

IE-0217 Estructuras abstractas de datos y algoritmos para ingeniería

Proyecto 1: ABB vs Árboles Rojo y Negro

Timna Belinda Brown Ramírez

B61254

`timna.brown@ucr.ac.cr`

`belindabrownr@gmail.com`

I-2019

Tabla de contenidos

1. Introducción	2
2. Reseña del programa	2
3. Funcionamiento del programa	2
4. Experimentos realizados	3
5. Marco teórico	3
6. Resultados obtenidos	4
6.1. Método 1: <code>string recorrer_arbol_por_ancho()</code>	5
6.2. Método 2: <code>bool arbol_esta_lleno()</code>	8
6.3. Método 3: <code>bool arbol_esta_completo()</code>	8

6.4. Método 4: bool eliminar_nodo(int)	9
6.5. Método 5: bool compara_cant_elem(ArbolABB*)	10
6.6. Método 6: bool de_altura_son_iguales(ArbolABB*)	11
7. Conclusiones	18
7.1. Comparación de tiempo de método insertar	18
7.2. Comparación de tiempo de método eliminar	18
7.3. Compración tiempo de implementación de todas las funciones	19
7.4. Gráfico función de tiempo	21
8. Apéndice	22
8.1. Código fuente	23

1. Introducción

Los árboles en la programación representan estructuras no-lineales y dinámicas de datos. Dinámicas, puesto que la estructura del árbol puede variar de acuerdo se ejecute el programa. Poseen la cualidad de no ser lineales, debido que cada elemento del árbol puede tener su descendencia. Algunas de las aplicaciones que poseen son por ejemplo priorización de trabajos, descripción de expresiones matemáticas, algoritmos de compresión.

2. Reseña del programa

El objetivo de este proyecto es crear un algoritmo que desarrolle el funcionamiento de los Árboles Búsqueda Binaria (ABB) y los Árboles Rojo y Negro con el fin de realizar un análisis de la complejidad de cada uno y su debida comparación.[2]

3. Funcionamiento del programa

El proyecto fue desarrollado en C++, está conformada por una carpeta nombrada Proyecto1_B61254 dentro del cual existen dos más una con los archivos correspondientes al contenido del desarrollo de las funciones.[1]

El algoritmo de negro y rojo toma como fuente un los datos de entrada por el usuario y el de búsqueda binaria corre pruebas ya creadas.

4. Experimentos realizados

Se realizaron pruebas para asegurar que el código no tuviera errores de sintáxis por lo que se compiló, sucesivamente se realizaron las modificaciones necesarias para que ejecutara adecuadamente las instrucciones generales. Con lo que respecta al sistema operativo en el que funciona, es ejecutable sobre cualquier plataforma que interprete C++ por medio de su propio intérprete. El programa recibe las instrucciones realizadas por medio del teclado con el fin de volver más accesible su ejecución.[4]

Por otra parte, respecto a las variables de respuesta. Por lo mencionado, fue necesaria la creación y consumación de pruebas experimentales tal que se validan mediante test unitarios ya fue necesario descomponer las funciones de los árboles en comportamientos cualificables. [3]

5. Marco teórico

Un ABB tiene la característica de que todos los subelementos de un nodo a su izquierda están los menores y a su derecha los mayores almacenados. Por otr lado, la complejidad de los árboles negro y rojo es mayor, ya que es un árbol de búsqueda binaria de datos con asignación de colores.

Poseen las siguientes características: Los nodos tienen un un color propio ya sea rojo o negro. Aparte de todo lo que cumple un árbol BB se debe cumplir lo siguiente:

- Nodo rojo o negro
- Raíz negra
- Hojas negras
- Hijos de todo nodo rojo son negros
- Cada camino tiene la misma cantidad de nodos negros y esto se denomina altura del árbol

6. Resultados obtenidos

Con lo que respecta al algoritmo generado para crear el ABB se hizo un tester con varios valores definidos y funciones aplicadas a los mismos los cuales suceden a continuación:

```
Belindas-MacBook-Air:Arboles_ABB belindabrown$ make
g++ -g --std=c++11 -Wall *.cpp -o a.exe
./a.exe

*****

Tester de arboles ABB #1

*****

-----

Agregando valores

-----

Agregar 35
Agregar 12
Agregar 50
Agregar 8
Agregar 17
Agregar 32
Agregar 53
Agregar 3
Agregar 10
Agregar 29
Agregar 57

-----

Comprobando cantidad de elementos entre árboles

-----

Como resultado obtenemos que:
No son semejantes

-----

Prueba de altura entre árboles

-----

Como resultado obtenemos que:
No son iguales en altura
```

Figura 1: Resultados ABB

6.1. Método 1: `string recorrer_arbol_por_ancho()`

Método que retorna una hilera con información de los nodos contenidos en el árbol ABB por niveles:

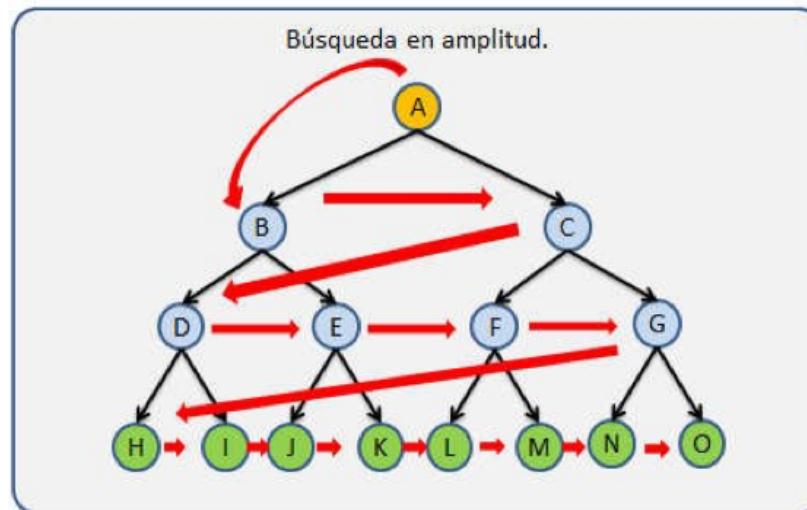


Figura 2: `recorrer_arbol_por_ancho()`

Funciona mediante el recorrido realizado por las flechas de color rojo, si se recuerda el funcionamiento del ABB todos los elementos almacenados en el subárbol izquierdo de cualquier nodo A son menores que el elemento almacenado en A, y todos los elementos almacenados en el subárbol derecho de A son mayores que el elemento almacenado en A. Se recorre de raíz A seguido por su subárbol izquierdo menor B hasta C, notamos que este comportamiento se realiza recursivamente hasta que ya no hayan más nodos.

