrar el árbol que se posee hasta el momento

          456
        78
      76
Tenemos:
45

  Binary  Search  Tree
*****

Debe seleccionar una opción:
1. Digitar elemento
2. Eliminar elemento
3. Preorder
4. Inorder
5. Postorder
6. Mostrar el árbol
7. Salir
*****

3
Preorder

45  76  456  78

  Binary  Search  Tree
*****

Debe seleccionar una opción:
1. Digitar elemento
2. Eliminar elemento
3. Preorder
4. Inorder

```

Figura 15: Resultados de BST

```

6. Mostrar el árbol
7. Salir
*****

4
Inorder

45  76  78  456

    Binary    Search    Tree
*****

Debe seleccionar una opción:
1. Digitar elemento
2. Eliminar elemento
3. Preorder
4. Inorder
5. Postorder
6. Mostrar el árbol
7. Salir
*****

5
Postorder

78  456  76  45

    Binary    Search    Tree
*****

Debe seleccionar una opción:
1. Digitar elemento
2. Eliminar elemento
3. Preorder
4. Inorder
5. Postorder
6. Mostrar el árbol
7. Salir
*****

```

Figura 16: Resultados de BST

```

2
Digite el valor que desea eliminar del árbol:
54
No se encuentra en el árbol

      Binary    Search    Tree
*****

Debe seleccionar una opción:
1. Digitar elemento
2. Eliminar elemento
3. Preorder
4. Inorder
5. Postorder
6. Mostrar el árbol
7. Salir
*****

2
Digite el valor que desea eliminar del árbol:
45

      Binary    Search    Tree
*****

Debe seleccionar una opción:
1. Digitar elemento
2. Eliminar elemento
3. Preorder
4. Inorder
5. Postorder
6. Mostrar el árbol
7. Salir
*****

6
Mostrar el árbol que se posee hasta el momento

          456
         /  \
        /    \
       /      \
      /        \
     /          \
    /            \
   /              \
  /                \
 /                  \
/                    \
456                  78

Tenemos:
  76

      Binary    Search    Tree
*****

Debe seleccionar una opción:
1. Digitar elemento
2. Eliminar elemento
3. Preorder
4. Inorder
5. Postorder
6. Mostrar el árbol
7. Salir
*****

```

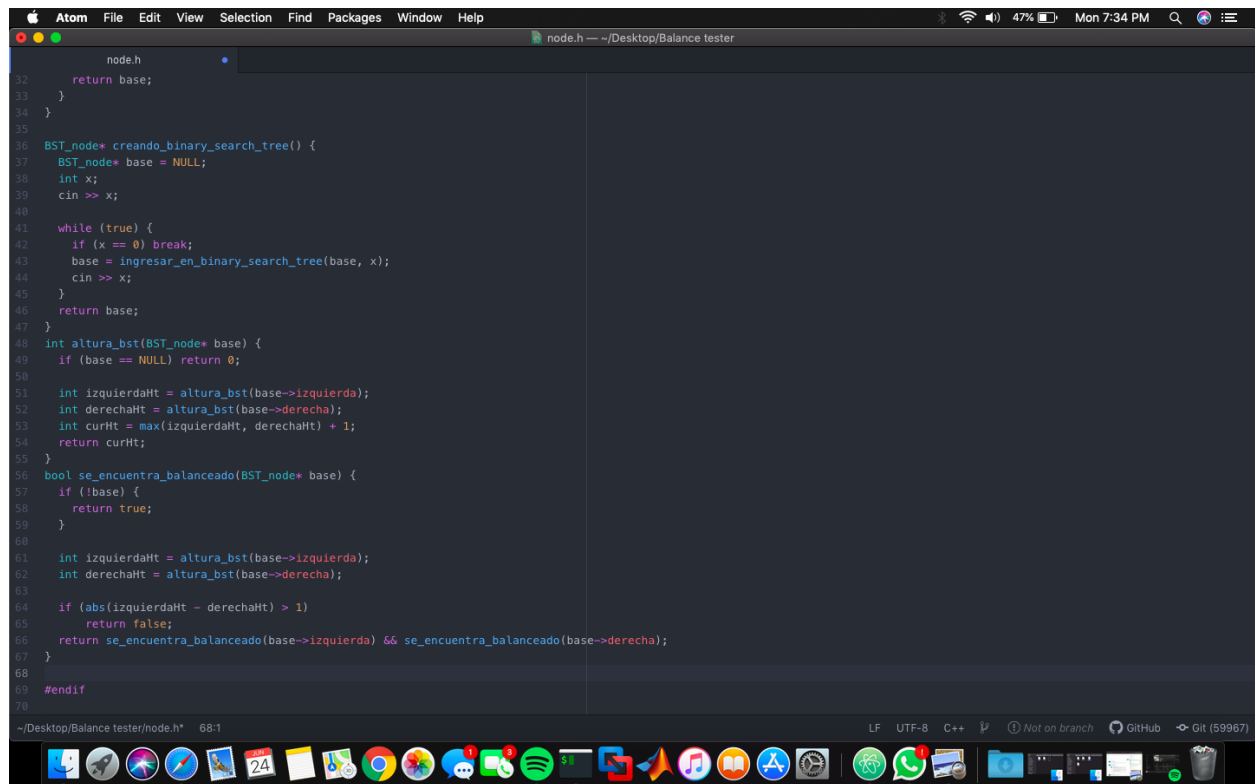
Figura 17: Resultados de BST


