

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS – CAMPUS FORMIGA**

Leandro Souza Pinheiro

**Documentação trabalho de Estrutura de Dados:**

**Gerenciador de árvores AVL**

FORMIGA-MG

2018

Leandro Souza Pinheiro

**Documentação trabalho de Estrutura de Dados:**

**Gerenciador de árvores AVL**

Documentação, referente ao trabalho de Estrutura de Dados sobre árvores binarias com utilização de AVL para balanceamento.

Professora: Denise Garcia Rezende

FORMIGA-MG

2018

**1 INTRODUÇÃO**

A globalização vem provocando mudanças em todos os tipos de organizações, em diversos setores. São transformações de todas as naturezas que forçam as empresas a se adaptarem, a fim de manterem sua posição no mercado (KERZNER, 2006). Segundo Gonçalves, (2000a, p. 13) “O futuro vai pertencer às empresas que conseguirem explorar o potencial da centralização das prioridades, as ações e os recursos nos seus processos”. Mediante a aplicação de estudos no âmbito de processos, as empresas podem se movimentar para um melhor entendimento das atividades internas e responder às mudanças com um sistema competitivo.

Com isso esse projeto de pesquisa consiste em ampliar o sistema que visa gerenciar melhor as reuniões gerenciadas pelo IFMG-Campus Formiga, como também todas as fases necessárias para a realização das mesmas. Esse sistema é dividido em quatro módulos, Pré-reunião, Reunião, Pós-Reunião e Busca.

O módulo de Pré-reunião consiste em definir os setores que será realizada a reunião, assim como os participantes que participará da mesma. Para isso é necessário receber informações como, data e horário da reunião, itens de pauta a ser discutidos, e arquivos a serem anexados para realizar a reunião.

Já o módulo Reunião é definido como responsável por registrar todos os acontecimentos ocorridos durante a reunião, como por exemplo registrar os participantes presentes e ausentes da reunião, marcar o tempo de cada item de pauta e registrar os posicionamentos dos participantes e os encaminhamentos que serão anexados na Ata. Vale ressaltar que ao final desse módulo será gerado uma versão inicial da Ata que será enviada para revisão e salva no sistema para possíveis modificações que serão aprovadas por todos os participantes.

No módulo Pós-reunião é responsável por gerenciar toda a parte de revisão das atas, nele os participantes irão revisar, concluir e assinar a ata, para que assim o responsável pela mesma a feche, finalize e publique para toda a comunidade interna e externa.

A parte de Busca é o módulo que efetuará buscas por atas que já foram publicadas.

**2 MateriaIS e MétodoS**

Nessa seção serão apresentados todos os conceitos e ferramentas que foram utilizados para a criação do sistema de gerenciamento de árvores.

**2.1 Implementação e Back-end do sistema**

* C: Para a implementação e execução do Back-end do sistema foi utilizado a linguagem de programação [estruturada](https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_estruturada), [imperativa](https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_imperativa), [procedural](https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_procedural), [padronizada pela ISO](https://pt.wikipedia.org/wiki/Organiza%C3%A7%C3%A3o_Internacional_para_Padroniza%C3%A7%C3%A3o), criada em [1972](https://pt.wikipedia.org/wiki/1972), por [Dennis Ritchie](https://pt.wikipedia.org/wiki/Dennis_Ritchie), no [AT&T](https://pt.wikipedia.org/wiki/AT%26T) [Bell Labs](https://pt.wikipedia.org/wiki/Bell_Labs), para desenvolver o [sistema operacional](https://pt.wikipedia.org/wiki/Sistema_operativo) [Unix](https://pt.wikipedia.org/wiki/Unix) (que foi originalmente escrito em [Assembly](https://pt.wikipedia.org/wiki/Linguagem_de_montagem)). C é uma das linguagens de programação mais populares e existem poucas arquiteturas para as quais não existem compiladores para C. C tem influenciado muitas outras linguagens de programação, mais notavelmente [C++](https://pt.wikipedia.org/wiki/C%2B%2B), que originalmente começou como uma extensão para C.

**3 Descrição das principais funções**

Para a implementação do sistema foi criado uma serie de funções que foram divididas em bibliotecas visando uma melhor prática de programação, essa seção irá explicar todas as bibliotecas que foram desenvolvidas para a criação do sistema.

**3.1 tadArvore.h**

Essa biblioteca é utilizada para criar o tipo abstrato de dados árvore, nela foram definidos seguintes structs e escopos de funções:

* Structs:

1. NO: Struct responsável por representar um ramo da árvore, nela são definidos um campo Inteiro que irá armazenar o valor salvo naquele ramo e os 2 ponteiros que serão utilizados para apontar para os nos a esquerda e a direita dessa determinada raiz.
2. ARVORE: Struct responsável por representar a arvore como um todo, essa struct possui um único elemento que é um apontador para a raiz da árvore.