De acuerdo con lo que se observa en la Figura 3 se puede deducir el orden y la estructura del árbol. La raíz contiene el valor 35, a su izquierda siendo B seguido por C a su derecha en recursividad hasta que ya no hayan más nodos, siendo esta la descripción mostrada anteriormente.

```

-----

Prueba de espacio en los árboles

-----

Realizando un recorrido a lo ancho del primer árbol
35 12 50 8 17 53 3 10 32 57 29

Realizando un recorrido a lo ancho del segundo árbol
53 12 57 8 50 249856000 3 10 17 32 29
Como resultado obtenemos que:
No esta lleno

Como resultado obtenemos que:
No esta completo

-----

Eliminado hojas

-----

Borrando hoja 29
1
Realizando un recorrido a lo ancho
35 12 50 8 17 53 3 10 32 57
Borrando nodo con 0 hijo 53
1
Realizando un recorrido a lo ancho
35 12 50 8 17 57 3 10 32
Borrando nodo con dos hijos 12
1
Realizando un recorrido a lo ancho
35 8 50 3 17 57 10 32

*****

Tester de arboles ABB #2

*****

El arbol (29,11,28,18,31):
No esta lleno

```

Figura 3: Resultados ABB

Como resultado de la Figura 3 obtenemos el siguiente árbol:

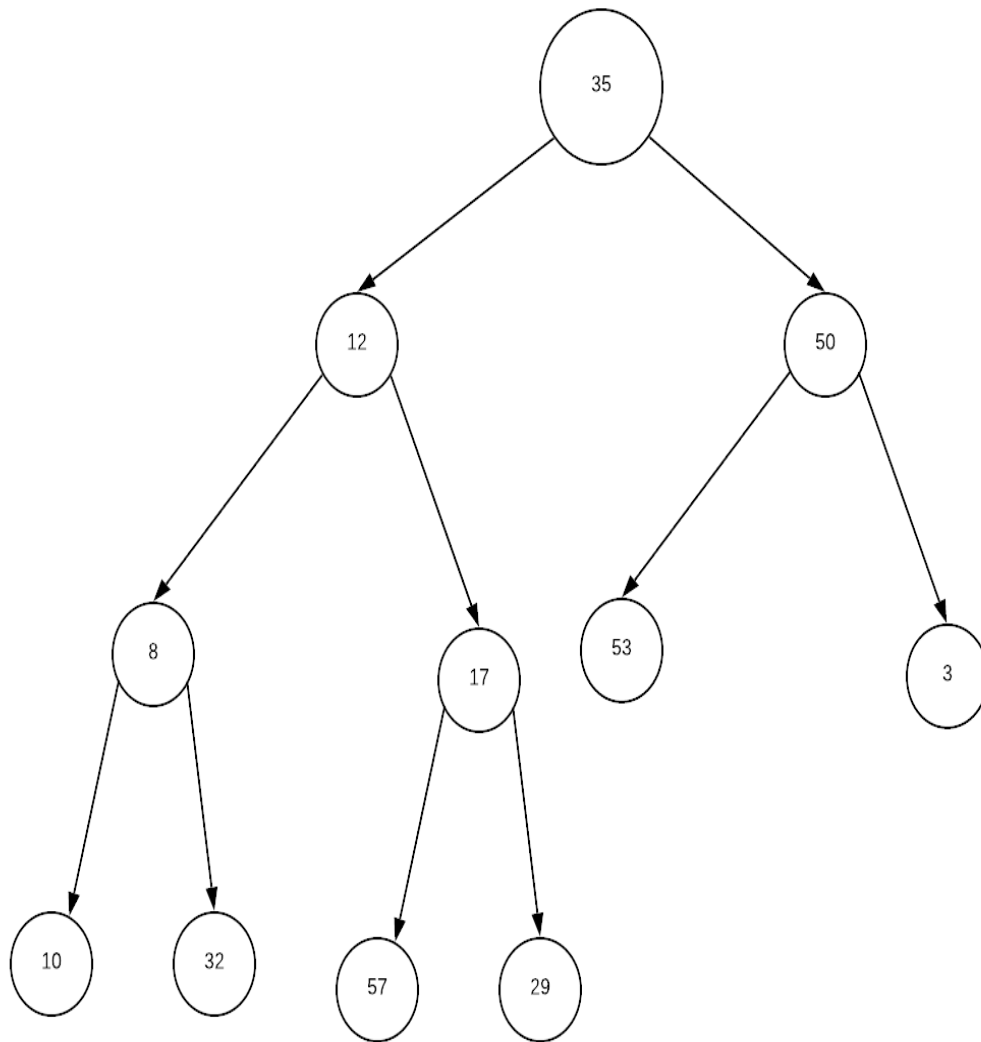


Figura 4: Árbol ABB 1

6.2. Método 2: bool arbol_esta_lleno()

Método que revisa la estructura del árbol y retorna verdadero si el árbol cumple la condición.

❖ Un árbol binario lleno es aquel en que cada nodo es un nodo interno con dos hijos no vacíos, o una hoja.

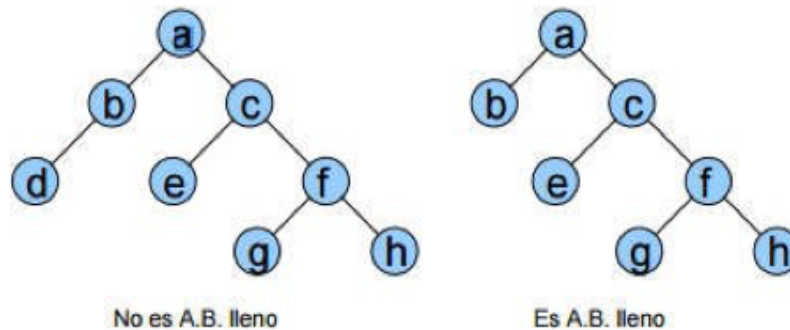


Figura 5: arbol_esta_lleno()

6.3. Método 3: bool arbol_esta_completo()

Método que revisa la estructura del árbol y retorna verdadero si el árbol cumple la condición de árbol completo vista en clase.

❖ Un árbol binario completo tiene una forma restringida, que se obtiene al ser llenado de izquierda a derecha. En un A.B. Completo de altura d , todos los niveles, excepto posiblemente el nivel d están completamente llenos.

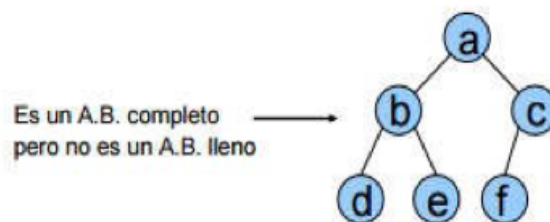


Figura 6: arbol_esta_lleno()

La Figura 4, se analiza si el árbol está en completo con lo que respecta a que el nivel de las hojas (nodos sin hijos) cada nodo tenga dos hojas. Se considera que el nodo con el valor de 53 no tiene hojas, por lo que el mismo es una hoja quedando un nivel más en comparación con el que se encuentra la hoja con el valor 29. Asimismo sucede con el valor 3, por lo que se concluye que el árbol no está completo. Se muestra en la Figura 5 y 6.

