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**

**TAREA:**
**PILA COLAS ARBOLES**

**ESTUDIANTES:**
**JESÚS ZÚÑIGA MÉNDEZ**

**PROFESOR:**
**RICARDO ROMÁN BRENES; M. SC.**

**II CICLO 2019**

# Índice



# Índice de figuras

# 1. Introducción.

A la hora de programar es necesario contar con estructuras de datos que sean eficaces para resolver ciertos problemas, en esta tarea se trabaja con pilas, colas y arbolas, las cuales son estructuras con diferentes ventajas útiles en distintos momentos a la hora de programar

# 2. Solución.

Para ejecutar el programa es necesario enviar mediante linea de comandos las instrucciones que se desean procesar, a continuación se describe el funcionamiento de cada uno de los códigos y su respectiva solución.

## 2.1. Balanceo de paréntesis.

Se uso una pila en la cual se van almacenando los paréntesis, si es un paréntesis de apertura se almacena y si es de salida se saca, si todos los paréntesis que se van a sacar coinciden con el paréntesis que están almacenados se dice que la hilera es valida si no la hilera es invalida

## 2.2. Colas de Prioridad.

Consiste en crear tantas colas como el usuario desee, una vez hecho esto se eliminan datos de estas según la prioridad escogida para hacer esto se uso un arreglo que contiene todas las colas, con un puntero a cada cola se trabaja con cada una de ellas, y se van sacando los números conforme diga la prioridad, una vez se hayan sacado los números deseados se pasa a la siguiente cola y así sucesivamente hasta que todas estén vaciás

## 2.3. Árbol de búsqueda binaria

### 2.3.1. Creación del árbol

Si se trabaja con POO, para crear un árbol se debe de crear un objeto de tipo árbol, este objeto estará construido por un contador de ítems que sirve para verificar si el árbol esta vació, y por un objeto de tipo nodo llamado raíz que apuntara al nodo inicial, este objeto a su vez se construye con un dato donde se almacena el valor y con dos objetos de tipo nodo, uno que apunta a la rama izquierda y otro a la rama derecha, así, para crear un árbol vació se instancia el objeto el cual contiene todas las funciones que permitan construir un árbol.

### 2.3.2. Inserción

Para insertar elementos primero se debe verificar si el árbol tiene cero ítems, si es así, se asigna a la raíz la posición de memoria de el nodo creado, así el valor de la raíz según el ejemplo seria 4 y sus hijos izquierda y derecha están vacíos, ver Figura 1.

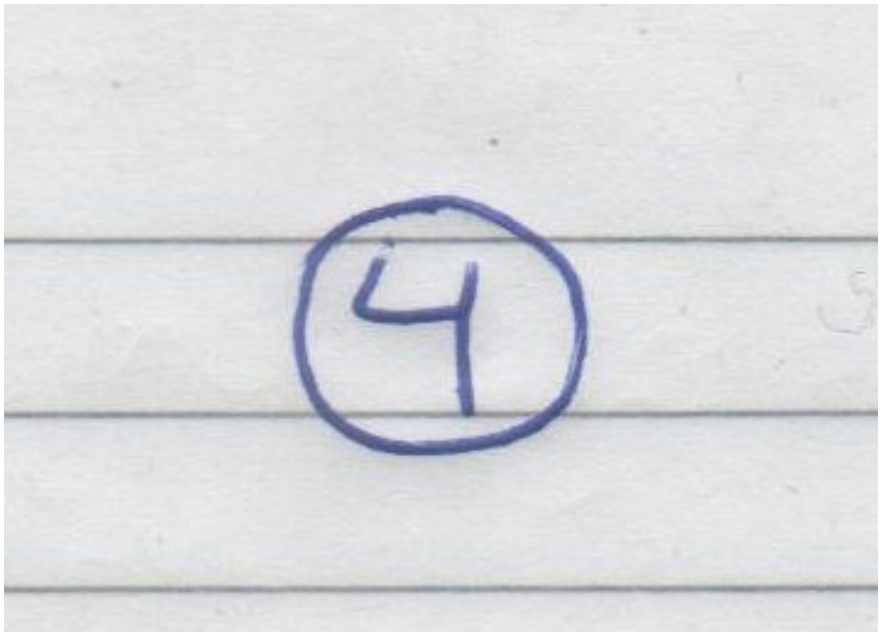


Figura 1: Nodo raíz del árbol

Continuando con la inserción de datos, en este caso el numero 7 se procede de la siguiente manera, primero se coloca un puntero sobre la raíz, si el valor de la raíz es menor que el valor a insertar se verifica que el nodo de la izquierda exista, si existe entonces se mueve el puntero a ese Nodo y se repite el ciclo, si no existe se crea el nodo y se asigna al puntero izquierdo de el ultimo nodo la dirección de memoria del nodo creado, ahora si el valor es mayor se realiza el mismo proceso pero esta vez en vez de moverse a la izquierda se mueve a la derecha. Ver Figura 2

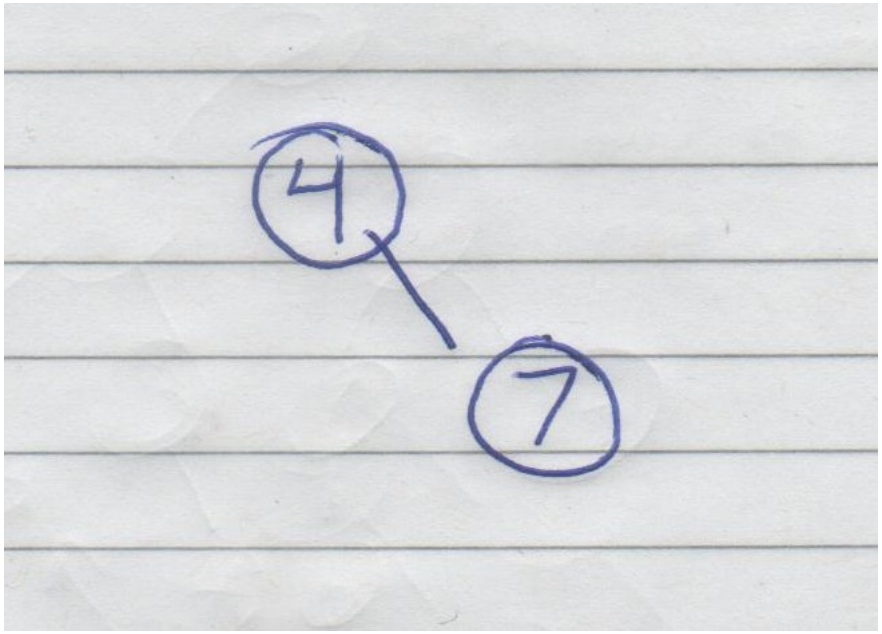


Figura 2: Nodo raíz con una hoja a la derecha

En las siguientes imágenes se puede ver el proceso de inserción, con una breve lista de los pasos seguidos