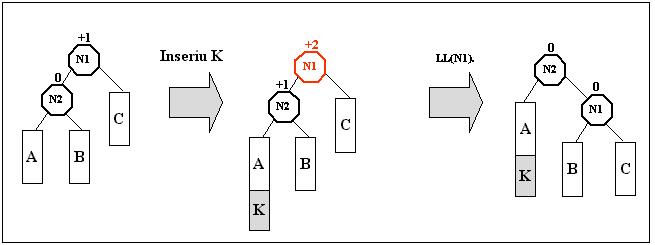
* Funções: (OBS: nesse tópico será apenas apresentado o escopo da função, no tópico tadArvore.c cada função será detalhadamente explicada para uma melhor compreensão.

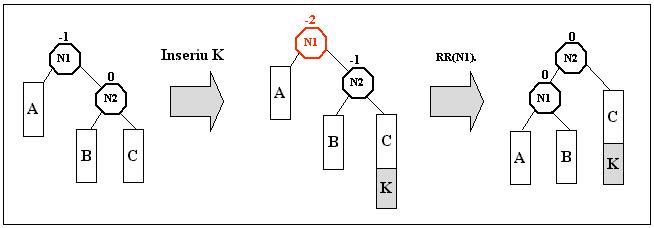
1. void seta\_vazia(Arvore \*t);
2. Arvore\* cria\_arvore();
3. int altura(No \*raiz);
4. int fator\_balanceamento(No \*raiz);
5. int verifica\_grau(No \*raiz);
6. int verifica\_nivel\_no(No \*no, int valor);
7. No\* rotacaoEsquerda(No \*no);
8. No\* rotacaoDireita(No \*no);
9. No\* rotacaoDE(No \*no);
10. No\* RotacaoED(No \*no);
11. No\* insere\_na\_arvore(No \*no,int valor);
12. void in\_ordem\_arquivo(No \*no, FILE \*arquivo);
13. void pre\_ordem\_arquivo(No \*no,FILE \*arquivo);
14. void pos\_ordem\_arquivo(No \*no,FILE \*arquivo);
15. boolean verifica\_vazia(Arvore \*t);
16. No\* verifica\_se\_existe\_valor\_arvore(No \*no, int valor);
17. No\* remove\_arvore(No\* no,int valor);
18. boolean apaga\_arvore(Arvore \*arvore);

Além das funções e structs essa biblioteca um typedef int boolean, que é utilizado para simular o tipo boolean, ou seja para que seja possível retorno de valores True e False.