6.4. Método 4: bool eliminar_nodo(int)

Recibe mediante la entrada de un valor de un elemento y lo elimina del ABB. Se aplica el criterio menor de los Mayores (mM) para realizar la elección que el nodo a eliminar cumpla con las siguiente condición: en el caso en que el nodo a eliminar tenga dos subárboles no nulos. El método retorna si la acción pudo ser realizada o no.

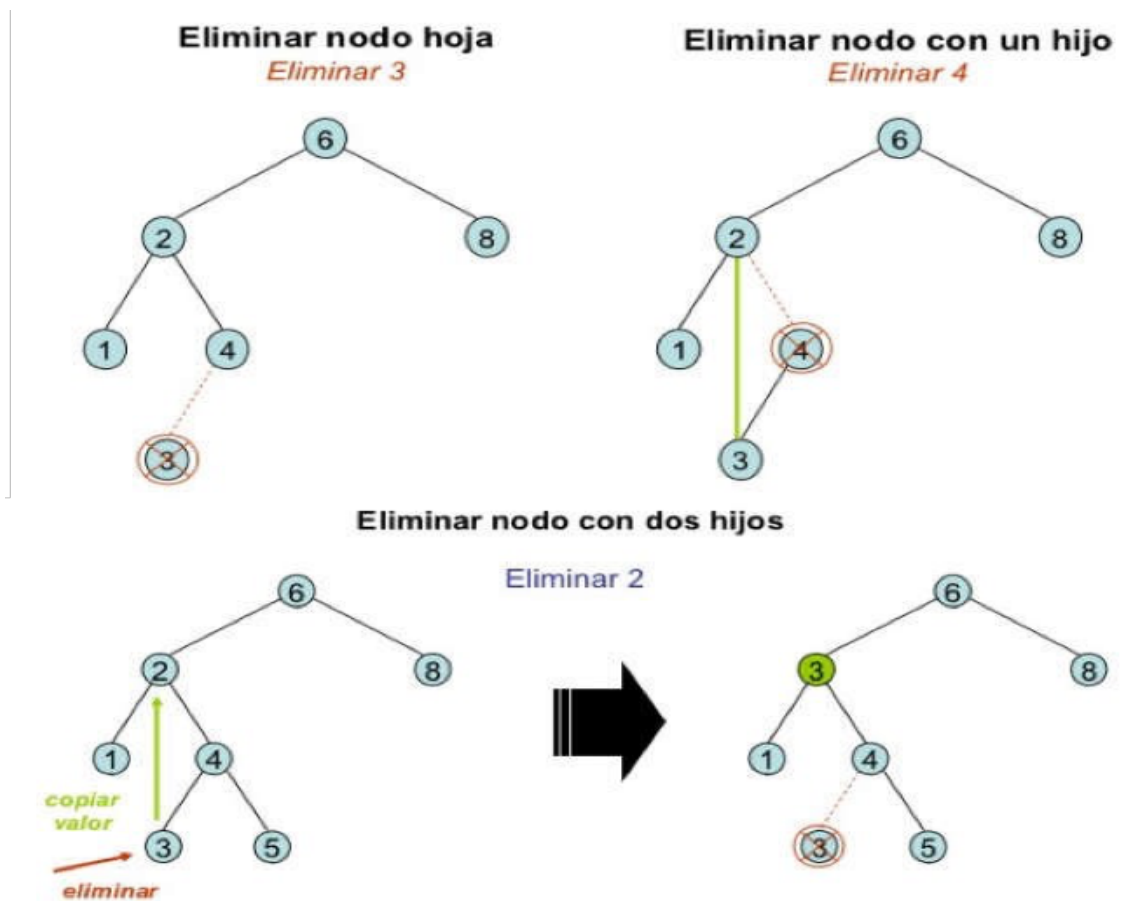


Figura 7: eliminar_nodo(int)

La última prueba correspondiente a la Figura 3, es la eliminación de la hoja 29 y se realiza otro recorrido por ancho. Después, se borra la hoja 53, por último, se elimina el nodo 12 con dos subárboles. El resultado obtenido es el que se muestra en la Figura 7.

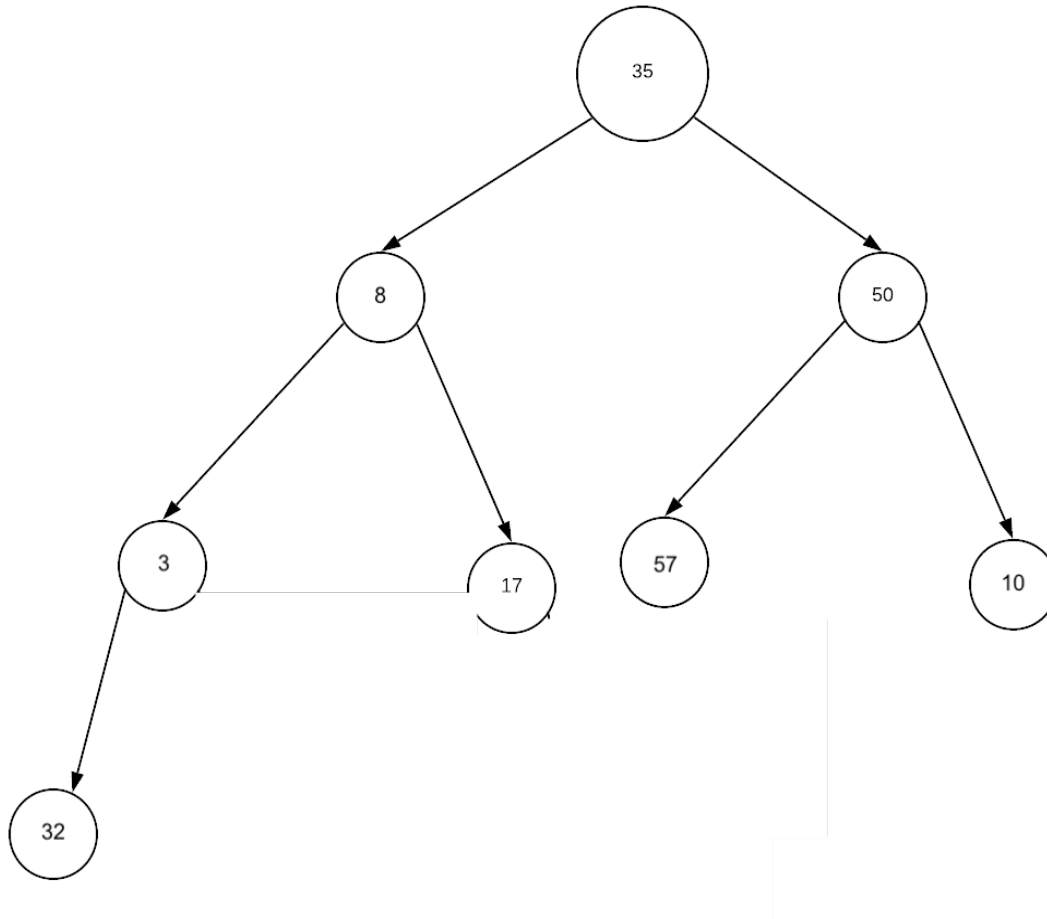


Figura 8: Árbol ABB tester 1

La Figura 8, representa los resultados respectivos al segundo tester. Se realizan funciones del método 2 y método 3. Aparte de estas funciones se obtiene resultados de las que se describen a continuación:

