Universidade Federal dos Vales do Jequitinhonha e Mucuri
Sistemas de Informação
Algoritmos e Estrutura de Dados III
UFVJM

Amanda Cristina Lopes Santos
Beatriz Cristina Alcântara
Henrique Teixeira Dumont

Trabalho Prático De AEDS III

Pseudocódigo da codificação das rotações de uma árvore AVL.

Diamantina - Minas Gerais
2020

Pseudocódigo

O que é?

Para um(a) programador(a) desenvolver um algoritmo, é necessário seguir uma sequência de passos, que resulta o início, meio e fim. A representação de um algoritmo pode ser vista em diversas situações: em um texto que explique um passo a passo, em uma lista com tarefas ordenadas ou até mesmo em um fluxograma. Não existe uma regra que formalize qual meio utilizar, esse conceito se torna um pouco distante do mundo das linguagens de programação. Foi para mudar essa realidade e aproximar os conceitos que o pseudocódigo foi desenvolvido.

Logo, apresentaremos o pseudocódigo da codificação das rotações de uma árvore AVL, desenvolvido para a disciplina de Algoritmos & Estruturas de Dados III, do ERE (ensino remoto emergencial).

Representação

INÍCIO

```
struct Node
{
        int informacao;
        Node *esquerda;
        Node *direita;
        int altura;
};
```

Explicação:

variável especial Node (nó em português) de uma árvore = criando a árvore avl
        criando uma variável do tipo inteiro chamada informação;
        Node (Nó) com um ponteiro variáveis que armazenam o endereço de memória de outras variáveis), apontando para a esquerda;
        Node (Nó) com um ponteiro variáveis que armazenam o endereço de memória de outras variáveis), apontando para a direita;
        criando uma variável do tipo inteiro chamada altura.

//definição da árvore, inicialização da raiz como null

```
Arv()
        {
                root = NULL;
        }
```

Explicação:

variável especial Arv (que faz referência a uma árvore)
        root (raíz em português) recebendo o valor NULL (um ponteiro nulo).
//liberação do valor da raiz.

```
Arv()
	{
		root = libera(root);
	}

	Node *root;
	Node *Inserir(Node *aux, int valor); //Inserir valor na árvore

	Node *Remover(Node *aux, int valor); //Remover valor na árvore
	Node *Pesquisar (Node *aux); //Pesquisar na subárvore a direita o menor valor
```

Explicação:

declarando a variável Arv;

root (raíz) recebe o parâmetro libera(raíz);

Node acessa root;
Node acessa Inserir e recebe os parâmetros (Node, *aux(função com passagem por referência), variável do tipo inteiro chamada valor);
Node acessa Remover e recebe os parâmetros (Node, *aux(função com passagem por referência), variável do tipo inteiro chamada valor);
Node acessaPesquisar e recebe os parâmetros (Node, *aux(função com passagem por referência).

```
bool Pesquisar2(Node *aux, int valor);

	int Altura(Node *aux); //mostrar a altura do nó
	int bigger_alt(int alt_esquerda, int alt_direita); //Conferir se a altura a direita ou
esquerda do nó definir o maior
	int fatorBalanceamento(Node *aux); //Fator de Balanceamento
	Node *RotacaoDE(Node *aux); //rotação (-2) -> (1).
	Node *RotacaoDD(Node *aux); //rotação (2) -> (1).
	Node *RotacaoEE(Node *aux); //rotação (-2) -> (-1).
	Node *RotacaoED(Node *aux); //rotação (2) -> (-1).

	void ImprimirPreOrdem(Node *aux);
	void *search(Node *aux, int valor);

	Node *libera(Node *root);
```

};

Explicação:

Criando uma variável do tipo boleana (o valor pode ser verdadeiro ou falso) chamada Pesquisar2, recebendo os parâmetros Node (nó) apontando para *aux(função com passagem por referência) e uma variável do tipo inteiro, chamada valor.

variável do tipo inteiro chamada Altura recebendo os parâmetros Node (nó) apontando para *aux(função com passagem por referência);
variável do tipo inteiro chamada bigger_alt recebendo os parâmetros alt_esquerda do tipo inteiro e alt_direita do tipo inteiro.

