# Implementação de AVL

O projeto consiste em uma simples demonstração de uso de uma árvore AVL que aceita (pelo usuário) duas operações: ADICIONAR e REMOVER

Além da parte demonstrativa do programa, o código do projeto tem todo um TAD implementado para futuros usos desta mesma árvore AVL como uma biblioteca funcional, além de adicionar e remover.

## Uso do programa (./myavl)

### Rodando o programa (básico)

```
 # Criar executaveis
make;

# Executar programa (limpo)
./myavl;

# Exemplo de uso do programa (mais funcional)
./myavl < input.in > output.ou;

# Limpar executáveis
make clean;
```

### Utilizando o programa

- O programa aceita dos tipos de entradas
- O programa consiste em linhas de entradas (inserir ou remover) para a sua AVL, que ao fim do programa imprime uma arvore AVL em pre-ordem com os niveis
- Inserir numero: `i 10`
- Remover numero: `r 10`
    - Se o numero nao existir, apenas ignora o pedido
- Para parar o programa e imprimer a arvore, basta chegar em uma EOF ou colocar um "modo" incompativel (diferente de I ou R)
- Exemplo:

```
 i 3
i 4
r 2
```

### Output

- No fim imprime uma arvore AVL de forma `numero, level`
- Raiz sendo o numero 0

```
 10, 1
15, 0
20, 1
```

## Arquivos

- `avl.h` : assinaturas das funções do TAD de AVL, além da struct de nodo utilizado por toda a árvore.
- `avl.c` : implementação das funções de manipulação de AVL, além de funções indiretas que auxilam esse processo.
- `main.c` : programa promiamente dito, além de ter o main( )

## Funcionalidades

### Funções indiretas (não podem ser chamadas)

- `Node* __create_node(int v, Node* father)` - uma função que cria um nodo novo (alocado dinamicamente) com um certo pai definido
- `int __balance_value(Node* n)` - acha o valor de balanceamento de um nodo, sendo 2 para desbalanceado na direita e -2 para desbalanceado a esquerda.
- `Node* __adjustment_avl(Node *n)` - apenas verifica se precisa de balanceamento e faz o ajuste da arvore nos moldes da AVL (usado no inserir e deletar)

### Funções de TAD AVL

- `Node* create_avl(int v)` - cria a arvore avl e retorna seu ponteiro de forma dinamica - util para o primeiro elemento
- `Node* insert_avl(int v, Node **root)` - insere um elemento numa arvore avl ja criada previamente e balancea a AVL
- `Node* rotate_tree_avl_left(Node *n)` - rotaciona a arvore para a esquerda
- `Node* rotate_tree_avl_right(Node *n)` - rotaciona a arvore para a direita
- `void view_avl(Node *n)` - imprime a avl de forma "numero, level" em uma pre-ordem (ordenado)
- `Node* search_avl(int v, Node *root)` - procura um elemento e retorna seu nodo caso seja encontrado (retorna o primeiro numero encontrado)
- `size_t height_avl(Node *n)` - retorna o tamanho da arvore atual
- `Node* delete_avl(int v, Node *root)` - deleta um numero definido, caso nao tenha o numero, apenas nao faz nada. Caso encontre, deleta o numero e balanceia a arvore
- `void destroy_avl(Node *root)` - destroi a avl no fim, desalocando toda a memoria alocada.

## Créditos


- Feito por Gustavo Benitez Frehse
- Informática Biomédica - GRR20235087
- Algoritmos III
- 16/10/2024