6.5. Método 5: `bool compara_cant_elem(ArbolABB*)`

Se define árboles binarios por sus siglas AB semejantes si tienen la misma caantidad de nodos y los valores de los nodos del primer árbol son iguales a los valores de los nodos del segundo nodo, No es relevante la relación de parentesco entre ellos, como se muestra en la Figura 9.

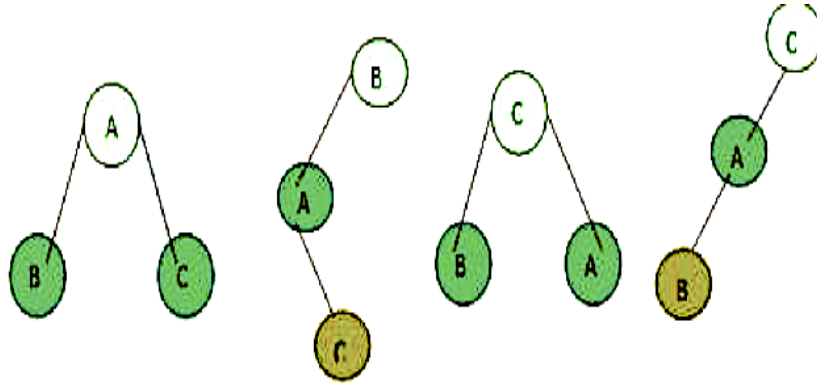


Figura 9: `compara_cant_elem(ArbolABB*)`

El método `compara_cant_elem(ArbolABB*)` recibe otro árbol como parámetro y revisa las estructuras de ambos para verificar en forma binaria tratando de encontrar un verdadero, si los árboles cumplen la condición de ser semejantes o no.

6.6. Método 6: `bool de_altura_son_iguales(ArbolABB*)`

Se define Árboles Binarios Ilos cuales de altura son idénticos si tienen la misma estructura aunque el contenido de cada uno de sus nodos sea diferente. Como se presenta en la Figura 10.

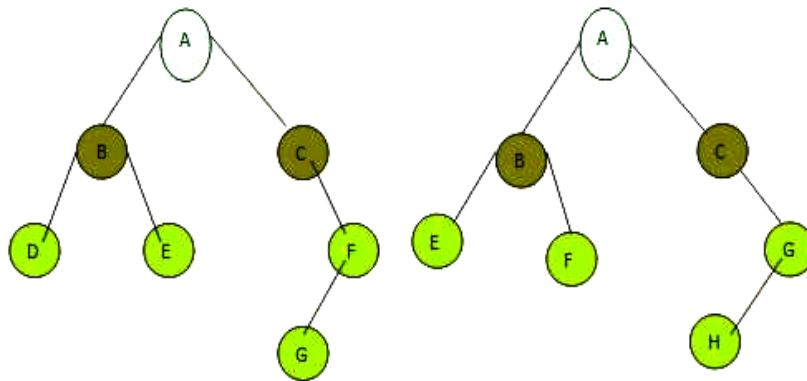


Figura 10: `de_altura_son_iguales(ArbolABB*)`

Por medio del método `de_altura_son_iguales(ArbolABB*)`, recibe otro árbol como parámetro y revisa las alturas de ambos para determinar en forma verdadera si los árboles cumplen la condición de ser iguales en altura o no.

Las operaciones realizadas en el tester 2 son similares a como se acomodaba la estructuras mencionadas anteriormente. Por lo que no se adjuntaran imágenes.

```
*****
Tester de arboles ABB #2
*****

El arbol (29,11,28,18,31):
No esta lleno

El arbol (29,11,28,18,31,8):
No esta lleno
El arbol (29,11,28,18,31,8)
No esta completo

El arbol (29,11,28,18,31,8,9,5,10,12)
Como resultado obtenemos que:
No esta completo

Los arboles (50,46,12,23,48,53,57,68,80) y (28,29,9,7,11,32,12,90,17)

Como resultado obtenemos que:
Comparando la altura no son iguales

Los arboles(50,46,12,23,48,53,57,68,80) y (53,80,23,57,46,12,50,68,48)

Como resultado obtenemos que:
Si son semejantes

////////////////////////////////////

Vamos a realizar las siguientes operaciones:

////////////////////////////////////

3 10 8 17 32 35 50 57
Borrando hoja 32
1
3 10 8 17 35 50 57
Borrando hoja 29
0
3 10 8 17 35 50 57
Borrando hoja 28
0
3 10 8 17 35 50 57
Borrando hoja 7
0
3 10 8 17 35 50 57
Borrando hoja 12
0
3 10 8 17 35 50 57
Borrando hoja 90
0
3 10 8 17 35 50 57
Borrando hoja 17
1
3 10 8 35 50 57
Borrando hoja 9
0
3 10 8 35 50 57
Borrando hoja 11
0
3 10 8 35 50 57
rm a.exe
```

Figura 11: Resultados ABB tester 2

Por otro lado, para realizar la comparación de los ABB vs los árboles negro y rojo (ANR), se realizó un algoritmo que funciona de la siguiente manera: Los datos son digitados por el usuario, uno por uno. Cada valor que se desea así como las operaciones a realizar. Debido a esto, los resultados adjuntos en la Figura 12, son del funcionamiento de la inserción de datos así como de la petición a repetir la opción digitada por el usuario si se ingresa más de un ítem por instrucción.

```
Belindas-MacBook-Air:Arboles_Rojo_y_Negro belindabrown$ make
g++ -g --std=c++11 -Wall *.cpp -o a.exe
./a.exe

*****
Árbol rojo y negro
*****

1. Agregar elemento al árbol
2. Buscar un elemento
3. PRE-ORDER
4. POST-ORDER
5. Eliminar un elemento del árbol
6. Salir
*****
Escoja una opción >>>> 1

Va a ser insertado ... 45
-----
Se agregó el valor
-----

>>>>Escoja una opción >>>> 1

Va a ser insertado ... 23
-----
Se agregó el valor
-----

>>>>Escoja una opción >>>> 1

Va a ser insertado ... 45 67
-----
Se agregó el valor
-----

>>>>Escoja una opción >>>> Favor digite una opción dentro del menú

>>>>Escoja una opción >>>> 1

Va a ser insertado ... 67
-----
Se agregó el valor
-----

>>>>Escoja una opción >>>> 1

Va a ser insertado ... 2
-----
Se agregó el valor
-----
```

Figura 12: ANR inserción, menú y petición

Con base a la Figura 13, se obtiene la forma del árbol ANR mediante la operación de post_Order.