2 menor que 4 entonces izquierda, se inserta en nodo vació

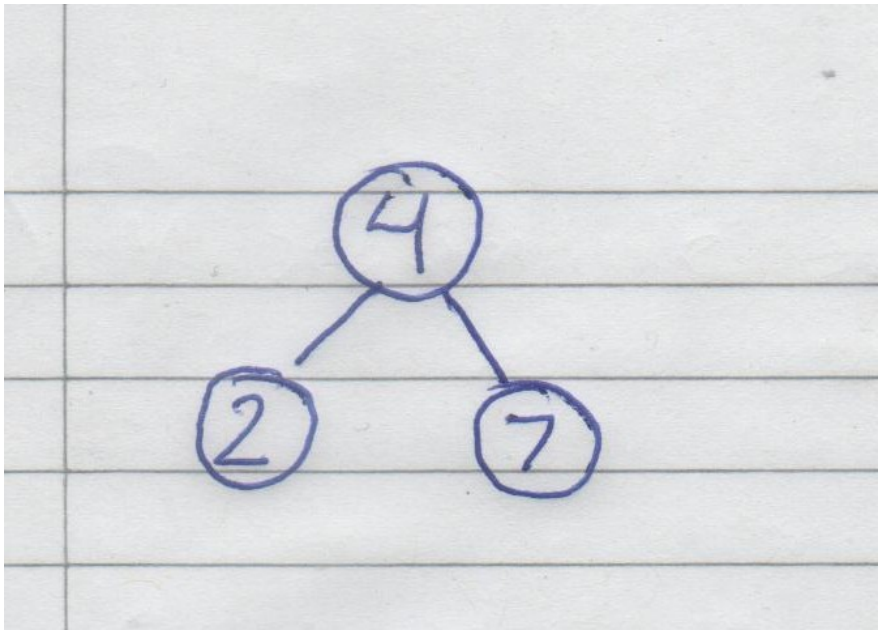


Figura 3: Inserción del 2

9 mayor que 4 entonces derecha, mayor que 7 entonces derecha, se inserta en nodo vacío

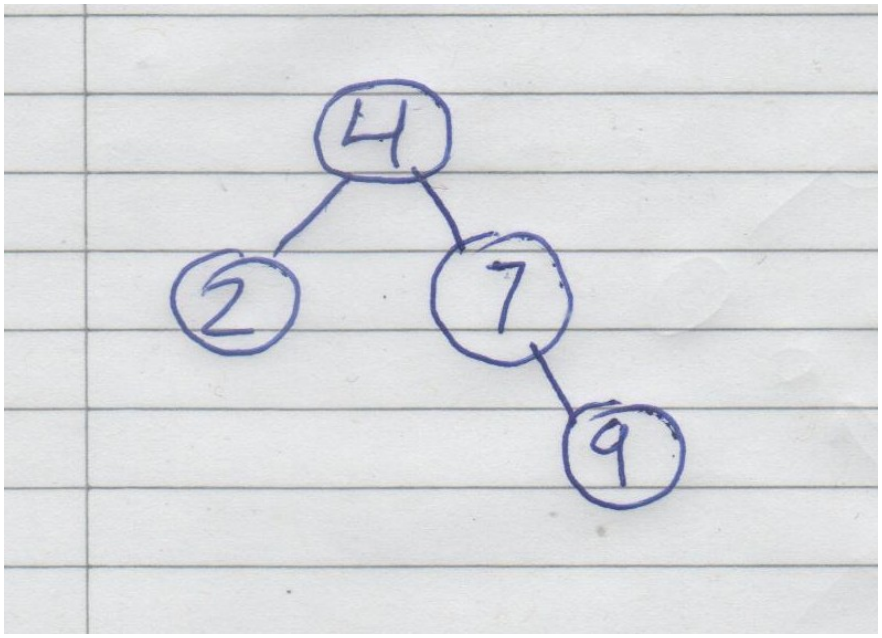


Figura 4: Inserción del 9

5 mayor que 4 entonces derecha, menor que 7 entonces izquierda, se inserta en nodo vacío

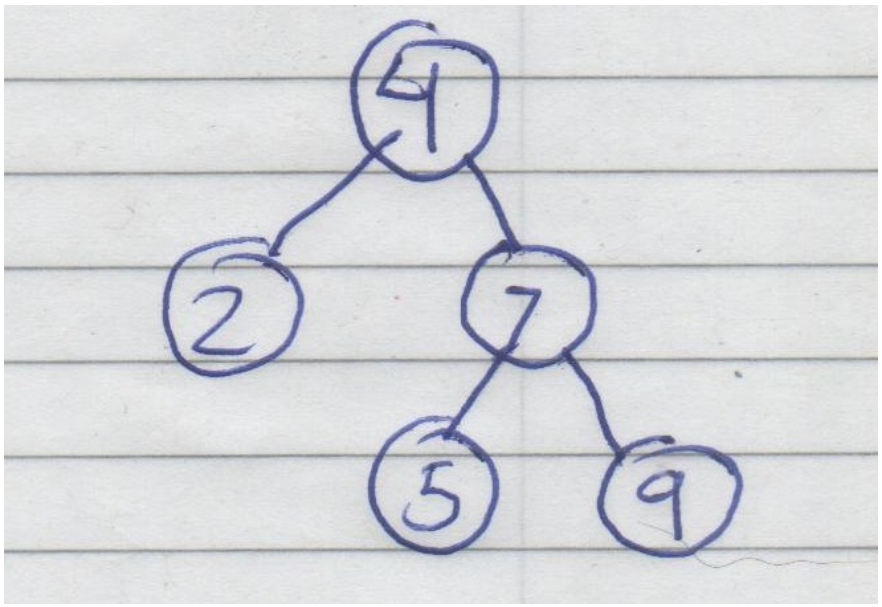


Figura 5: Inserción del 5

6 mayor que 4 entonces derecha, menor que 7 entonces izquierda,mayor que 5 entonces derecha, se inserta en nodo vacío