//Usado para conferir se a altura a direita ou esquerda do nó definir o maior.

Criando uma variável do tipo inteiro chamada fatorBalanceamento, que tem os parâmetros Node (nó) apontando para *aux(função com passagem por referência).
//Indica o fator de balanceamento.

A variável Node acessa RotacaoDE (direita, esquerda), que tem os parâmetros Node acessa *aux(função com passagem por referência);

A variável Node acessa RotacaoDD (direita, direita), que tem os parâmetros Node acessa *aux(função com passagem por referência);

A variável Node acessa RotacaoEE (esquerda, esquerda), que tem os parâmetros Node acessa *aux(função com passagem por referência);

A variável Node acessa RotacaoED (esquerda, direita), que tem os parâmetros Node acessa *aux(função com passagem por referência)

Retorna ImprimirPreOrdem que tem os os parâmetros Node acessa *aux(função com passagem por referência);
o endereço search com parâmetros (Node acessando *aux, int valor);
Node acessa libera(Node acessa raíz)

```
Node *Arv ::Inserir(Node *aux, int valor)
{
	//percorre a arvore até encontrar um nó vazio para inserir um valor
	if (aux == NULL)
	{
```

```
                aux = new Node;
                aux->direita = NULL;
                aux->esquerda = NULL;
                aux->informacao = valor;
                aux->altura = 0;
                if (root == NULL)
                        root = aux;
        }
        else if (valor < aux->informacao)
        {
                aux->esquerda = Inserir(aux->esquerda, valor);
        }
        else if (valor > aux->informacao)
        {
                aux->direita = Inserir(aux->direita, valor);
}


        //Confere se a árvore está balanceada, se não, faz a rotação
```

Explicação:

Node aponta para Arv e busca a função inserir, recebe os parâmetros Node, apontando para *aux(função com passagem por referência) e uma variável do tipo inteiro, chamada valor;
Então,
Se aux for igual a NULL:
        aux recebe um novo nó;
        aux aponta para a direita e recebe NULL;
        aux aponta para esquerda e recebe NULL;
        aux aponta para informacao e recebe valor;
        aux aponta altura e recebe valor 0;
Se raíz for igual a NULL;
        root = aux;

Senão, se, valor for menor que aux, então aponta para informação,
Então,
        aux aponta para esquerda e recebe Inserir(aux apontando para a esquerda, valor

Senão, se, valor for maior que aux, aponta para direita, valor

```
        if (fatorBalanceamento(aux) == 2)
```

```
        {
                if (fatorBalanceamento(aux->esquerda) == 1)
                {
                        aux = RotacaoDD(aux);
                }
                else
                {
                        aux = RotacaoED(aux);
                }
        }
        else if (fatorBalanceamento(aux) == -2)
        {
                if (fatorBalanceamento(aux->direita) == -1)
                {
                        aux = RotacaoEE(aux);
                }
                else
                {
                        aux = RotacaoDE(aux);
                }
        }

        //atualiza a altura dos nós
        aux->altura = bigger_alt(Altura(aux->esquerda), Altura(aux->direita)) + 1;

        return aux;
}
```

Explicação:

Se fatorBalanceamento(aux) for igual a 2,
        Se fatorBalanceamento(aux -> esquerda) for igual a 1,
                aux recebe rotação para direita direita (DD)
Senão,
        aux recebe rotação esquerda direita (ED);
Senão, se,
        fatorBalanceamento(aux) for igual a -2
Se, fatorBalanceamento(aux->direita) for igual a -1,
        aux recebe rotação esquerda esquerda (EE)
Senão,
        aux recebe rotação direita esquerda (DE)

aux aponta para altura e recebe bigger_alt, recebendo os parâmetros (Altura(aux->esquerda), Altura(aux->direita)) + 1

e, retornando aux.

```
Node *Arv ::Remover(Node *aux, int valor)
{

	if (aux == NULL)
	{
		return NULL;
	}

	//caminha na arvore até encontrar o valor será removido
	if (valor == aux->informacao)
	{
		//Se o nó a ser removido possuir 1 ou zero filhos
		if (aux->direita == NULL || aux->esquerda == NULL)
		{
			Node *Node = aux;
			if (aux->esquerda != NULL)
			{
				aux = aux->esquerda;

			}
			else
			{
				aux = aux->direita;
			}

			free (Node);
		}
		//Se o nó a ser removido possuir dois filhos
```

