

## Laboratorio Nro. 4 Tablas de Hash y Árboles

**Alejandra Palacio Jaramillo**  
Universidad Eafit  
Medellín, Colombia  
apalacioj@eafit.edu.co

**Valentina Moreno Ramírez**  
Universidad Eafit  
Medellín, Colombia  
vmorenor@eafit.edu.co

### 3) Simulacro de preguntas de sustentación de Proyectos

#### 3.1 Estructura de datos implementada: árbol n-ario.

Para solucionar este problema, se diseñó un algoritmo que simula un Octree ya que, como su nombre lo indica, los 8 subnodos que tiene representan cuadrantes de las dimensiones en el espacio. Para representar este árbol, se utilizaron como estructuras base un vector dinámico y listas enlazadas ya que, en el caso de los vectores dinámicos, la complejidad para acceder a un elemento es de  $O(1)$  y, en el caso de las listas enlazadas, la complejidad para insertar. Cada casilla de este vector dinámico representa un sector o cuadrante del espacio y dentro de cada una de ellas se tiene una lista enlazada que contiene las abejas que se encuentran en ese sector. Ahora, si la diagonal en cada sector es mayor que 100, se crea otro octree dentro de este nodo y se repite el proceso recursivamente. Este procedimiento se realiza con el fin de obtener como nodos terminales sectores donde la diagonal sea menor o igual a 100 para así saber cuales son los conjuntos de abejas que colisionan.

La complejidad del algoritmo es  $O(n \log n)$ , donde  $n$  es el número de abejas, ya que la complejidad de crear un octree es de  $O(n)$  y, como se crea recursivamente pero cada vez con menos datos porque estos se dividen en 8 nodos, entonces la complejidad final sería  $O(n \log n)$ .

**3.3** Un árbol que se muestra con una organización pre-orden, es aquel en donde se identifica primero el nodo de la raíz, para después ir indicando los valores de todo el subárbol izquierdo y cuando culmine este proceso se irá por los valores del subárbol derecho. En nuestro código, implementamos las clases, Binary Tree Node y Binary Tree para crear nuestro árbol con todos los métodos que una estructura de datos debería de tener. Para llevar a cabo los métodos recursivos, en el caso del pre-orden, se recibirá primero el valor inicial de la raíz del árbol, se imprimirá el valor que reciba el método para después realizar el llamado recursivo que recorrerá todos los elementos de sub-árbol izquierdo, y después se hará el llamado

## ESTRUCTURA DE DATOS 1

### Código ST0245

nuevamente, pero en este caso recorriendo los elementos del sub-árbol derecho. Cuando se realiza el método de pos-order cambia lo que cambiará será la ubicación de la función de imprimir, la cual irá al final del método en vez del inicio. Ya así tendremos nuestra manera de calcular los árboles de manera post-ordenada y pre-ordenada fácilmente.

**3.5** La complejidad de todos los métodos es de  $O(n)$  ya que, para imprimir en preorden y postorden se debe pasar por todos los nodos del árbol. La complejidad de cualquiera de estos dos algoritmos, en términos de  $T(n)$  es  $2T(n/2)+c$ , lo cual nos lleva a una complejidad lineal. El método `buildingTree` tiene una complejidad de  $O(n \log n)$  puesto que, mediante un ciclo `for`, se recorre el arreglo dado en preorden para luego insertar cada elemento en su posición en el árbol. Finalmente, la complejidad final del algoritmo que soluciona el problema planteado es  $O(n) + O(n \log n)$ , lo cual se puede simplificar a  $O(n \log n)$ .

**3.6** La variable  $n$  será el número de nodos que contiene el árbol binario dado.

#### 4) Simulacro de Parcial

4.1

4.1.1 b

4.1.2 d

4.2 c

4.3

a. false

b. `return suma == a.data`

c. `sumaElCamino(a.der, suma - a.data)`

d. `|| sumaElCamino(a.izq, suma - a.data)`

4.4

4.4.1 c

4.4.2 a

4.4.3 d

4.4.4 a

4.5

4.5.1 `tolInsert == p.data`

4.5.2 `tolInsert > p.data`

4.6

4.6.1 d

4.6.2 `return 0`

4.6.3 `raiz.hijos.size() == 0`

4.7

4.7.1 a

**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas

Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627





















Tel: (+57) (4) 261 95 00 Ext. 9473

## ESTRUCTURA DE DATOS 1

### Código ST0245

- 4.7.2 b
- 4.8 b
- 4.9 a
- 4.10 Por definir...
- 4.11
  - 4.11.1 b
  - 4.11.2 a
  - 4.11.3 b
- 4.12
  - 4.12.1 i)
  - 4.12.2 a
  - 4.12.3 a
- 4.13
  - 4.13.1 suma[e.id]
  - 4.13.2 a

