EIE
Escuela de
Ingeniería Eléctrica

Universidad de Costa Rica

Facultad de Ingeniería

Escuela de Ingeniería Eléctrica

UNIVERSIDAD DE COSTA RICA

`IE-0217` Estructuras abstractas de datos y algorítmos 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

- 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 4. Las cuales incluyen los métodos de ordenamiento y búsqueda genéricos y hacer listas enlazadas.

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

Los documentos con el cóodigoque permite la resolución de este laboratorio se encuentran en el git del grupo 7 en la subdivisión "Lab4-B61254".[4]

3. Apéndice

3.1. Código fuente

[3]

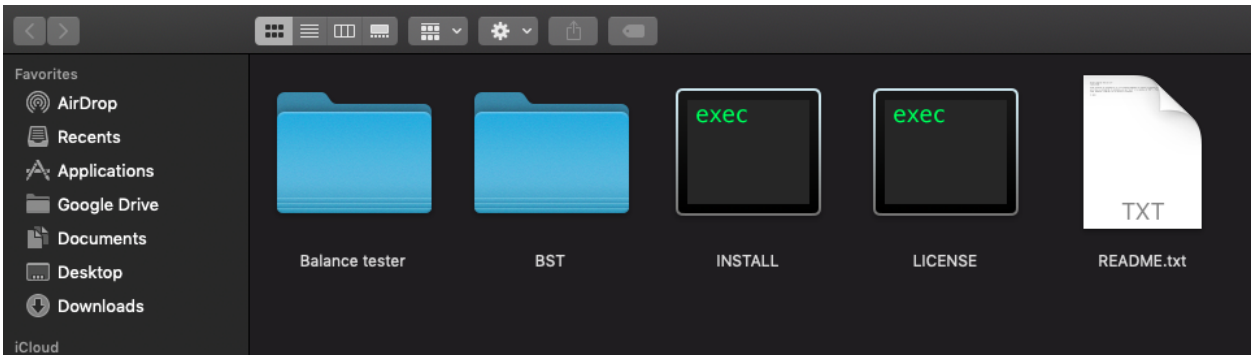

Figura 1: Dentro de Lab5

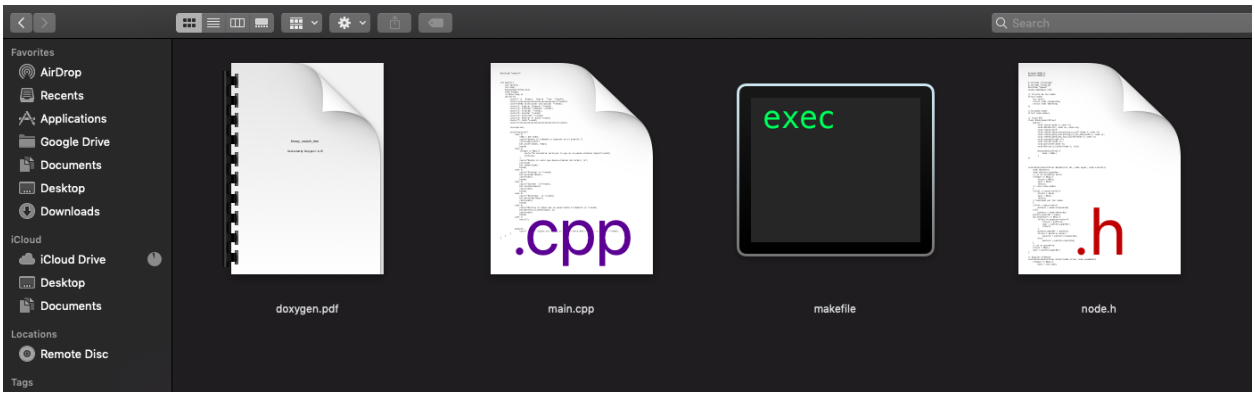

Figura 2: Dentro de BST

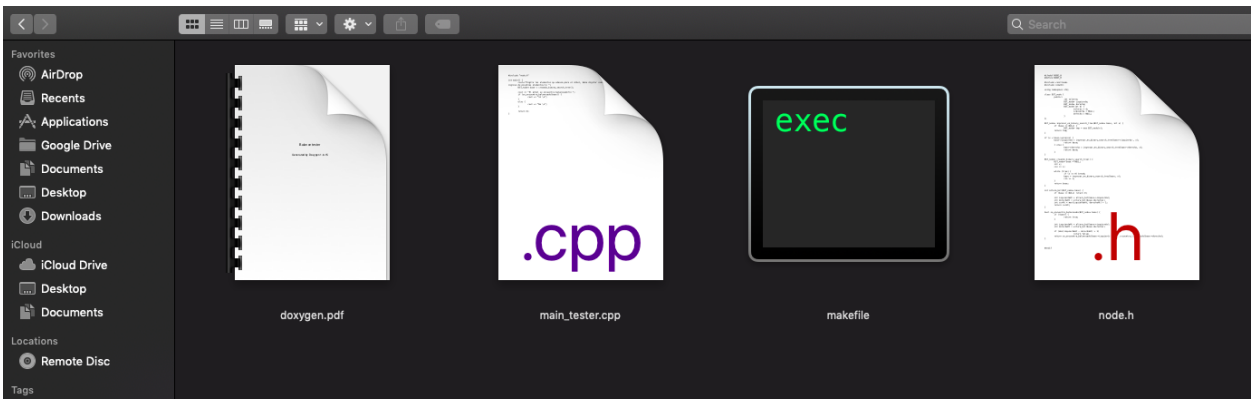

Figura 3: Dentro de Balance Tester

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

```
        switch(opcion){
            case 1:
                temp = new node;
                cout<<"Digite el elemento a ingresar en el árbol\n ";
                cin>>temp->valor;
                bst.insert(base, temp);
                break;
            case 2:
                if(base == NULL){
                    cout<<"Se encuentra vacía por lo que no se puede eliminar algo\n"<<endl;
                    continue;
                }
                cout<<"Digite el valor que desea eliminar del árbol: \n";
                cin>>num;
                bst.remove(num);
                break;
            case 3:
                cout<<"Preorder \n "<<endl;
                bst.preorder(base);
                cout<<endl;
                break;
            case 4:
                cout<<"Inorder  \n"<<endl;
                bst.inorder(base);
                cout<<endl;
                break;
            case 5:
                cout<<"Postorder  \n "<<endl;
                bst.postorder(base);
                cout<<endl;
                break;
            case 6:
                cout<<"Mostrar el árbol que se posee hasta el momento \n "<<endl;
                bst.Mostrar_el_Arbol(base, 1);
                cout<<endl;
                break;
            case 7:
                exit(1);



            default:
                cout<<"\nFavor digite una opción que se encuentre dentro de las opciones \n"<<endl;
        }
    }
}
```

Figura 7: Código de main.cpp

node.h

```
#ifndef NODE_H
#define NODE_H


# include <iostream>
# include <cstdlib>
#include "queue"
using namespace std;

// structs de los nodos
struct node{
    int valor;
    struct node *izquierda;
    struct node *derecha;
};

//inciando nodos
struct node *base;

// clase BST
class BinarySearchTree{
    public:
        void insert(node *, node *);
        void dataIn(int, node **, node **);
        void remove(int);
        void remove_when_everything_is_null(node *, node *);
        void remove_when_everything_is_not_emty(node *, node *);
        void remove_when_has_two_hijoren(node *, node *);
        void preorder(node *);
        void inorder(node *);
        void postorder(node *);
        void Mostrar_el_Arbol(node *, int);

        BinarySearchTree(){
            base = NULL;
        }
};



void BinarySearchTree::dataIn(int val, node **par, node **local){
    node *puntero;
    node *puntero_guardar;
    // si se encuentra vacio
    if(base == NULL){
        *local = NULL;
```