**3.1 tadArvore.c**

* **Defines**: Para que seja possível retornar valores True e False foram definidas as constantes com os respectivos nomes e valores atribuídos em 0 e 1 para que seja possível o retorno desse tipo de valores.
* **void seta\_vazia(Arvore \*t):** Essa função é responsável por receber por referencia o ponteiro de arvore e apontar o mesmo para NULL para evitar erros de inserção de valores iniciais na árvore.
* **Arvore\* cria\_arvore():** Função responsável por alocar um ponteiro de árvore na memória, chamar a função que aponta a arvore para NULL e posteriormente retornar a árvore que foi criada.
* **int altura(No \*raiz):** Função responsável por receber um apontador de No e verificar qual a altura que aquele nó possui, inicialmente é verificado se o mesmo está apontado para NULL, caso esteja a função retorna False indicando que o nó não existe, caso contrario é chamado de forma recursiva todos os elementos a esquerda e empilhando os mesmos e depois todos os elementos a direita, assim é pegado a altura a esquerda e direita e verificado qual a maior para que assim seja retornado a altura somando 1 no valor, assim os valores na pilhas são incrementados e retornados no final.
* **int fator\_balanceamento(No \*raiz):** Função responsável por receber um NO da arvore e caso o mesmo não esteja apontado para NULL é chamado a função que verifica a altura a esquerda e a direita do NO recebido, feito isso é realizado o cálculo da diferença entre esquerda e a direita do NO e retornado esse fator para que possa ser verificado a necessidade de alguma rotação na árvore.
* **int verifica\_grau(No \*raiz):** Função extremamente simples, nela é recebido um apontador de algum NO presente na árvore e verificado qual grau aquele NO possui, ou seja se ele possui 1,2 ou 0 filhos, após a verificação é retornado o grau do NO recebido.
* **int verifica\_nivel\_no(No \*no, int valor):** Função responsável por verificar o nível em que se encontra algum elemento da arvore, para isso é recebido o NO raiz e o valor que se deseja verificar, para isso é verificado se o valor esta a direita ou esquerda da raiz atual e assim é caminhado em sua direção com a ajuda de um NO auxiliar e um contador que vai contando os níveis passados, ao encontrar o elemento ou o apontador auxiliar seja NULL é retornado o nível calculado ou FALSE indicando que o elemento não está na árvore.
* **No\* rotacaoEsquerda(No \*no):** Função responsável por executar uma rotação para a esquerda do NO recebido, para isso ela cria um auxiliar (q) que recebe o NO a direita e outro auxiliar (temp) que recebe o NO a esquerda da raiz recebida, feito isso o auxiliar a esquerda de “q” recebe o no raiz que foi passado por referência e NO, a direita da raiz recebe o auxiliar “temp”, após isso a nova raiz passa a ser o auxiliar “q” que é retornado. (COLOCAR FOTO)
* **No\* rotacaoDireita(No \*no):** Função responsável por executar uma rotação para a direita do NO recebido, para isso ela cria um auxiliar (q) que recebe o NO a esquerda e outro auxiliar (temp) que recebe o NO a direita da raiz recebida, feito isso o NO a direita do auxiliar a esquerda de “q” recebe o no raiz que foi passado por referência e NO e por fim a esquerda da raiz recebe o auxiliar “temp”, após isso a nova raiz passa a ser o auxiliar “q” que é retornado. (COLOCAR FOTO)
* **No\* rotacaoDE(No \*no):** Função responsável por executar uma rotação a direita do no e posteriormente uma rotação a esquerda, para isso é apenas chamado as funções de rotação a direita e esquerda nessa ordem.
* **No\* RotacaoED(No \*no):** Função responsável por executar uma rotação a esquerda do no e posteriormente uma rotação a direita, para isso é apenas chamado as funções de rotação a esquerda e direita nessa ordem.
* **insere\_na\_arvore** **(No \*no,int valor):** Função extremamente importante para o funcionamento da árvore, nela são executados os procedimentos necessários para a verificação de balanceamento após inserir um elemento na arvore. Para inserir um elemento é recebido um ponteiro contendo um NO da arvore e o valor a ser inserido, primeiramente é verificado se o NO esta a apontado para NULL, caso esteja é alocado um NO na memória, inserido o valor nele e apontando os apontadores de NO a esquerda e direita para NULL para representar que foi criado um novo NO folha. Caso o NO recebido não esteja apontado para NULL é verificado se o valor passado por parâmetro é maior ou menor que o NO atual, caso seja maior é chamado de forma recursiva a função para inserir na árvore, porém com o ponteiro do NO a direita do inicial, contudo caso o valor seja menor também é chamado de forma recursiva porém agora com o NO a esquerda do ponteiro inicial, caso o elemento seja igual ao NO atual ele não é inserido e é apenas retornado o NO atual para que seja liberado a pilha de recursão. Ao final da função é verificado o Fator de balanceamento em cada do NO que foi colocado na pilha de recursão, e caso o fator de balanceamento seja maior que 1 indica que será necessário uma rotação a direita do NO atual, para isso é chamado a função de rotação a direita e passado o respectivo NO, caso o fator de balanceamento seja menor que -1 indica que será necessário uma rotação a esquerda do NO atual, para isso é chamado a função de rotação a esquerda e passado o respectivo NO.As imagens abaixo mostram as rotações que podem ser necessárias após a inserção de algum elemento na arvore.



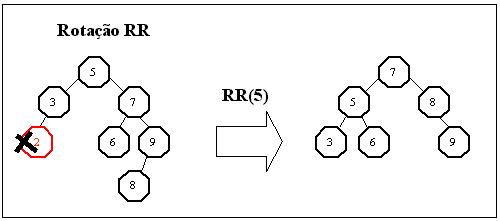
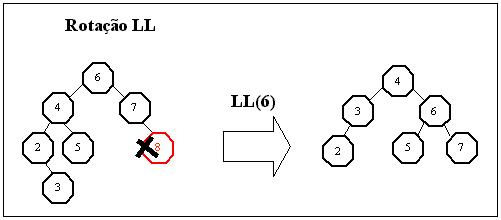