### 5) Trabajo en equipo

Se mejora código de detección de colisiones  apalacioj committed 1 hour ago		Verified		b49f63	<>
Commits on Oct 26, 2020					
Add files via upload  vmorenor16 committed 9 hours ago		Verified		1f12548	<>
Update Point2.java  vmorenor16 committed 9 hours ago		Verified		14f9cc2	<>
Update Octree.java  apalacioj committed yesterday		Verified		d8af01e	<>
Update Reader.java  apalacioj committed yesterday		Verified		52bb40c	<>
Punto1.1  apalacioj committed yesterday		Verified		7257edb	<>
informe  apalacioj committed yesterday		Verified		5fe8e14	<>
Commits on Oct 25, 2020					
Solución del punto 2.1  vmorenor16 committed yesterday		Verified		e752d42	<>
Solución alternativa del problema  vmorenor16 committed yesterday		Verified		6d5c280	<>
Clase abeja  apalacioj committed yesterday		Verified		53aafa6	<>

**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
 Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
 Tel: (+57) (4) 261 95 00 Ext. 9473



## ESTRUCTURA DE DATOS 1

### Código ST0245

Archivo Inicio Insertar Diseño Disposición Referencias Correspondencia Revisar Vista Ayuda

Portapapeles Fuente Párrafo Estilos Edición

calcular los árboles de manera **post-ordenada** y **pre-ordenada** fácilmente.

3.5 La complejidad de todos los métodos es de  $O(n)$  ya que, para imprimir en **preorden** y **postorden** se debe pasar por todos los nodos del árbol. La complejidad de cualquiera de estos dos algoritmos, en términos de  $T(n)$  es  $2T(n/2)+c$ , lo cual nos lleva a una complejidad lineal. El método **buildingTree** tiene una complejidad de  $O(n \log n)$  puesto que, mediante un ciclo **for**, se recorre el arreglo dado en **preorden** para luego insertar cada elemento en su posición en el árbol. Finalmente, la complejidad final del algoritmo que soluciona el problema planteado es  $O(n) + O(n \log n)$ , lo cual se puede simplificar a  $O(n \log n)$ .

3.6 La variable  $n$  será el número de nodos que contiene el árbol binario dado.

**4) Simulacro de Parcial**

4.1

4.1.1 b

4.1.2 d

4.2 c

4.3

a. false

b. return suma == a.data

c. sumaElCamino(a.der, suma - a.data)

d. | sumaElCamino(a.izq, suma - a.data)

4.4

4.4.1 c

4.4.2 a

4.4.3 d

4.4.4 a

4.5

4.5.1 toInsert == p.data

4.5.2 toInsert > p.data

4.6

Historial de versiones

Modificado por:	Fecha
Modificado por: Alejandra Palacio Jaramillo y Valentina Moreno Ramirez	4:09 p. m.
Modificado por: Alejandra Palacio Jaramillo y Valentina Moreno Ramirez	11:58 p. m.
Modificado por: Alejandra Palacio Jaramillo	11:43 p. m.
Modificado por: Alejandra Palacio Jaramillo	11:26 p. m.
Modificado por: Alejandra Palacio Jaramillo	11:15 p. m.
Modificado por: Alejandra Palacio Jaramillo	11:13 p. m.

Archivo Inicio Insertar Diseño Disposición Referencias Correspondencia Revisar Vista Ayuda

Portapapeles Fuente Párrafo Estilos Edición

calcular los árboles de manera **post-ordenada** y **pre-ordenada** fácilmente.

3.5 La complejidad de todos los métodos es de  $O(n)$  ya que, para imprimir en **preorden** y **postorden** se debe pasar por todos los nodos del árbol. La complejidad de cualquiera de estos dos algoritmos, en términos de  $T(n)$  es  $2T(n/2)+c$ , lo cual nos lleva a una complejidad lineal. El método **buildingTree** tiene una complejidad de  $O(n \log n)$  puesto que, mediante un ciclo **for**, se recorre el arreglo dado en **preorden** para luego insertar cada elemento en su posición en el árbol. Finalmente, la complejidad final del algoritmo que soluciona el problema planteado es  $O(n) + O(n \log n)$ , lo cual se puede simplificar a  $O(n \log n)$ .

