gthc.c low_level.rs

```
GREEN TEAM HACKER CLUB {
    [Programação Low Level]

        < Árvores >

}
```

Tabela de ‘Conteúdo’ {

01 Binary Tree

02 AVL Tree

03 B-Tree

}

gthc.c

low_level.rs

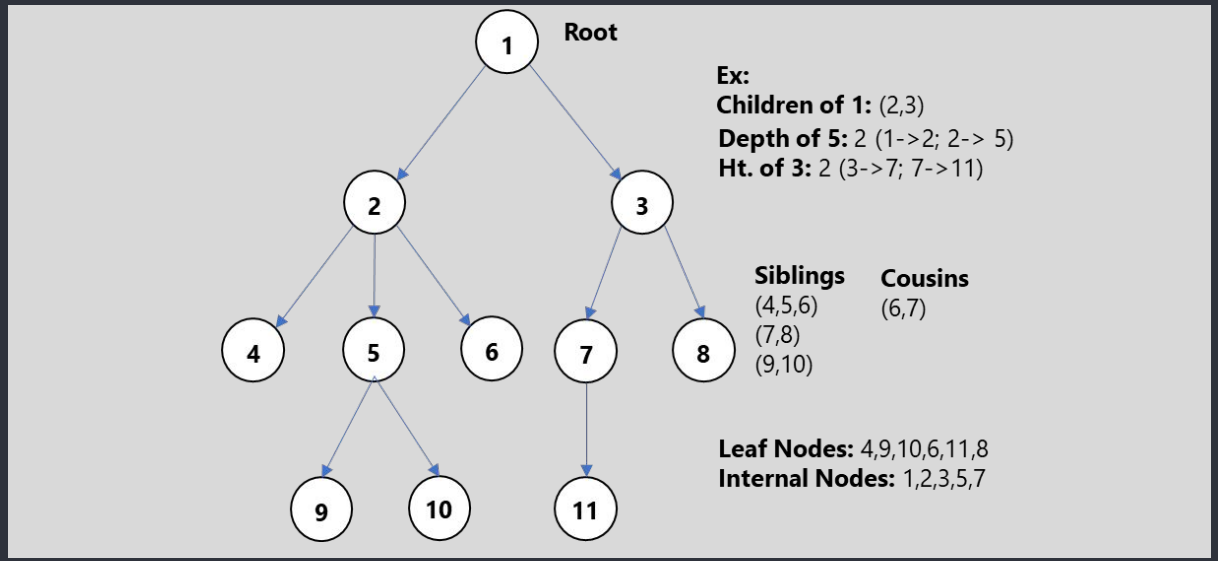
```
01 {
    [Binary Tree]
}
```


Definição < /1 > {

Uma árvore binária é uma estrutura de dados hierárquica não linear na qual cada nó tem no máximo dois filhos conhecidos como filho esquerdo e filho direito. Ela pode ser visualizada como uma estrutura hierárquica onde o nó mais alto é chamado de nó raiz e os nós na parte inferior são chamados de nós folha ou folhas.