```

node.h
1  #ifndef NODE_H
2  #define NODE_H
3
4  #include <iostream>
5  #include <cmath>
6
7  using namespace std;
8
9  class BST_node {
10 public:
11     int valores;
12     BST_node* izquierda;
13     BST_node* derecha;
14     BST_node(int d) {
15         valores = d;
16         izquierda = NULL;
17         derecha = NULL;
18     }
19 };
20
21 BST_node* ingresar_en_binary_search_tree(BST_node* base, int x) {
22     if (base == NULL) {
23         BST_node* tmp = new BST_node(x);
24         return tmp;
25     }
26
27     if (x < base->valores) {
28         base->izquierda = ingresar_en_binary_search_tree(base->izquierda , x);
29         return base;
30     } else {
31         base->derecha = ingresar_en_binary_search_tree(base->derecha, x);
32         return base;
33     }

```

Figura 18: Código de en Balance Tester node.cpp

A screenshot of the Atom text editor interface. The top menu bar includes 'Atom', 'File', 'Edit', 'View', 'Selection', 'Find', 'Packages', 'Window', and 'Help'. The title bar shows 'node.h' and the file path '~/Desktop/Balance tester'. The code in 'node.h' defines functions for creating, inserting, and checking the balance of a Binary Search Tree (BST). The code includes comments in Spanish. The status bar at the bottom shows the file path, line number 68, and various icons for file encoding, tabs, and Git status.

```
node.h
32     return base;
33 }
34 }
35
36 BST_node* creando_binary_search_tree() {
37     BST_node* base = NULL;
38     int x;
39     cin >> x;
40
41     while (true) {
42         if (x == 0) break;
43         base = ingresar_en_binary_search_tree(base, x);
44         cin >> x;
45     }
46     return base;
47 }
48
49 int altura_bst(BST_node* base) {
50     if (base == NULL) return 0;
51
52     int izquierdaHt = altura_bst(base->izquierda);
53     int derechaHt = altura_bst(base->derecha);
54     int curHt = max(izquierdaHt, derechaHt) + 1;
55     return curHt;
56 }
57
58 bool se_encuentra_balanceado(BST_node* base) {
59     if (!base) {
60         return true;
61     }
62
63     int izquierdaHt = altura_bst(base->izquierda);
64     int derechaHt = altura_bst(base->derecha);
65
66     if (abs(izquierdaHt - derechaHt) > 1)
67         return false;
68     return se_encuentra_balanceado(base->izquierda) && se_encuentra_balanceado(base->derecha);
69 }
70 #endif
```

Figura 19: Código de en Balance Tester node.cpp

A screenshot of the Atom text editor interface showing the 'main_tester.cpp' file. The code includes the 'node.h' header and implements the 'main' function. It prompts the user to enter elements for the tree, calls the 'creando_binary_search_tree' function, and then checks if the tree is balanced using 'se_encuentra_balanceado'. The status bar at the bottom shows the file path, line number 18, and various icons for file encoding, tabs, and Git status.

```
main_tester.cpp
1  #include "node.h"
2
3  int main() {
4      cout<<"Digite los elementos que desea para el árbol, debe digitar como último elemento 0 para detener el ingreso
5      de posibles elementos:\n ";
6      BST_node* base = creando_binary_search_tree();
7
8      cout << "El árbol se encuentra balanceado?\n ";
9      if (se_encuentra_balanceado(base)) {
10         cout << "Sí \n";
11     }
12     else {
13         cout << "No \n";
14     }
15
16     return 0;
17 }
18
```

Figura 20: Código de main_tester.cpp

```
g++ -g --std=c++11 -Wall *.cpp -o a.exe
./a.exe
Digite los elementos que desea para el árbol, debe digitar como último elemento 0 para detener el ingreso de posibles elementos:
1
8
23
47
0
El árbol se encuentra balanceado?
No
rm a.exe
rm -rf a.exe.dSYM
Belindas-MacBook-Air:Balance tester belindabrown$ make
g++ -g --std=c++11 -Wall *.cpp -o a.exe
./a.exe
Digite los elementos que desea para el árbol, debe digitar como último elemento 0 para detener el ingreso de posibles elementos:
45
2
0
El árbol se encuentra balanceado?
Sí
rm a.exe
rm -rf a.exe.dSYM
```

Figura 21: Resultados de Balance tester

Referencias

- [1] Kroah-Hartman G Corbet. J, Rubini. A. *Linux Coding*. O'Reilly books, 1998.
- [2] Computer Science Labs. *Tecnology- commands*. O'Reilly books, 2018.
- [3] Mark Summerfield. *Programming in Python 3: A Complete Introduction to the Python Language*. Anaya Multimedia, 2009.
- [4] A. M. Turing. *On computable numbers with an application to the Entscheidungs problem*. Proceedings of the london mathematical society, 1997.