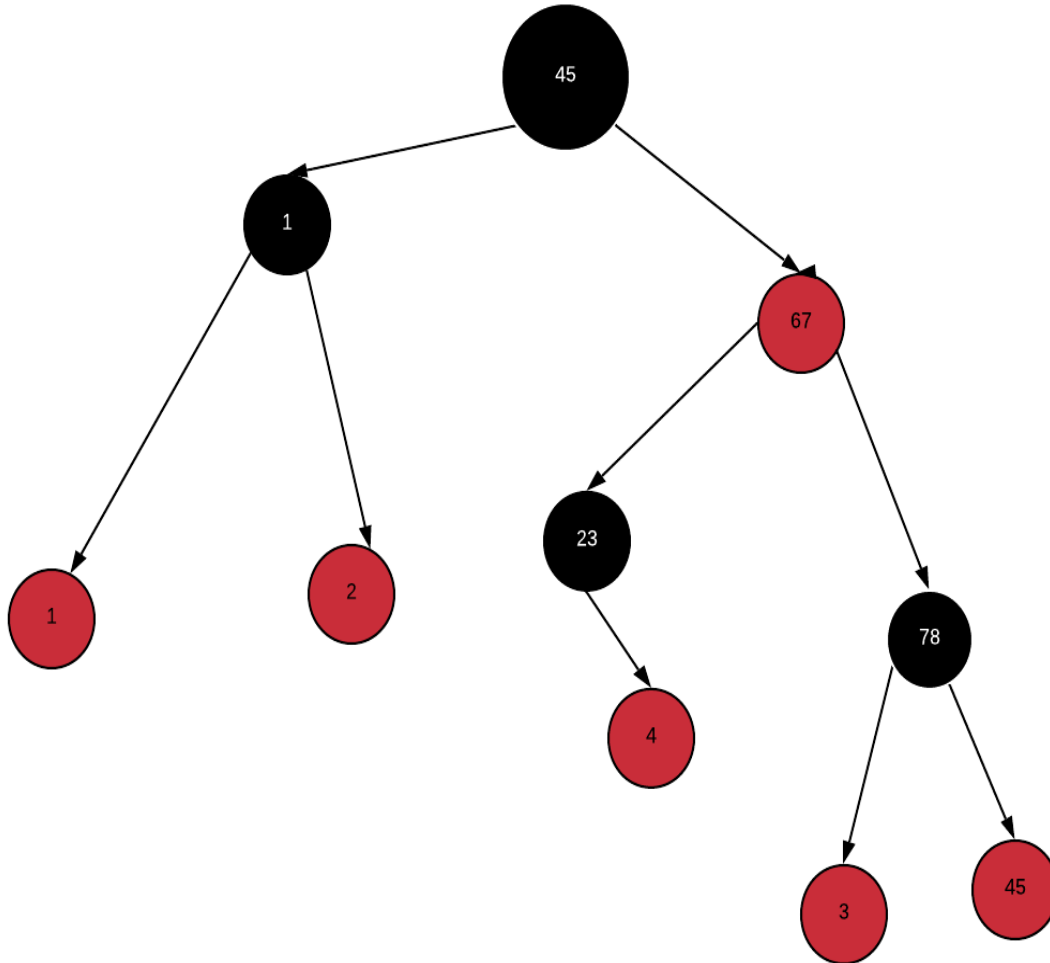


Figura 13: ANR

La diferencia entre el ABB y el ANR es notoria, no solo por su color sino en la forma de organizar los elementos y en las características que deben tener los hijos de nodos, nodo rojo o negro, raíz negra, hojas negras, hijos de todo nodo rojo son negros, cada camino tiene la misma cantidad de nodos negros y esto se denomina altura del árbol como se mencionan en el marco teórico.

```

Va a ser insertado ... 3
-----
Se agregó el valor
-----

****Escoja una opción **** 1

Va a ser insertado ... 1
-----
Se agregó el valor
-----

****Escoja una opción **** 1

Va a ser insertado ... 1
-----
Se agregó el valor
-----

****Escoja una opción **** 1

Va a ser insertado ... 4
-----
Se agregó el valor
-----

****Escoja una opción **** 1

Va a ser insertado ... 78
-----
Se agregó el valor
-----

****Escoja una opción **** 3
Pre-Order **** 45[Negro]**** 3[Rojos]**** 1[Negro]**** 1[Rojos]**** 2[Rojos]**** 23[Negro]**** 4[Rojos]**** 67[Rojos]**** 45[Rojos]**** 78[Rojos]

****Escoja una opción **** 4
Post-Order **** 1[Rojos]**** 2[Rojos]**** 1[Negro]**** 4[Rojos]**** 23[Negro]**** 3[Rojos]**** 45[Rojos]**** 78[Rojos]**** 67[Rojos]**** 45[Negro]

****Escoja una opción **** 2

Va a ser buscado **** 4
Se encontró el valor ****

```

Figura 14: ANR inserción, pre-order y post-order

Para finalizar con el ANR, se realiza la eliminación de un valor y después se recorre el árbol de forma pre-order y post-orden.

```

>>>>Escoja una opción >>>> 5

>>>>Eliminar el valor >>>>1

>>>>Escoja una opción >>>> 3
Pre-Order >>>> >>>> 45[Negro]>>>> 3[Roj]>>>> 2[Negro]>>>> 1[Roj]>>>> 23[Negro]>>>> 4[Roj]>>>> 67[Roj]>>>> 45[Roj]>>>> 78[Roj]

>>>>Escoja una opción >>>> 4
Post-Order >>>> >>>> 1[Roj]>>>> 2[Negro]>>>> 4[Roj]>>>> 23[Negro]>>>> 3[Roj]>>>> 45[Roj]>>>> 78[Roj]>>>> 67[Roj]>>>> 45[Negro]

>>>>Escoja una opción >>>> 6

Salir >>>>
rm a.exe

```

Figura 15: ANR eliminación, pre-order y post-order

El paso anterior, permite obtener el siguiente gráfico:

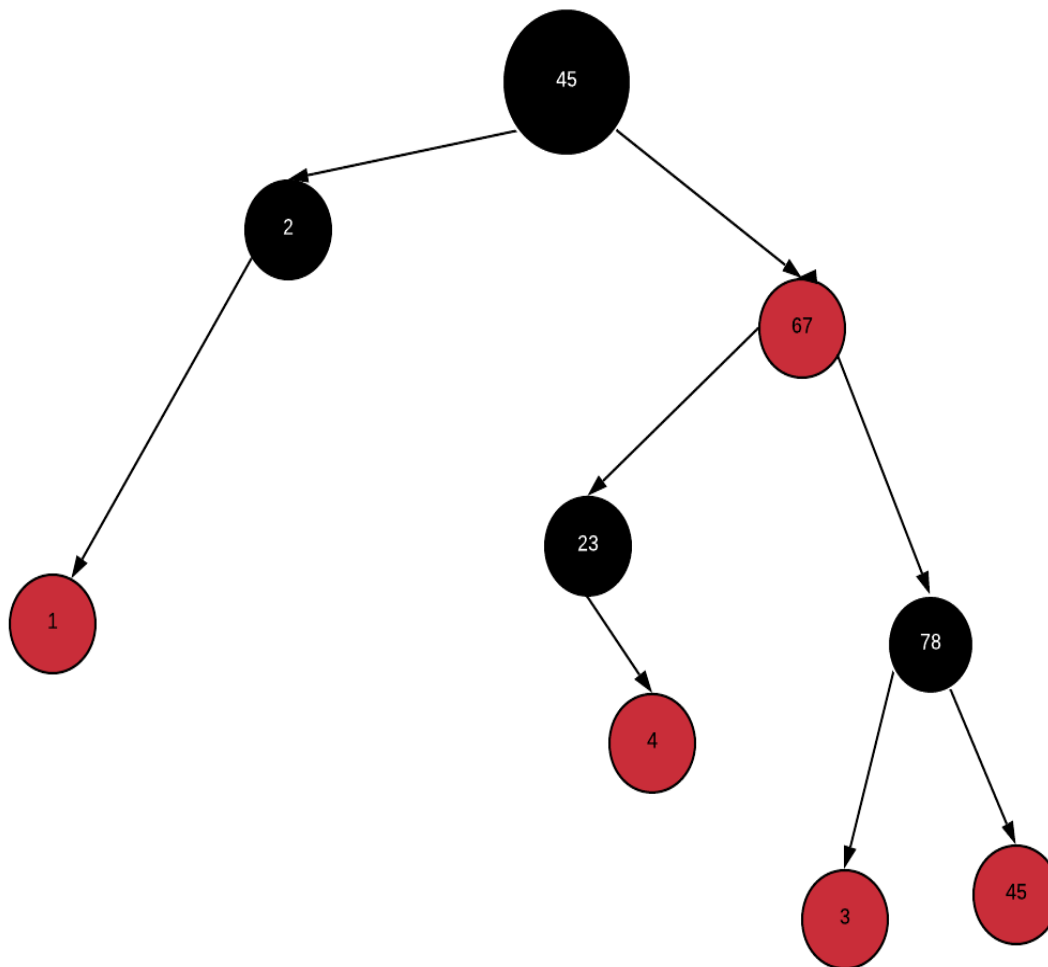


Figura 16: ANR eliminación

El cual nos permite ver como se organiza el ANR después de la eliminación de un valor. Se concluye finalmente que en funcionabilidad y tiempo, inserción del ABB y ANR es casi lo mismo. Por otro lado, eliminación y búsqueda son más eficientes los del ANR.

7. Conclusiones

7.1. Comparación de tiempo de método insertar

En esta sección se analizará el tiempo de ejecución de ambas funciones de insertar. Para el ABB se tiene 0.004s donde para el ANR es de 0.007s lo que indica que el tiempo de inserción es menor para el ABB pero se debe contemplar que para ANR se trabaja con un menú y no con un tester predeterminado.

Insertar

ABB

Parsed in 0.003 seconds at 38.20 KB/s

Statistics

Language used: cpp		
Statistic	Your source	Average for cpp
Characters in source:	110 (3307 in highlighted result)	2508.566
Total time highlighting:	0.004 seconds	0.109 seconds
Characters per second:	28720.956 chars per second	0.000/82090.532/1816989.643 chars per second

Black & Red

9.0

Parsed in 0.006 seconds at 263.65 KB/s

Statistics

Language used: cpp		
Statistic	Your source	Average for cpp
Characters in source:	1726 (12515 in highlighted result)	2508.533
Total time highlighting:	0.007 seconds	0.109 seconds
Characters per second:	238364.516 chars per second	0.000/82097.107/1816989.643 chars per second

Figura 17: Comparación de tiempo de método insertar

7.2. Comparación de tiempo de método eliminar

Por otro lado, se tiene el tiempo de ejecución de ambas funciones de eliminar. Para el ABB se tiene 0.012s donde para el ANR es de 0.018s lo que indica que el tiempo de inserción es menor para el ABB pero se debe contemplar que para ANR se trabaja con un menú y no con un tester predeterminado.

En la comparación de eliminación, se debe contemplar que el ANR lee casi el aproximadamente 32 mil caracteres más por segundo. Por lo que la comparación de es realmente mucho más eficiente para el ANR.

Eliminar		
ABB	Parsed in 0.011 seconds at 337.18 KB/s	
	Statistics	
	Language used: cpp	
	Statistic	Average for cpp
	Characters in source:	2508.592
Black & Red	Total time highlighting:	0.109 seconds
	Characters per second:	0.000/82106.972/1816989.643 chars per second
	Language used: cpp	
	Statistic	Average for cpp
	Characters in source:	2508.748
	Total time highlighting:	0.109 seconds
	Characters per second:	0.000/82118.127/1816989.643 chars per second

Figura 18: Comparación de tiempo de método eliminar

7.3. Compración tiempo de implementación de todas las funciones

Para finalizar se va a analizar la implementación de todas las funciones de cada árbol como conjunto para estimar el tiempo.

Implementación		
ABB	Parsed in 0.020 seconds at 342.60 KB/s	
	Statistics	
	Language used: cpp	
	Statistic	Average for cpp
	Characters in source:	2508.936
Black & Red	Total time highlighting:	0.109 seconds
	Characters per second:	0.000/82128.441/1816989.643 chars per second
	Parsed in 0.005 seconds at 232.39 KB/s	
	Statistics	
	Language used: cpp	
	Statistic	Average for cpp
	Characters in source:	2508.885
	Total time highlighting:	0.109 seconds
	Characters per second:	0.000/82133.694/1816989.643 chars per second

Figura 19: Implementación de todas las funciones

En la figura anterior se obtiene que el tiempo de las funciones del main toma para el ABB que es de tester 1 y 2 aproximadamente 0,021s en cambio realizando con menú como en el ANR tarda 0,006 más el tiempo que le toma al usuario. Por lo que se puede concluir, si es para hacer pruebas concretas se recomienda usar el método de pruebas que es el que se usa en el ABB, si se desea más versatilidad en las pruebas se debería usar el método de menú como se ejemplifica en el algoritmo ANR.