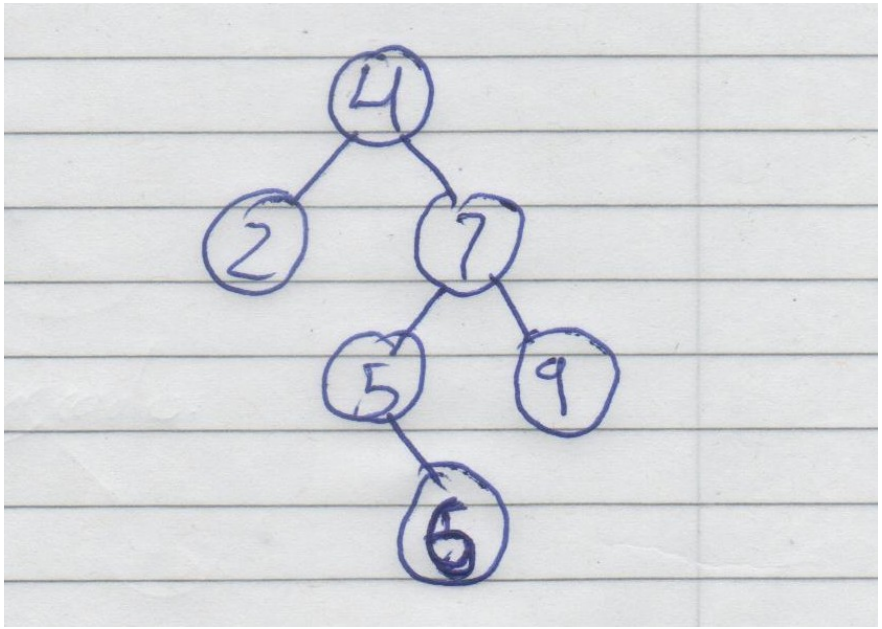


Figura 6: Inserción del 6

1 menor que 4 entonces izquierda, menor que 2 entonces izquierda, se inserta en nodo vacío

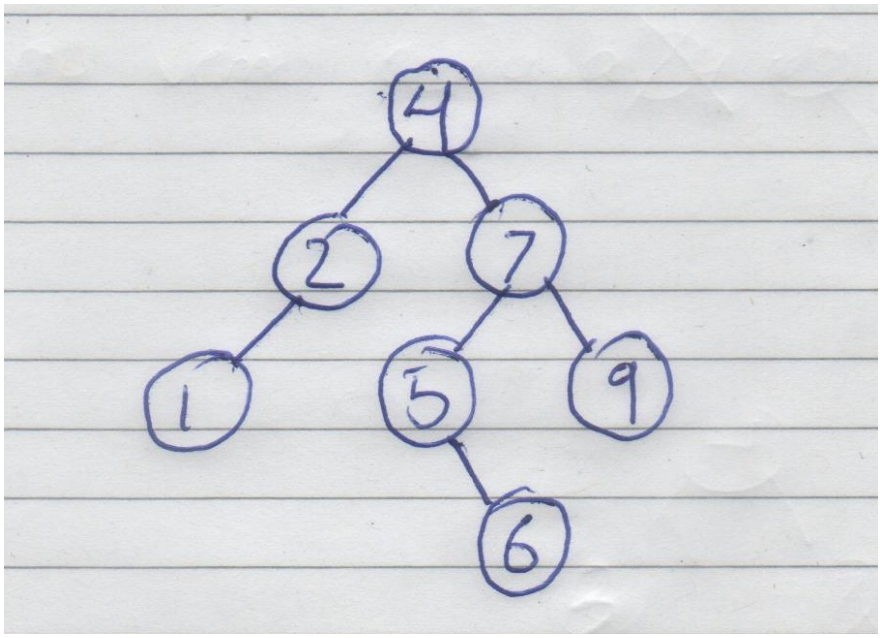


Figura 7: Inserción del 1

0 menor que 4 entonces izquierda, menor que 2 entonces izquierda,menor que 1 entonces izquierda, se inserta en nodo vacío

15 mayor que 4 entonces derecha, mayor que 7 entonces derecha,mayor que 9 entonces derecha, se inserta en nodo vacío

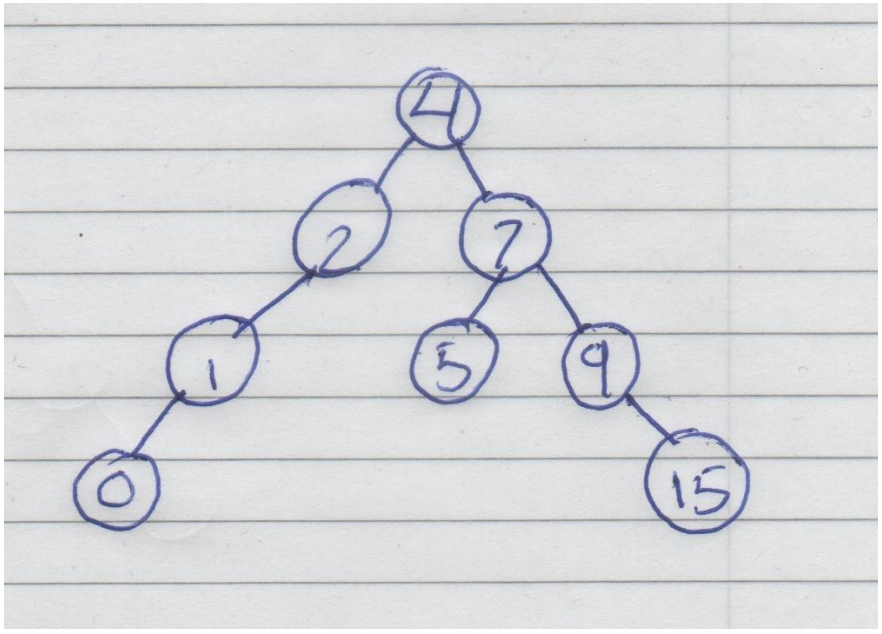


Figura 8: Inserción del 0 y el 15

3 menor que 4 entonces izquierda, mayor que 2 entonces derecha, se inserta en nodo vacío