Figura 8: Código de node.cpp

node.h

```
        *local = NULL;
        *par = NULL;
        return;
    // sino tiene padre
    }
    if(val == base->valor){
        *local = base;
        *par = NULL;
        return;
    // buscando por las ramas
    }
    if(val < base->valor)
        puntero = base->izquierda;
    else
        puntero = base->derecha;
    puntero_guardar = base;
    while(puntero != NULL){
        if(val == puntero->valor){
            *local = puntero;
            *par = puntero_guardar;
            return;
        }
        puntero_guardar = puntero;
        if(val < puntero->valor)
            puntero = puntero->izquierda;
        else
            puntero = puntero->derecha;
    }
    // no se encuentra
    *local = NULL;
    *par = puntero_guardar;
}

// Digitar elemento
void BinarySearchTree::insert(node *tree, node *newNode){
    if(base == NULL){
        base = new node;
        base->valor = newNode->valor;
        base->izquierda = NULL;
        base->derecha = NULL;
        cout<<"Se creó la base del árbol"<<endl;
        return;
    }
    if(tree->valor == newNode->valor){
        cout<<"Ya se encuentra en el árbol, favor revisar si desea otro valor"<<endl;
        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 vacio, 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_emty(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 encuenta 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 encuenta 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 encuenta 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 encuenta 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 nivelos
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

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

Figura 17: Resultados de BST

node.h

```cpp
#ifndef NODE_H
#define NODE_H

#include <iostream>
#include <cmath>

using namespace std;

class BST_node {
  public:
    int valores;
    BST_node* izquierda;
    BST_node* derecha;
    BST_node(int d) {
      valores = d;
      izquierda = NULL;
      derecha = NULL;
    }
};

BST_node* ingresar_en_binary_search_tree(BST_node* base, int x) {
  if (base == NULL) {
    BST_node* tmp = new BST_node(x);
  return tmp;
}