7.4. Gráfico función de tiempo

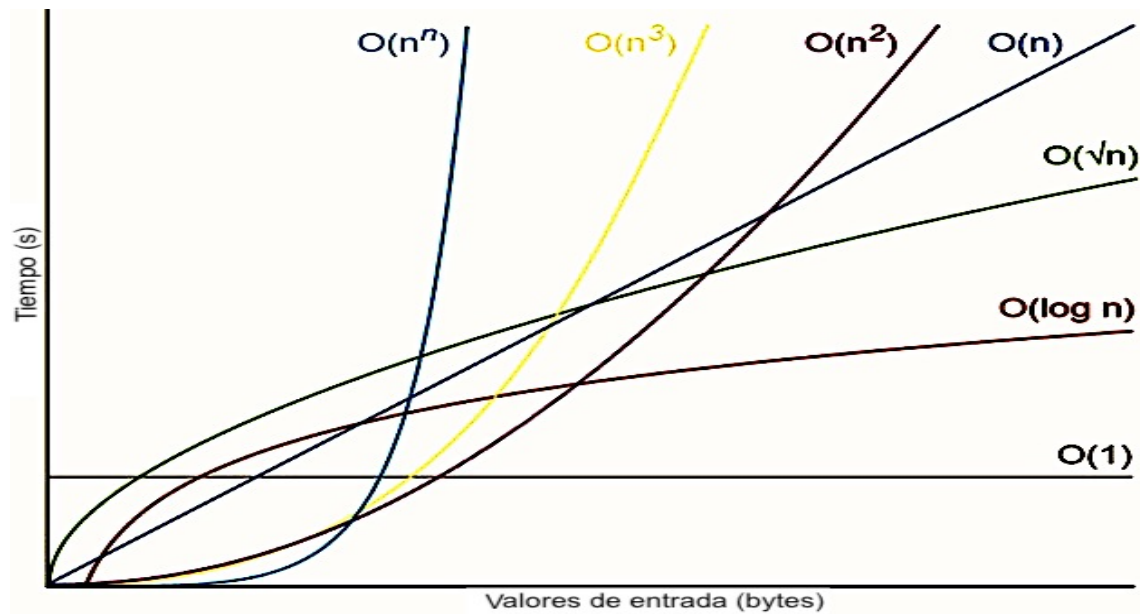


Figura 20: Tiempos y memoria

En general se puede concluir que los AB (árboles binarios) son de gran utilidad en la programación si se logra su correcta comprensión. Se debe comprender en su totalidad las funciones y operaciones ya que su implementación puede resultar compleja sin esto. Es esencial utilizar punteros. Y es importante recordar que el tiempo de operación de un ABB es $O(n)$ en cambio el de ANR $O(\log n)$. Como se mostró en la Figura 20.

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 C++: A Complete Introduction to the C++ 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.