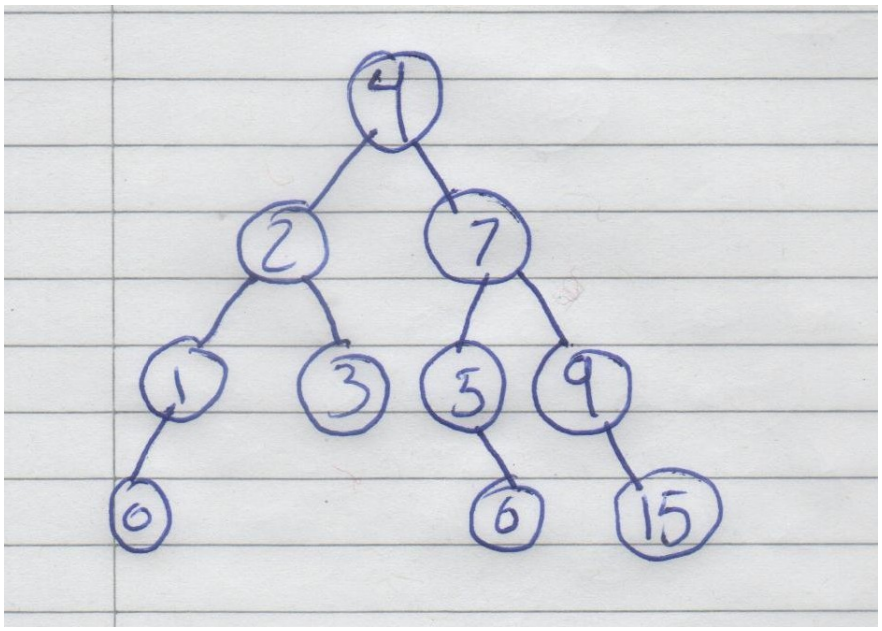


Figura 9: Inserción del 3

### 2.3.3. Borrado

Para borrar un nodo es necesario realizar dos búsquedas, la primera es buscar el nodo mayor a la izquierda del nodo a borrar, y la segunda es buscar el nodo menor a la derecha del nodo a borrar, si ambas existen entonces se puede escoger entre cualquiera, si no entonces se escoge la verdadera y se sustituye por el valor correspondiente, si el mayor a la izquierda o el menor a la derecha tienen hijos, es necesario realizar el proceso recursivamente, entonces en el ejemplo podemos realizar el borrado del 2, primero nos posicionamos sobre el numero correspondiente, realizamos ambas búsquedas, en este caso ambas existen, el mayor a la izquierda es el numero 1 y el menor a la derecha es el 3 como ambas existen entonces se puede usar cualquiera de las dos, si usamos el 3 por ejemplo simplemente es asignar el valor 3 al nodo donde esta el 2 y borrar el nodo 3, así logramos el resultado de la Figura 10.

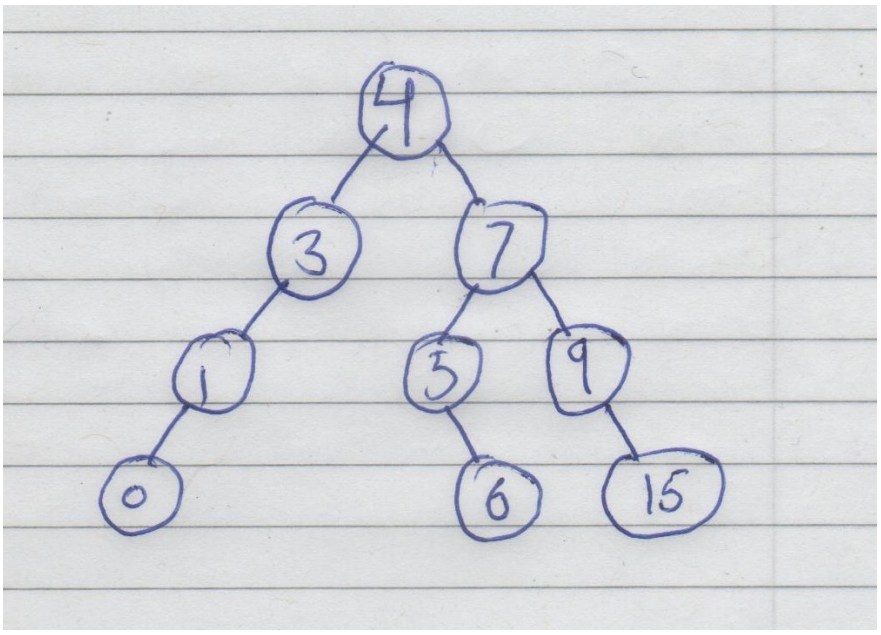


Figura 10: Borrado del 2

Para borrar el numero 6 realizamos el mismo procedimiento, buscamos el mayor a la izquierda y el menor a la derecha, como en este caso ninguno de los dos existe quiere decir que el 6 es una hoja, como no tiene hijos simplemente se borra, sucede lo mismo con el 0 y el 15. Así borrando los 3 números el resultado escoge el de la Figura 11