3.6 La variable  $n$  será el número de nodos que contiene el árbol binario dado.

**4) Simulacro de Parcial**

4.1

4.1.1 b

4.1.2 d

4.2 c

4.3

a. false

b. return suma == a.data

c. sumaElCamino(a.der, suma - a.data)

d. | sumaElCamino(a.izq, suma - a.data)

4.4

4.4.1 c

4.4.2 a

4.4.3 d

4.4.4 a

4.5

4.5.1 toInsert == p.data

4.5.2 toInsert > p.data

4.6

Historial de versiones

Modificado por:	Fecha
Modificado por: Valentina Moreno Ramirez	9:39 p. m.
Modificado por: Valentina Moreno Ramirez	9:36 p. m.
Modificado por: Valentina Moreno Ramirez	9:24 p. m.
Modificado por: Valentina Moreno Ramirez	9:21 p. m.
Modificado por: Valentina Moreno Ramirez	9:20 p. m.
Modificado por: Valentina Moreno Ramirez	8:23 p. m.
Modificado por: Alejandra Palacio Jaramillo	7:48 p. m.

**PhD. Mauricio Toro Bermúdez**

Docente | Escuela de Ingeniería | Informática y Sistemas  
 Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
 Tel: (+57) (4) 261 95 00 Ext. 9473



## ESTRUCTURA DE DATOS 1

### Código ST0245

calcular los árboles de manera post-ordenada y pre-ordenada fácilmente.

3.5 La complejidad de todos los métodos es de  $O(n)$  ya que, para imprimir en preorden y postorden se debe pasar por todos los nodos del árbol. La complejidad de cualquiera de estos dos algoritmos, en términos de  $T(n)$  es  $2T(n/2) + c$ , lo cual nos lleva a una complejidad lineal. El método buildingTree tiene una complejidad de  $O(n \log n)$  puesto que, mediante un ciclo for, se recorre el arreglo dado en preorden para luego insertar cada elemento en su posición en el árbol. Finalmente, la complejidad final del algoritmo que soluciona el problema planteado es  $O(n) + O(n \log n)$ , lo cual se puede simplificar a  $O(n \log n)$ .

3.6 La variable n será el número de nodos que contiene el árbol binario dado.

**4) Simulacro de Parcial**

```

4.1
4.1.1 b
4.1.2 d
4.2 c
4.3
  a. false
  b. return suma == a.data
  c. sumaElCamino(a.der, suma - a.data)
  d. || sumaElCamino(a.izq, suma - a.data)
4.4
4.4.1 c
4.4.2 a
4.4.3 d
4.4.4 a
4.5
4.5.1 !olnser == p.data
4.5.2 !olnser > p.data
4.6
  
```

**Historial de versiones**

Fecha	Modificado por	Acción
Hoy, 27 de octubre de 2020	Alejandra Palacio Jaramillo	Modificado por: Alejandra Palacio Jaramillo 12:59 a. m.
	Alejandra Palacio Jaramillo	Modificado por: Alejandra Palacio Jaramillo 12:56 a. m.
	Alejandra Palacio Jaramillo	Abrió versión
Ayer, 26 de octubre de 2020	Alejandra Palacio Jaramillo	Modificado por: Alejandra Palacio Jaramillo 11:42 p. m.
	Alejandra Palacio Jaramillo y Valentina Moreno Ramirez	Modificado por: Alejandra Palacio Jaramillo y Valentina Moreno Ramirez 4:09 p. m.
	Alejandra Palacio Jaramillo y Valentina Moreno Ramirez	Modificado por: Alejandra Palacio Jaramillo y Valentina Moreno Ramirez 4:09 p. m.
25 de octubre de 2020	Alejandra Palacio Jaramillo	Modificado por: Alejandra Palacio Jaramillo 11:58 p. m.
	Alejandra Palacio Jaramillo	Abrió versión

**PhD. Mauricio Toro Bermúdez**  
 Docente | Escuela de Ingeniería | Informática y Sistemas  
 Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627  
 Tel: (+57) (4) 261 95 00 Ext. 9473