8. Apéndice

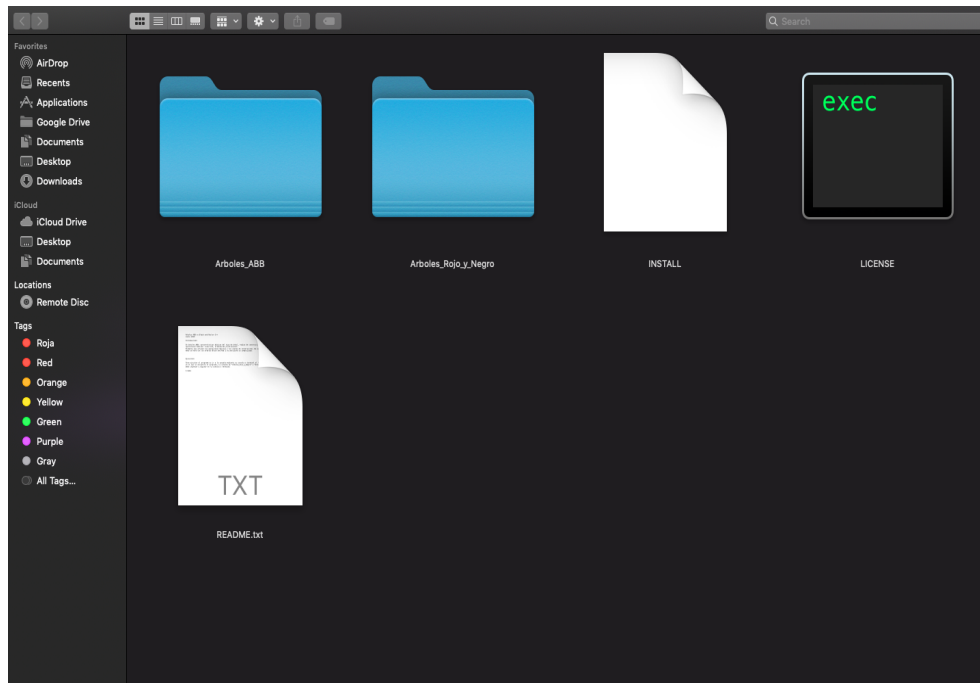


Figura 21: Dentro de Proyecto1_B61254

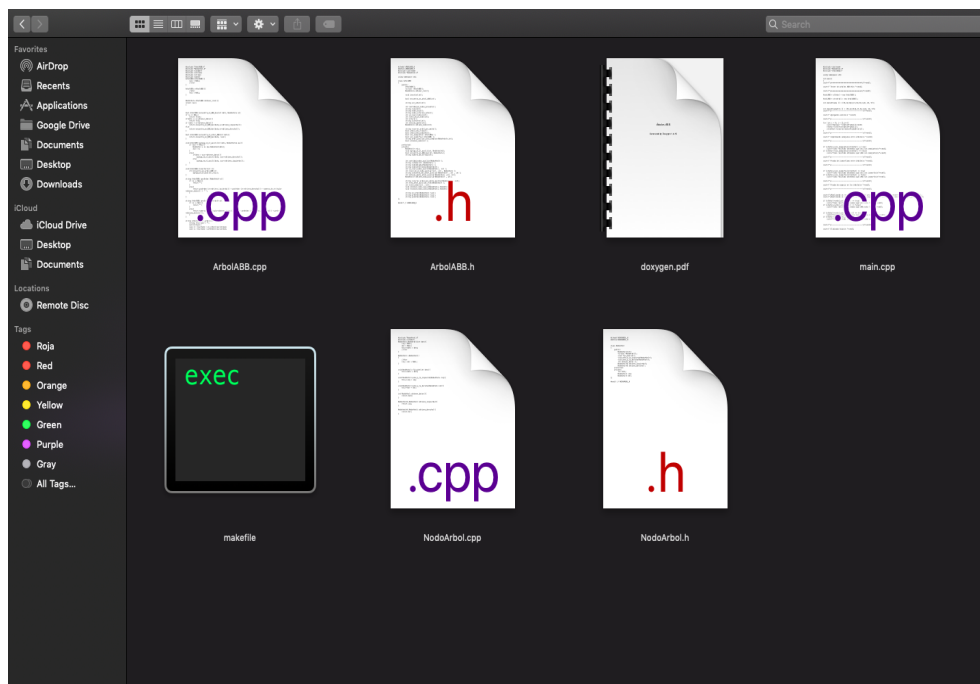


Figura 22: Dentro de Arboles_ABB

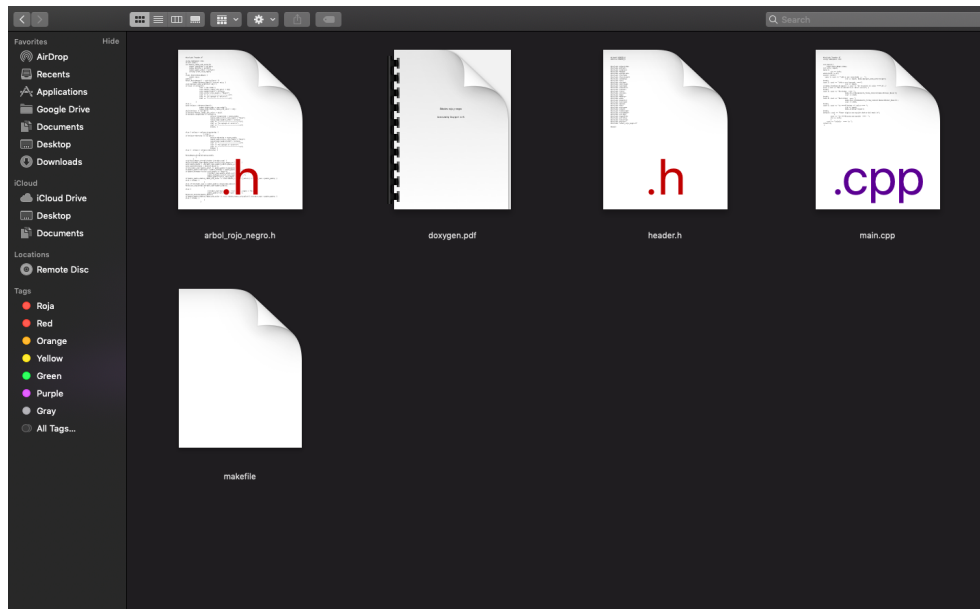


Figura 23: Dentro de Arboles_Rojo_y_Negro

8.1. Código fuente

Todos tienen en común lo siguiente:

```
Árboles ABB vs Black and Red en C++
Junio 2019

Introducción:

En árboles ABB: características básicas del tipo de árbol, reglas de construcción,
operaciones básicas: inserción, eliminación, localización.
Ejemplos que aclaran las operaciones básicas y las reglas de construcción. De este mismo
modo se hará con los árboles Black and Red y se analizará su complejidad.

Ejecución:

Para ejecutar el programa es ir a la carpeta mediante su consola o terminal al folder
en el que se encuentra el programa, la carpeta de "Arboles_Rojo_y_Negro" o "Arboles_ABB"
Debe ingresar y digitar en la consola o terminal:

$ make
```

Figura 24: Readme[2]

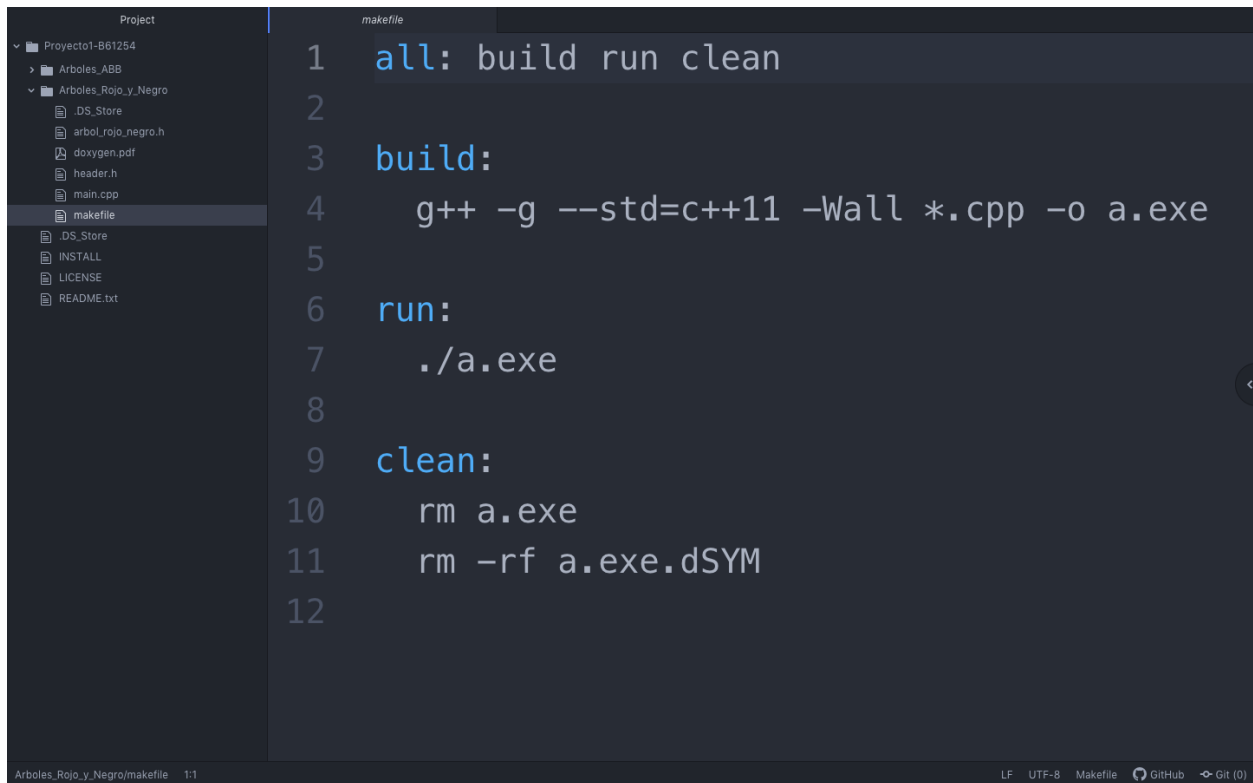
Proyecto comparación de árboles ABB vs black and red en C++

License Apache 2.0

Se distribuye un Makefile con 3 reglas:

- * build: compila los fuentes.
- * run: ejecuta un corrida de ejemplo.
- * clean: borra los binarios.

Figura 25: Install



```
1 all: build run clean
2
3 build:
4     g++ -g --std=c++11 -Wall *.cpp -o a.exe
5
6 run:
7     ./a.exe
8
9 clean:
10    rm a.exe
11    rm -rf a.exe.dSYM
12
```

Figura 26: Makefile[1]