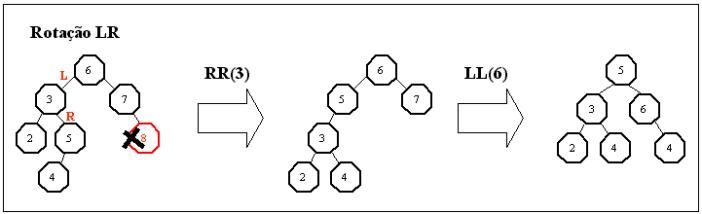
* **void in\_ordem\_arquivo(No \*no, FILE \*arquivo):** Função responsável por escrever no arquivo os elementos da árvore, para isso é recebido o NO raiz da arvore e um ponteiro de arquivo que precisa necessariamente já estar aberto para que o a escrita seja mais rápida, para isso é chamado de maneira recursiva todos os elementos a esquerda do NO raiz e escrito no arquivo esses elementos, depois é chamado um elemento a direita e empilhado na pilha de recursão, depois o ciclo de inicia novamente. Vale ressaltar que no inicio da função é verificado se o NO recebido não está apontado para NULL.
* **void pre\_ordem\_arquivo(No \*no,FILE \*arquivo):** Função responsável por escrever no arquivo os elementos da árvore, para isso é recebido o NO raiz da arvore e um ponteiro de arquivo que precisa necessariamente já estar aberto para que o a escrita seja mais rápida, assim é escrito no arquivo o elemento recebido e posteriormente é chamado de maneira recursiva os elementos a esquerda do NO raiz e, depois é chamado elemento a direita e empilhado na pilha de recursão, depois o ciclo de inicia novamente. Vale ressaltar que no início da função é verificado se o NO recebido não está apontado para NULL.
* **void pos\_ordem\_arquivo(No \*no,FILE \*arquivo):** Função responsável por escrever no arquivo os elementos da árvore, para isso é recebido o NO raiz da arvore e um ponteiro de arquivo que precisa necessariamente já estar aberto para que o a escrita seja mais rápida, com isso é chamado de maneira recursiva os elementos a esquerda do NO raiz e, depois é chamado elemento a direita e empilhado na pilha de recursão posteriormente é escrito no arquivo o elemento recebido, depois o ciclo de inicia novamente. Vale ressaltar que no início da função é verificado se o NO recebido não está apontado para NULL.
* **boolean verifica\_vazia(Arvore \*t):** Função responsável por verificar se a raiz da arvore está apontada para NULL, caso esteja indica que a arvore está vazia, com isso é retornado TRUE, caso contrario é retornado FALSE indicando que a árvore possui elementos.
* **No\* verifica\_se\_existe\_valor\_arvore(No \*no, int valor):** Função responsável por receber um valor por parâmetro e o NO raiz da arvore, caso o NO recebido seja NULL indica que o elemento não está presente na arvore, caso o valor recebido seja maior que o do NO atual indica que o elemento está a direita do NO atual caso seja menor indica que o elemento esta a esquerda e caso seja igual indica que o elemento foi encontrado, para isso é feito essas verificações e chamado de forma recursiva com o NO a esquerda ou direita dependendo do valor como descrito anteriormente.
* **boolean apaga\_arvore(Arvore \*arvore):** Função responsável por apagar a árvore por completo, para isso ela remove todos os elementos contidos na árvore, para efetuar a remoção dos elementos ela sempre apaga a raiz principal da arvore ate que a mesma seja apontada para NULL, quando isso ocorrer ela libera o ponteiro da árvore com o intuito de liberar a memória utilizada.
* **No\* remove\_arvore(No\* no,int valor):**

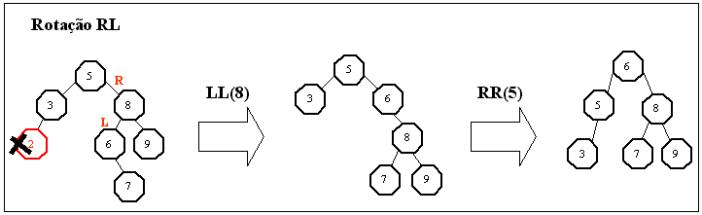
A função de remover elemento da arvore recebe por parâmetro um no e o valor que será removido, feito isso ela verifica se o no recebido esta apontado para NULL, caso esteja retorna NULL indicando que o elemento não existe na arvore. Caso o NO não seja NULL é verificado se o valor recebido é maior ou menor que o valor presente no NO atual, caso seja maior é chamado de forma recursiva para a direita do NO atual, caso seja menor chama de forma recursiva a esquerda do NO atual. Caso o elemento seja igual indica que esse NO deve ser removido, feito isso é necessário verificar os 3 possíveis casos, o primeiro caso indica que o NO a ser removido é uma folha, isso indica que ele pode ser removido sem maiores problemas. O segundo caso indica que o NO a ser removido possui apenas um filho, ou seja, um filho a esquerda ou um filho a direita, assim é criado um auxiliar que recebe o NO a ser removido, e o NO a ser removido passa a ser o elemento a direita ou a esquerda dependendo de onde o filho esteja. O terceiro caso considerado o mais difícil indica que o NO a ser removido possui 2 filhos, para tratar esse problema é criado um auxiliar que recebe o NO a esquerda do NO a ser removido e depois o auxiliar vai o mais à direita possível pegando o elemento antecessor do NO a ser removido, assim é feito a troca do NO a ser removido pelo seu antecessor e chamado de forma recursiva a função de remover o elemento da arvore que assim irá cair novamente em um dos 3 casos possíveis. Ao Final da função é verificado se é necessário realizar o balanceamento da arvore com base no NO a ser removido, para isso é chamado as funções de rotação a esquerda ou a direita dependendo do fator de balanceamento.