if (x < base->valores) {
  base->izquierda = ingresar_en_binary_search_tree(base->izquierda , x);
    return base;
  } else {
    base->derecha = ingresar_en_binary_search_tree(base->derecha, x);
    return base;
  }
```

Figura 18: Código de en Balance Tester node.cpp

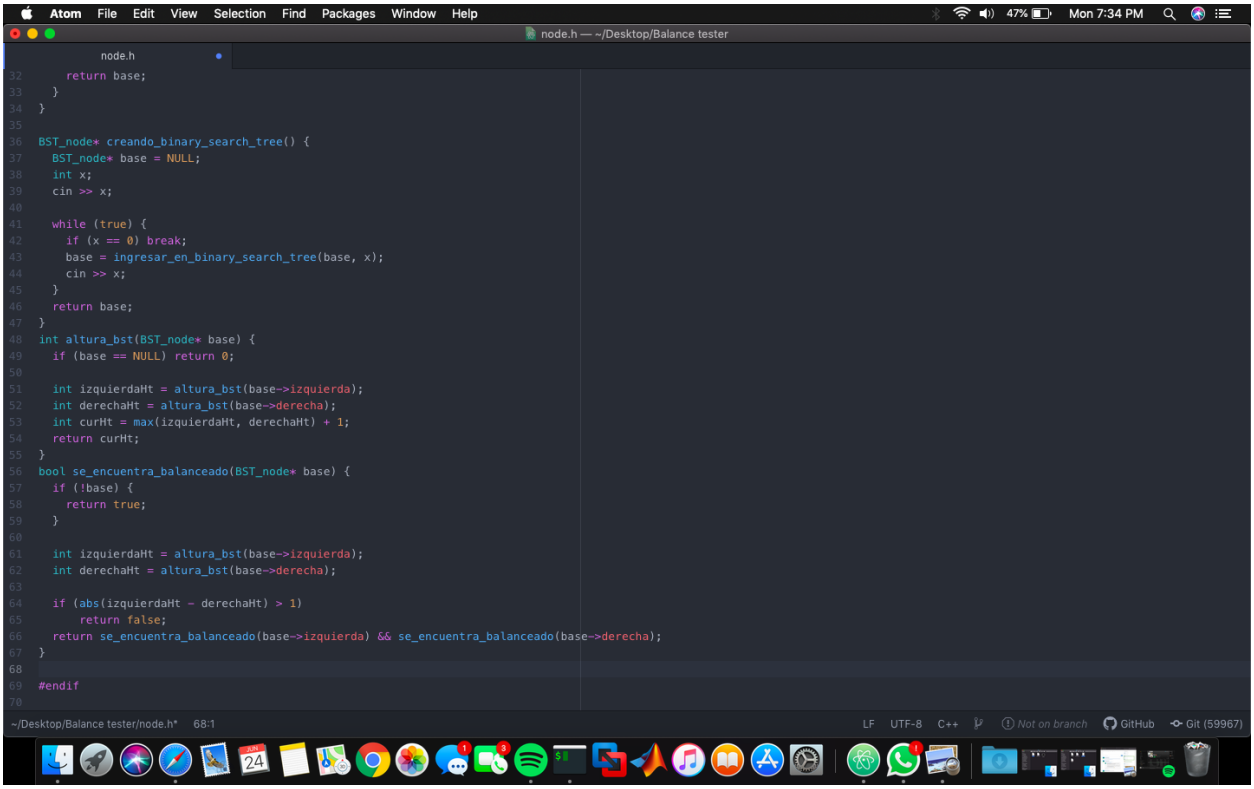

Figura 19: Código de en Balance Tester node.cpp

Figura 20: Código de main_tester.cpp

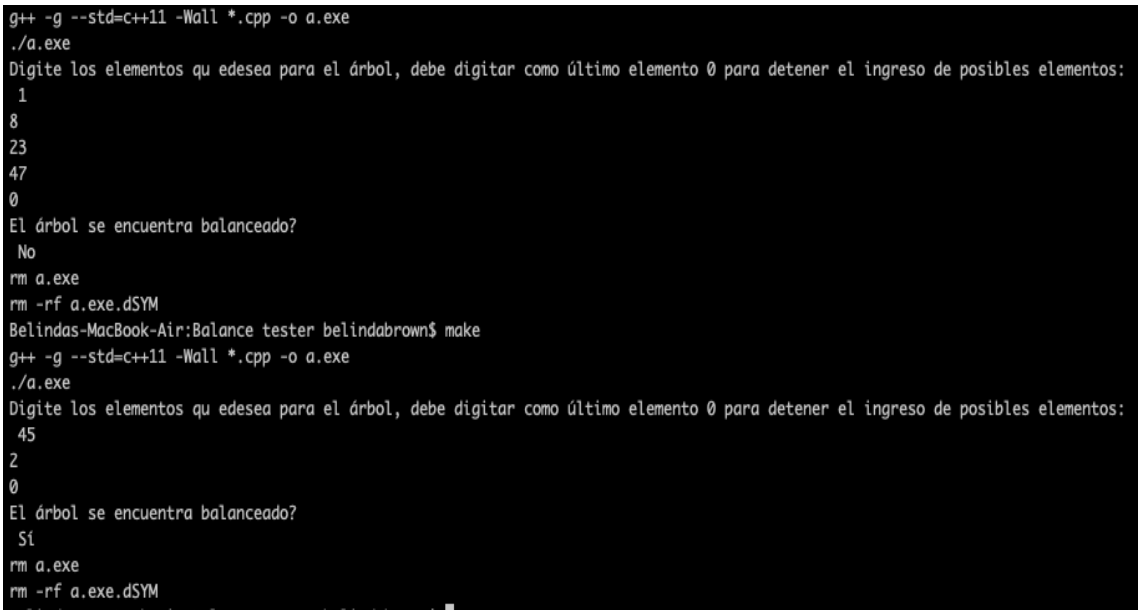

Figura 21: Resultados de Balance tester

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.