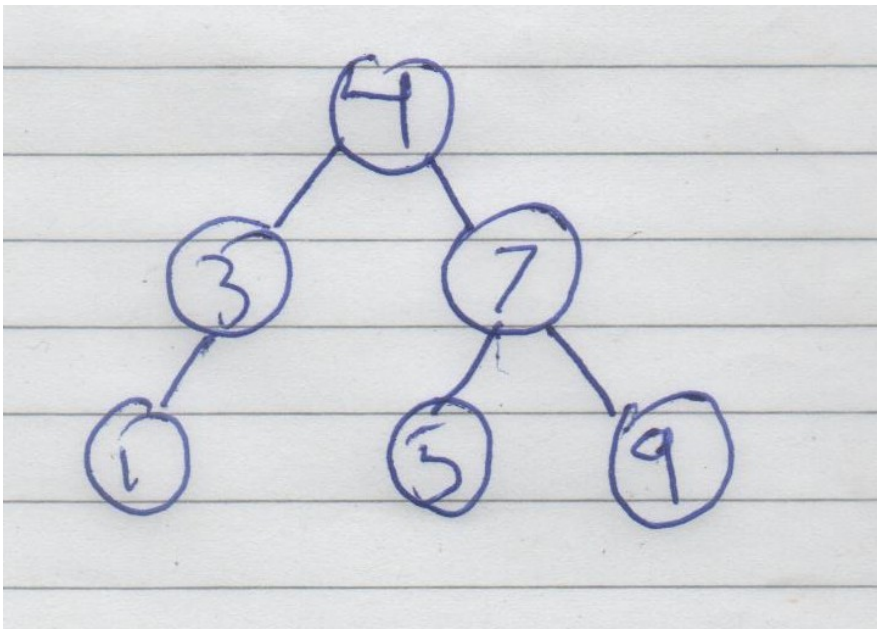


Figura 11: Borrado del 6, del 0, del 15

### 2.3.4. Balanceo

Para balancear lo mas fácil es crear un nuevo árbol a partir de el árbol desvalanceado, para esto primero se hace una lista con los números del árbol como en la Figura 12

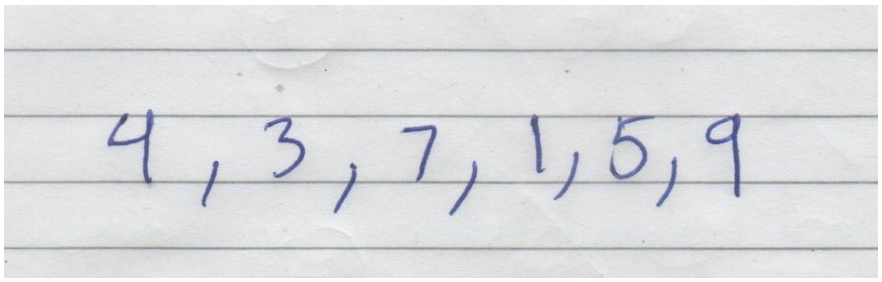


Figura 12: Lista de números

Después se acomodan en orden de menor a mayor. Ver Figura 13

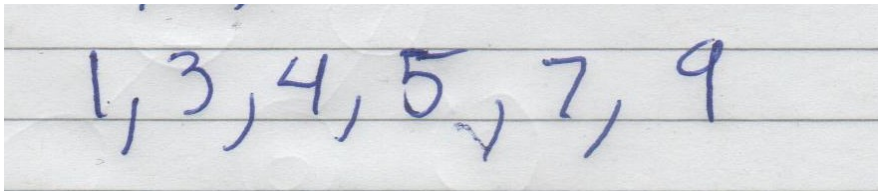


Figura 13: Lista de números ordenados

Con la lista acomodada basta realizar el siguiente proceso, se busca el valor del medio en este caso puede ser 4 o 5, tomamos el 5 y lo insertamos como en la Figura 14, después recursivamente seguimos con el mismo proceso buscamos la mitad de la mitad en este caso el 3 (Figura 15), después la mitad derecha que seria el 7 (Figura 16) y así sucesivamente hasta terminar con la lista