Como essa função é considerada a mais complexa do sistema, para facilitar o entendimento será exemplificado os casos em tópicos de como a função funciona tecnicamente:

1. Verificar se a pilha está vazia, se sim, o algoritmo termina, caso contrário vá para o passo 2
2. Desempilhar um nó e verificar se a diferença de altura entre a sub-árvore da esquerda e da direita desse nó é maior que 1. Caso seja maior é necessário rotacionar os nós. Dependendo do tipo de rotação realizada, o algoritmo pode não terminar aqui. Se ele não terminar, vá para o passo 1, senão, é executado o passo 1.

As imagens abaixo mostram as rotações que podem ser necessárias após a remoção de algum elemento na arvore. 





**3.2 desenvolvimento.h**

Nessa biblioteca possui um include para a biblioteca de árvore e o escopo da função para ler o arquivo. Essa função será explicada no tópico a seguir.

**3.3 desenvolvimento.h**

* **void le\_arquivo(char url\_entrada[],char url\_saida[],Arvore \*arvore):** Função responsável por executar a leitura do arquivo de entrada e gerar a saída no arquivo de saída. Para isso primeiramente é gerado um ponteiro para abrir o arquivo de entrada e outro para abrir o arquivo de saída.Feito isso é criado um LOOP para repetir ate que chegue ao fim do arquivo, assim é lido a primeira string ate encontrar o “espaço em branco”, e verificado qual a palavra foi lida, existem basicamente 4 palavras chaves que são:

1. **INCLUI:** Caso encontre essa palavra é lido qual valor será inserido na arvore e chamado a função para executar essa tarefa que foi especificado no TAD de arvore.
2. **EXCLUI:** Caso encontre essa palavra indica que será excluído o elemento lido, para isso é lido o valor a ser excluído e chamado a função responsável por executar essa tarefa que foi especificado no TAD de arvore.
3. **IMPRIME:** Caso encontre essa palavra é verificado qual tipo de impressão será executada, para isso é aberto o arquivo de saída que é fechado ao final desse caso, assim é lido o tipo de impressão que pode ser:
   1. **INORDEM:** Caso seja a impressão inordem é chamado a função que foi especificado no TAD de arvore.
   2. **POSORDEM:** Caso seja a impressão posordem é chamado a função que foi especificado no TAD de arvore.
   3. **PREORDEM:** Caso seja a impressão posordem é chamado a função que foi especificado no TAD de arvore.
4. **BUSCA:** Caso encontre essa palavra indica que será buscado o elemento lido, para isso é lido o valor a ser buscado e chamado a função responsável por executar essa tarefa que foi especificado no TAD de arvore.

* Ao final da função é fechado o arquivo de entrada para evitar erros.

**3.5 main.c**

**5 ConclusÃO**

Ao final do período definido todos resultados que eram esperados foram atingidos, contudo para serem desenvolvidos foi necessário realizar várias pesquisas e definir estratégias de desenvolvimento com o framework *Laravel* que iria atender as necessidades que foram especificadas.

Diante de todos os tópicos abordados esse projeto de pesquisa apresenta ser uma inovação enorme para o IFMG Campus formiga, pelo fato de todas as reuniões realizadas a partir de sua implementação seja realizada virtualmente de forma que facilite a discussão dos participantes fazendo com que as reuniões se tornem mais dinâmicas e produtivas.

Para uma possível continuação e melhorias desse projeto poderiam ser inseridas inúmeras inovações para melhorar a utilização dos usuários, além de inovações o projeto poderia ser ampliado para que possa ser utilizado em outros Campus do IFMG, tornando-se assim um sinônimo de inovação em todos os Campus.

**6 Referências**