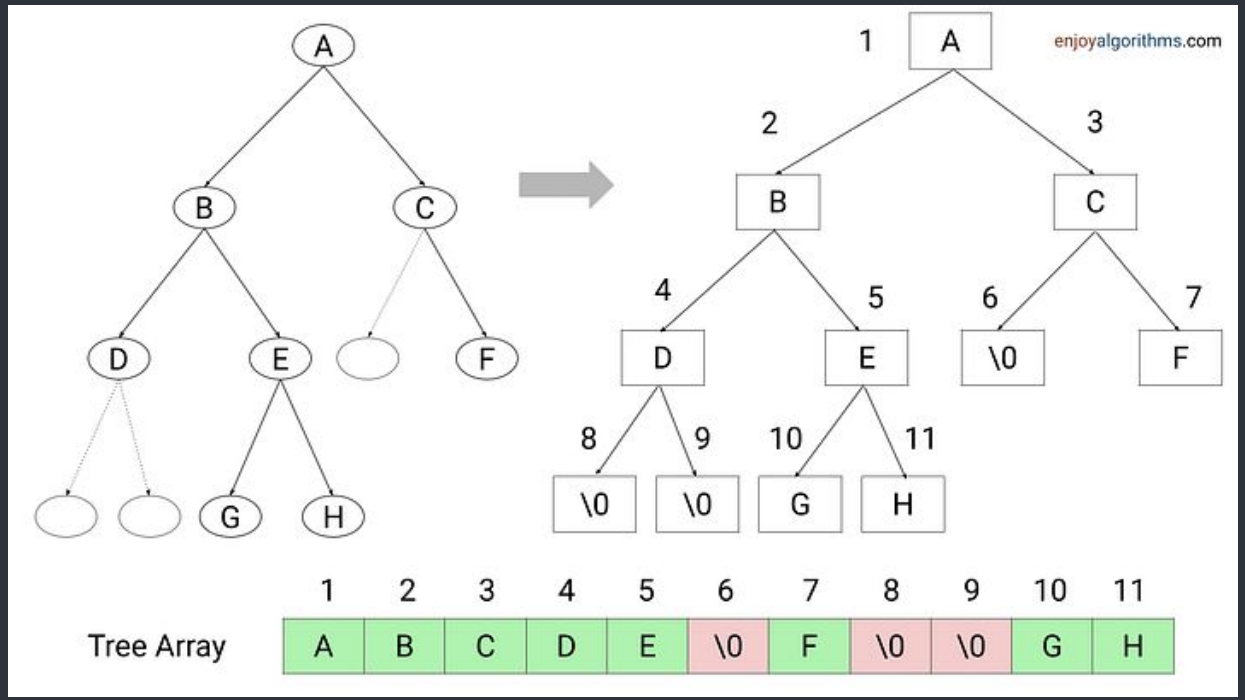
}

Definição < /1 > {


1
Root
2
3
4
5
6
7
8
9
10
11
Ex:
Children of 1: (2,3)
Depth of 5: 2 (1->2; 2-> 5)
Ht. of 3: 2 (3->7; 7->11)
Siblings
(4,5,6)
(7,8)
(9,10)
Cousins
(6,7)
Leaf Nodes: 4,9,10,6,11,8
Internal Nodes: 1,2,3,5,7

}

Definição < /1 > {



}

Tempo < /2 > {

	AVL Tree		Binary Search Tree	
	Average	**Worst**	**Average**	**Worst**
Insertion	O(logn)	O(logn)	O(logn)	O(n)
Deletion	O(logn)	O(logn)	O(logn)	O(n)
Searching	O(logn)	O(logn)	O(logn)	O(n)
Traversal	O(n)	O(n)	O(n)	O(n)

}

Operações < /1 > {

A implementação da estrutura de dados da rvore pode fornecer algumas das seguintes operações:

- Insertion
- Deletion
- Search
- Traversing (Pre-order, Post-order and In-order)

}

Implementação< /1 > {

https://gist.github.com/tsprates/8cefd5c7f855120c83c9462ddb99b845

}

gthc.c

low_level.rs

```
02 {
    [AVL]
}
```

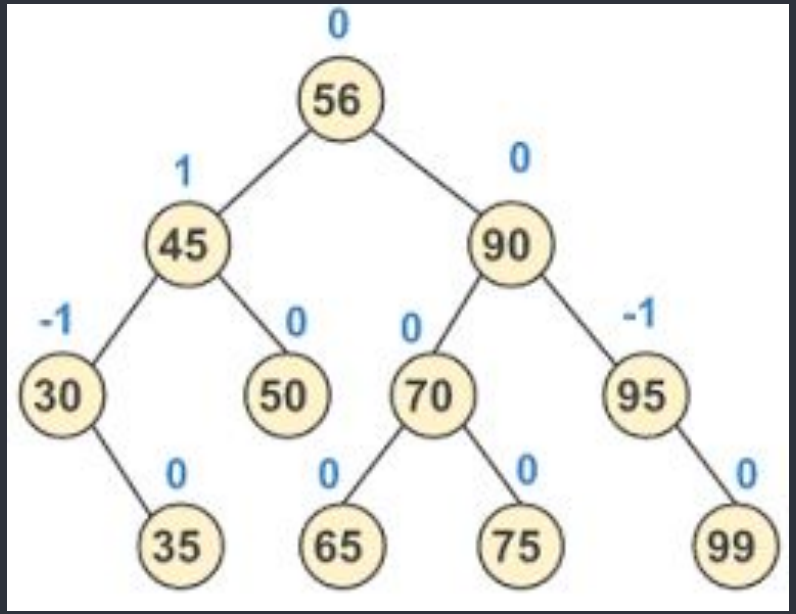
UFABC

Definição < /1 > {

A árvore AVL é uma Árvore Binária de Busca (BST) autobalanceada, onde a diferença entre as alturas das subárvores esquerda e direita não pode ser maior que um para todos os nós.

}

Definição < /1 > {



}

Tempo < /2 > {

	AVL Tree		Binary Search Tree	
	Average	**Worst**	**Average**	**Worst**
Insertion	O(logn)	O(logn)	O(logn)	O(n)
Deletion	O(logn)	O(logn)	O(logn)	O(n)
Searching	O(logn)	O(logn)	O(logn)	O(n)
Traversal	O(n)	O(n)	O(n)	O(n)

}

Operações < /1 > {

A implementação da estrutura de dados da lista pode fornecer algumas das seguintes operações:

- Right Rotation
- Left Rotation

}

Implementação< /1 > {

https://github.com/xieqing/avl-tree/blob/master/avl_bf.c

}

gthc.c

low_level.rs

```
03 {
    [B-Tree]
}
```

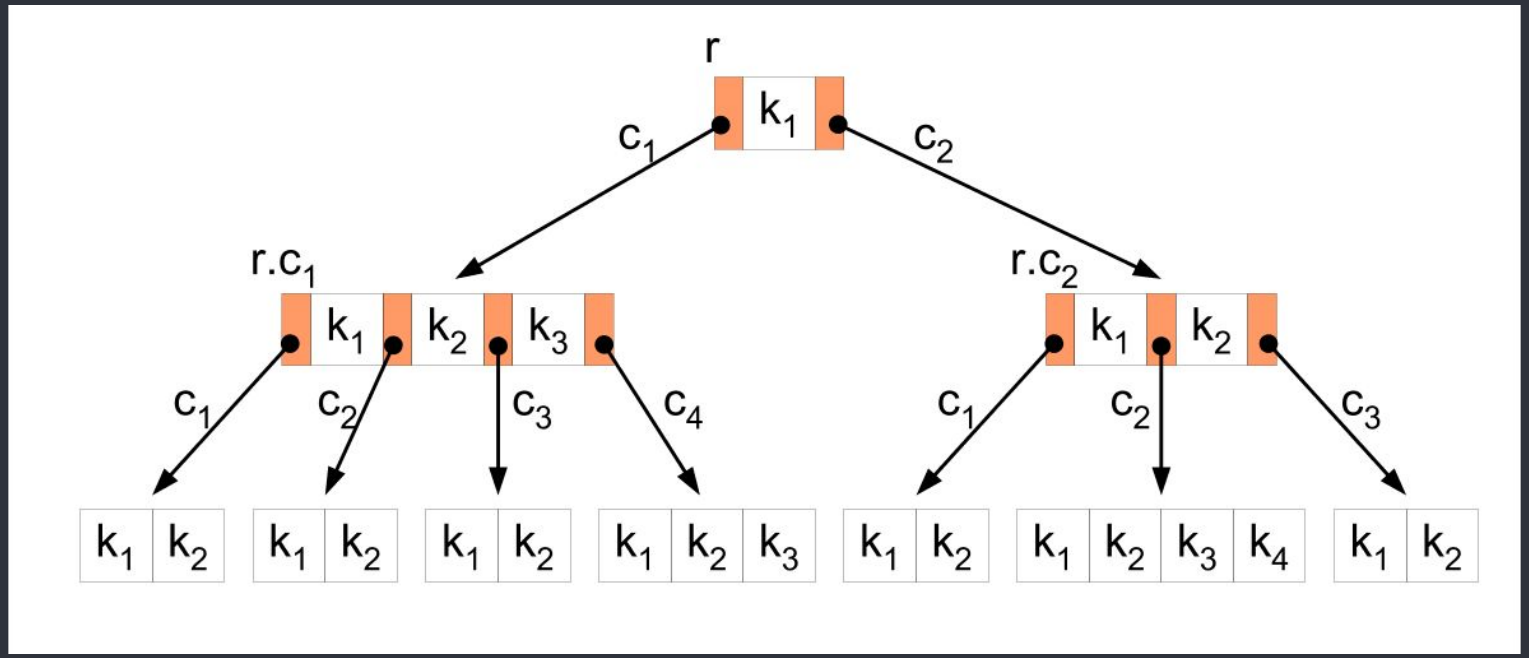

Definição < /1 > {

Uma B-Tree é uma árvore m-way especializada projetada para otimizar o acesso a dados, especialmente em sistemas de armazenamento baseados em disco.

Em uma B-Tree de ordem m, cada nó pode ter até m filhos e m-1 chaves, permitindo que ele gerencie com eficiência grandes conjuntos de dados.
O valor de m é decidido com base nos tamanhos de bloco e chave do disco.

}

Definição < /1 > {


r
k_1
c_1
c_2
$r.c_1$
k_1 k_2 k_3
c_1 c_2 c_3 c_4
k_1 k_2
k_1 k_2
k_1 k_2
k_1 k_2 k_3
$r.c_2$
k_1 k_2
c_1 c_2 c_3
k_1 k_2
k_1 k_2 k_3 k_4
k_1 k_2

Tempo < /2 > {

Operation	Time Complexity: Worst Case
Search	O(log n)
Insert	O(n)
Delete	O(log n)
Traverse	O(n)

}

Operações < /1 > {

https://www.youtube.com/watch?v=K1a2Bk8NrYQ

}

Implementação< /1 > {

https://github.com/falcaopetri/B-Tree/blob/master/src/btree.c

}

Bônus: Aplicações< /1 > {

- Sistemas operacionais
- Compiladores
- Bancos de Dados
- Criptografia
- Binary Heap

}

Visualização {

<Data Structure Visualizations - https://www.cs.usfca.edu/~galles/visualization/Algorithms.html >

<https://visualgo.net/en>

}

Referências {

<Data Structures Using C 2nd edition - https://github.com/GauravWalia19/Free-Algorithms-Books/blob/main/Library/src/C/Data-Structures-Using-C-2nd-edition.pdf >

}

Próximo Encontro {

< Arquitetura >

}

```
Obrigado; {

    'Dúvidas?'

        luccas.h.cortes@hotmail.com
        https://github.com/Cortesz/

                CREDITS: This presentation template was
                created by Slidesgo, including icons by
                Flaticon, and infographics & images by Freepik

                < https://github.com/greenteamhc >

}
```