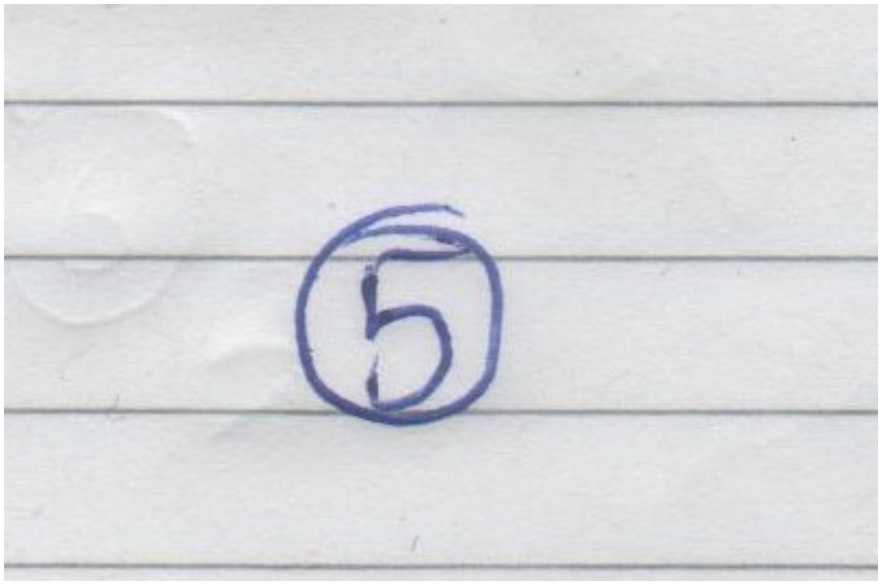


Figura 14: Inserción del 5

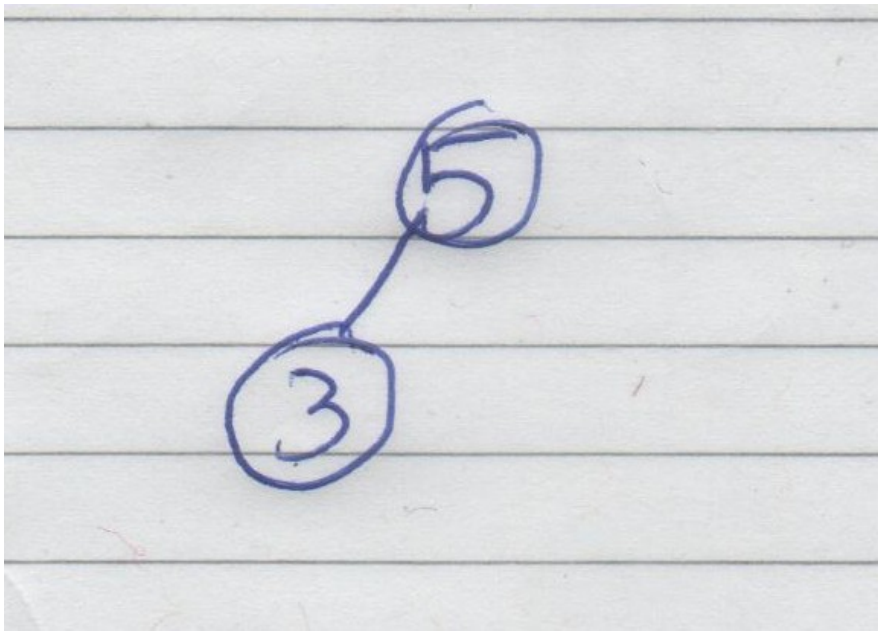


Figura 15: Inserción del 3

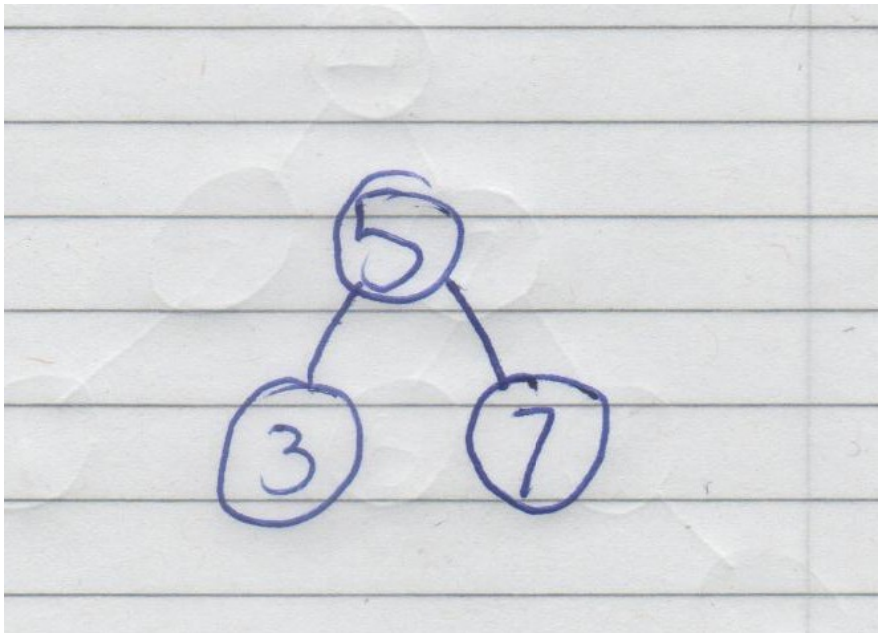


Figura 16: Inserción del 7

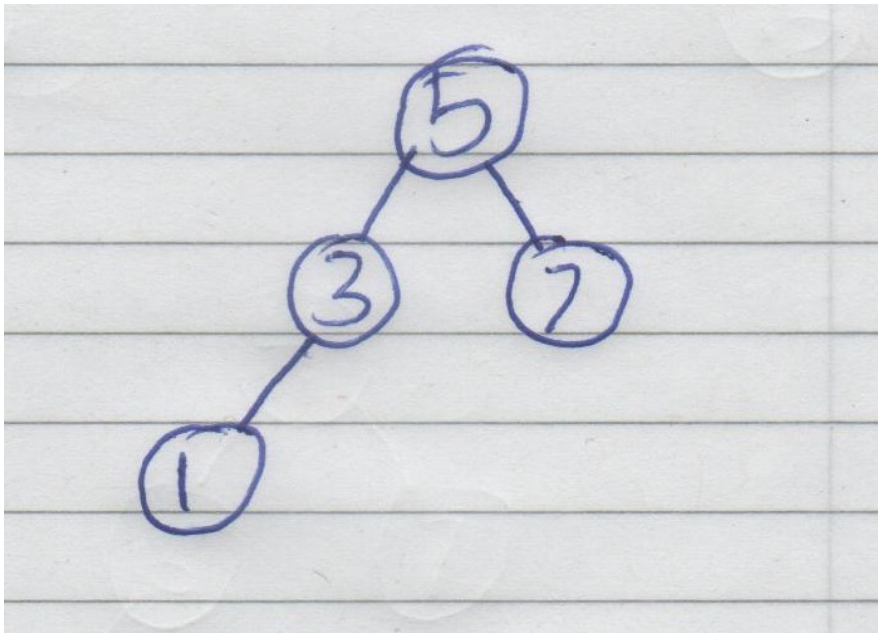


Figura 17: Inserción del 1

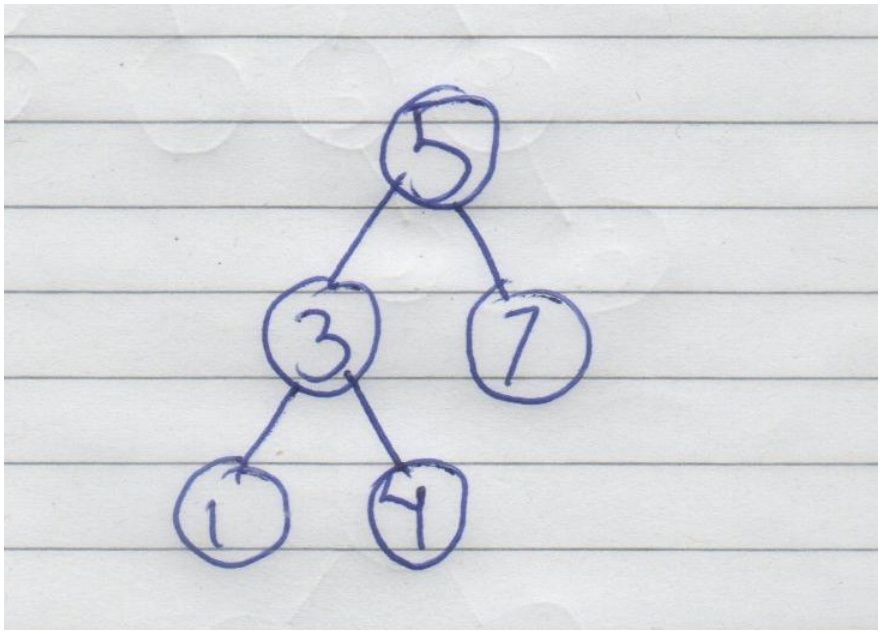


Figura 18: Inserción del 4

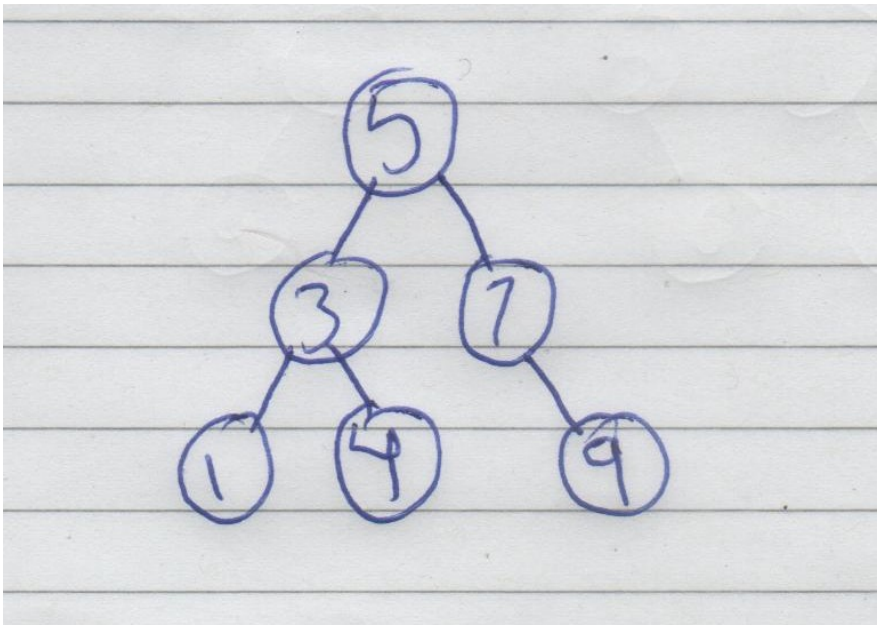


Figura 19: Inserción del 9

Al finalizar el proceso el árbol queda balanceado.

## 3. Conclusiones

Después de desarrollar esta tarea se logro concluir

- El manejo de pilas y colas es muy importante para manejar el orden de ejecución de procesos
- Son modelos relativamente sencillos y debido a su simpleza son muy cómodos de trabajar
- Un árbol binario puede realizar búsqueda de información en un tiempo muy reducido

## 4. Bibliografía

## Referencias

A continuacion se anexa el codigo del archivo main en el cual se realizaron las funciones correspondientes

# 5. Anexos

### 5.0.1. main.cpp

"main.cpp"

```cpp
#include "./Includes/Includes.h"


using namespace std;
#define Data Element<string> // vagancia
#define DataEntero Element<int> // vagancia



int numeroRandom(int minimo, int tope){
    //codigo tomado de https://es.stackoverflow.com/questions/148661/por-qu%C3%A9-el-n%C3%
    // Tenemos control sobre el algoritmo y distribución 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 número pseudo-aleatorio con el algoritmo
    mt19937 distribuido uniformemente entre 0 y 100 */
    int a = distribucion(generador);
    return a;
}


/**
 * @brief funcion que imprime un menu
 * @return devuelve la seleccion del usuario
 */
int Menu(){
    int respuesta=0;
    cout << "Digite la opcion que quiere probar" << endl;
    cout << "1: Parentesis" << endl;
    cout << "2: Prioridad" << endl;
    cout << "0: Salir" << endl;
    cin >> respuesta;
    return respuesta;
}

/**
 * @brief Funcion que revisa que los parentesis esten valanceados
 */
void Balanceo(string cadena){
    Stack<Data> laPila;
    string laCadena = cadena;
```

```
46      string elCaracter = "";
47      cout << "Revisando: " << laCadena << endl;
48      int tamanioArregloValidos = 3;
49      string arregloValidosApertura[tamanioArregloValidos] = {"{","[","("};
50      string arregloValidosCerradura[tamanioArregloValidos] = {"}","]",")"};
51      int estaBalanceado = 1;
52      //en este for se gusrda en una pila solo los parentesis de la cadena
53      for (uint i = 0; i < laCadena.size(); i++){
54          int esParentesisApertura=0;
55          int esParentesisCerradura=0;
56          int indexParentesis = 0;
57          elCaracter = laCadena[i];
58          for (int j = 0; j < tamanioArregloValidos; j++){
59              if (arregloValidosApertura[j] == elCaracter){
60                  esParentesisApertura = 1;
61              }
62              if (arregloValidosCerradura[j] == elCaracter){
63                  esParentesisCerradura = 1;
64                  indexParentesis = j;
65              }
66          }
67          if (esParentesisApertura == 1){
68              cout << "es apertura" << endl;
69              laPila.push(Data(elCaracter));
70          }
71          if (esParentesisCerradura == 1){
72              cout << "es cerradura" << endl;
73              Data e = laPila.pop();
74              if (e.isValid())
75              {
76                  cout << "comparo" << e.get() << " con " << arregloValidosApertura[indexPare
77                  if (e.get() != arregloValidosApertura[indexParentesis]){
78                      cout << "entre" << endl;
79                      estaBalanceado = 0;
80                  }
81              }
82          }
83      }
84
85      if (estaBalanceado == 0){
86          cout << "Hilera no valida" << endl;
87      }else{
88          cout << "Hilera valida" << endl;
89      }
90  }
91
92
93  /**
94   * @brief funcion que hace cola de prioridades
95   */
96  void Prioridad(string cadena){
97      string laCadena = cadena;
98      cout << "Voy a proccesar " << laCadena << endl;
99      string prioridad = "";
```

```
100     string espacio = " ";
101     string caracter = "";
102     string numeros = "1234567890";
103     string numero = "";
104     string cantidadColasCadena = "";
105     int cantidadColas = 0;
106     int yaEmpezeCadenaColas = 0;
107 
108     //sacamos el numero de colas
109     for (uint i = 0; i < laCadena.size(); i++){
110         caracter = laCadena[i];
111         if (caracter != " "){
112             for (uint j = 0; j < numeros.size(); j++){
113                 numero = numeros[j];
114                 if (caracter == numero){
115                     cantidadColasCadena = cantidadColasCadena + numero;
116                     yaEmpezeCadenaColas = 1;
117                 }
118             }
119         }else if (yaEmpezeCadenaColas == 1){
120             i = laCadena.size();
121             cantidadColas = stoi(cantidadColasCadena);
122         }
123     }
124 
125 
126     //llenamos el arreglo con la cantidad de colas
127     Queue<DataEntero> *ArregloColas[cantidadColas];
128     Queue<DataEntero> *ArregloColasRespaldo[cantidadColas];
129     int generadoAlestorio = 0;
130     for (int i = 0; i < cantidadColas; i++){
131         int cantidadNumerosEnLaCola = numeroRandom(1,10);
132         Queue<DataEntero> *cola = new (Queue<DataEntero>);
133         Queue<DataEntero> *colaRespaldo = new (Queue<DataEntero>);
134         for (int j = 0; j < cantidadNumerosEnLaCola; j++){
135             generadoAlestorio = numeroRandom(1,100);
136             cola->enqueue(DataEntero(generadoAlestorio));
137             colaRespaldo->enqueue(DataEntero(generadoAlestorio));
138         }
139         ArregloColas[i] = cola;
140         ArregloColasRespaldo[i] = colaRespaldo;
141     }
142 
143     //se imprimen las colas hechas
144     for (int i = 0; i < cantidadColas; i++)
145     {
146         Queue<DataEntero> *cola = ArregloColas[i];
147         string impresion = "Cola " + to_string(i+1) + " : ";
148         int seguir = 0;
149         while (seguir == 0){
150             DataEntero e = cola->dequeue();
151             if (e.isValid())
152             {
153                 impresion = impresion +  to_string(e.get()) + ",";
```

```
154             }else{
155                 seguir = 1;
156             }
157         }
158 
159         cout << impresion << endl;
160     }
161 
162     //hacemos un arreglo con las prioridades
163     int arregloPrioridades[cantidadColas];
164     int contadorEspacios = 0;
165     string cantidadPrioridad = "";
166     int contador = 0;
167     for (uint i =0; i < laCadena.size(); i++){
168         caracter = laCadena[i];
169         if ((caracter != " ") && (contadorEspacios == 0)){
170             contadorEspacios = 1;
171         }else if ((caracter != " ") && (contadorEspacios == 1)){
172             if ((caracter == ":") || (i == (laCadena.size() - 1))){
173                 if (i == (laCadena.size() - 1)){
174                     cantidadPrioridad = cantidadPrioridad + caracter;
175                 }
176                 if (contador < cantidadColas){
177                     arregloPrioridades[contador] = stoi(cantidadPrioridad);
178                     contador++;
179                 }else{
180                     i = laCadena.size();
181                 }
182                 cantidadPrioridad = "";
183             }else{
184                 cantidadPrioridad = cantidadPrioridad + caracter;
185             }
186         }
187     }
188 
189 
190     //calculamos el resultado
191     int seguir = 0;
192     string resultado = "Salidas: ";
193     while (seguir == 0){
194         for (int i = 0; i < cantidadColas; i++){
195             int cantidad = arregloPrioridades[i];
196             Queue<DataEntero> *cola = ArregloColasRespaldo[i];
197             for (int j = 0; j < cantidad; j++){
198                 if (ArregloColasRespaldo[i] == 0x0){
199                     int contador = 0;
200                     for (int k = 0; k < cantidadColas; k++){
201                         if (ArregloColasRespaldo[k] == 0x0){
202                             contador ++;
203                         }
204                     }
205                     if (contador == cantidadColas){
206                         seguir = 1;
207                     }
```

```
208                 }else{
209                     DataEntero e = cola->dequeue();
210                     if (e.isValid())
211                     {
212                         resultado = resultado +  to_string(e.get()) + ",";
213                     }else{
214                         ArregloColasRespaldo[i] = 0x0;
215                     }
216                 }
217             }
218 
219         }
220 
221     }
222 
223     cout << resultado << endl;
224 
225 //Queue<Data> laCola;
226 
227 //laCola.enqueue(Data(elCaracter));
228 
229 //Data e = laCola.dequeue();
230         /*for (uint i = 0; i < laCadena.size(); i++)
231     {
232         Data e = laCola.dequeue();
233         if (e.isValid())
234         {
235             cout << e.get() << endl;
236         }
237     }*/
238 
239     //Stack<Data> laPila;
240     //laPila.push(elCaracter);
241     // Data e = laPila.peek();
242     // if (e.isValid())
243     //     cout << "peeking: " << e.get() << endl;
244 
245     // for (uint i = 0; i < laCadena.size(); i++)
246     // {
247     //     Data e = laPila.pop();
248     //     if (e.isValid())
249     //     {
250     //         cout << e.get() << endl;
251     //     }
252     // }
253 
254 }
255 
256 
257 int main(int argc, char** argv)
258 {
259     int seleccion = 0;
260     string laCadena = argv[1];
261     do{
```

```
262         seleccion = Menu();
263         if (seleccion == 1){
264             cout << "Balanceo" << endl;
265             Balanceo(laCadena);
266         }
267 
268         if (seleccion == 2){
269             cout << "Prioridad" << endl;
270             Prioridad(laCadena);
271         }
272     }while (seleccion != 0);
273     return 0;
274 }
```