Explicação:

Node aponta para Arv e busca a função remover, recebe os parâmetros Node, apontando para *aux(função com passagem por referência) e uma variável do tipo inteiro, chamada valor;

Se aux for igual a NULL
retorna NULL

Se, valor for igual aux aponta para informação

Se aux apontar para direita e for igual a NULL ou aux apontar para a esquerda e for igual a NULL,

Node aponta para Node e recebe aux

Se aux aponta para esquerda for diferente de NULL,
aux recebe aux apontando a esquerda
Senão se, aux recebe aux apontando a direita

```
Node *Arv ::Remover(Node *aux, int valor)
{

	if (aux == NULL)
	{
		return NULL;
	}

	//caminha na arvore até encontrar o valor será removido
	if (valor == aux->informacao)
	{
		//Se o nó a ser removido possuir 1 ou zero filhos
		if (aux->direita == NULL || aux->esquerda == NULL)
		{
			Node *Node = aux;
			if (aux->esquerda != NULL)
			{
				aux = aux->esquerda;

			}
			else
			{
				aux = aux->direita;
			}

			free (Node);
		}
		//Se o nó a ser removido possuir dois filhos


else
		{
			Node *No;
			No = Pesquisar(aux->direita);
			aux->informacao = No->informacao;
			aux->direita = Remover(aux->direita, aux->informacao);
```

```
            }
        }
        else if (valor < aux->informacao)
        {
            aux->esquerda = Remover(aux->esquerda, valor);
        }
        else if (valor > aux->informacao)
        {
            aux->direita = Remover(aux->direita, valor);
        }


        //Confere se a Arv está balanceada, se não, faz a rotação
```

Explicação:

Senão,

Node aponta para No

Node recebe Pesquisar(aux apontando a direita);

aux apontando a direta recebe Remover(aux apontando a direta, aux apontando para informacao)

Senão,se,

valor for menor que aux apontando para informação

aux aponta a esquerda e recebe Remover(aux apontando a esquerda, valor;

Senão, se,

valor for maior que aux apontando a direita, recebe Remover(aux apontando a direita, valor.

Se fatorBalanceamento(aux) for igual a 2

Se fatorBalanceamento(aux apontando a esquerda) for igual a 1,

aux recebe rotação direita direita(DD)

Senão;

aux recebe rotação esquerda direita (ED)

Senão se,

fatorBalanceamento(aux) for igual a -2,

fatorBalanceamento(aux apontando a direita) for igual a -1

aux recebe rotação esquerda esquerda (EE)

aux recebe rotação direita esquerda (DE)

e retorna aux.

```
	if (fatorBalanceamento(aux) == 2)
	{
		if (fatorBalanceamento(aux->esquerda) == 1)
		{
			aux = RotacaoDD(aux);
		}
		else
		{
			aux = RotacaoED(aux);
		}
	}
	else if (fatorBalanceamento(aux) == -2)
	{
		if (fatorBalanceamento(aux->direita) == -1)
		{
			aux = RotacaoEE(aux);
		}
		else
		{
			aux = RotacaoDE(aux);
		}
	}


	return aux;
}



Node * Arv :: Pesquisar (Node *aux)
{
	Node *Node1 = aux;
	Node *Node2 = aux->esquerda;

	while (Node2 != NULL)
	{
		Node1 = Node2;
		Node2 = Node2->esquerda;
	}

	return Node1;
```

```
}
```

Explicação:

Node acessa Arv e busca função Pesquisar (Node acessa aux)

Node acessa Node1 e recebe aux;
Node acessa Node2 e recebe aux apontando a esquerda

Então, executa a estrutura de repetição:
Node2 diferente de NULL
Node1 recebe Node2;
Node2 recebe Node2 apontando a esquerda;
retorna Node1.

```
int Arv ::bigger_alt(int alt_esquerda, int alt_direita)
{
	if (alt_esquerda > alt_direita)
	{
		return alt_esquerda;
	}
	else
	{
		return alt_direita;
	}
}
```

Explicação:

Arv do tipo inteiro busca a função bigger_alt(int_esquerda, int alt_direita)

Se,
alt_esquerda for maior que alt_direita
retorna alt_esquerda;

Senão,
retorna alt_direita

```
int Arv ::Altura(Node *aux)
{

	if (aux == NULL)
		return -1;
	else
	{
		return aux->altura;
	}
}

//Fator de Balanceamento
```

Explicação:

Arv do tipo inteiro busca Altura(Node acessa aux)
	Se,
		aux for igual a NULL
			retorna -1;
	Senão,
			retorna aux que aponta para altura

```
int Arv ::fatorBalanceamento(Node *aux)
{
	if (aux == NULL)
		return 0;
	return (Altura(aux->esquerda) - Altura(aux->direita));
}
```

Explicação:

Arv do tipo inteiro busca fatorBalanceamento(Node acessa aux)
	Se,
	aux for igual a NULL
		retorna 0;
	retorna (Altura(aux aponta a esquerda) - Altura(aux aponta a direita)

```
Node *Arv ::RotacaoDD(Node *aux)
```

```
{
	Node *Node;

	Node = aux->esquerda;
	aux->esquerda = Node->direita;
	Node->direita = aux;

	//Atualiza a altura dos nós
	aux->altura = bigger_alt(Altura(aux->esquerda), Altura(aux->direita)) + 1;
	Node->altura = bigger_alt(Altura(Node->esquerda), aux->altura) + 1;

	//Se houve mudança do nó raiz
	if (aux == root)
	{
		root = Node;
	}

	//Se houve mudança apenas Nodes filhos
	aux = Node;

	return aux;
}
```

Explicação:

Node acessa Arv, busca RotacaoDD(Node acessa aux)
	Node acessa Node;
	Node recebe aux que aponta a esquerda;
	aux aponta a esquerda e recebe Node que aponta a direita

//Atualiza a altura dos nós

	aux aponta altura e recebe bigger_alt(Altura(aux aponta a esquerda), aux aponta para altura) +1;

//Se houve mudança do nó raiz

	Se,
	aux for igual a root (raíz)
		root recebe Node;

//Se houve mudança apenas Nodes filhos
	aux recebe Node;
	retorna aux

```
Node *Arv ::RotacaoEE(Node *aux)
{
	Node *Node;

	Node = aux->direita;
	aux->direita = Node->esquerda;
	Node->esquerda = aux;
	aux->altura = bigger_alt(Altura(aux->esquerda), Altura(aux->direita)) + 1;
	Node->altura = bigger_alt(aux->altura, Altura(Node->direita)) + 1;
	if (aux == root)
	{
		root = Node;
	}
	aux = Node;

	return aux;
}
```

Explicação:

Node acessa Arv, busca RotacaoEE(Node acessa aux)
	Node acessa Node;

	Node recebe aux apontando a direita;
	aux apontando a direita, rece Node apontando a esquerda
	Node aponta esquerda que recebe aux;
	Node apontando altura, rece bigger_alt(aux apontando altura, Altura(Node apontando a direita)) +1;
	Node aponta altura e recebe bigger_alt(aux apontando altura, Altura(Node apontando a direita)) +1;
Se,
aux for igual a root
	root recebe Node;
aux = Node;
retorna aux;

```
Node *Arv ::RotacaoED(Node *aux)
{
	aux->esquerda = RotacaoEE(aux->esquerda);
	aux = RotacaoDD(aux);
```

```
        return aux;
}
```

Explicação:

Node acessa Arv, busca RotacaoED(Node acessa aux)

aux aponta a esquerda e recebe RotacaoEE(aux apontando a esquerda)
aux recebe RotacaoDD(aux)

retorna aux;

```
Node *Arv ::RotacaoDE(Node *aux)
{
        aux->direita = RotacaoDD(aux->direita);
        aux = RotacaoEE(aux);

        return aux;
}
```

Explicação:

Node acessa Arv, busca RotacaoDE(Node acessa aux)

aux aponta a direita e recebe RotacaoDD(aux apontando a direita)
aux recebe RotacaoEE(aux)

retorna aux;

```
void *Arv ::search(Node *aux, int valor)
{
        if (aux != NULL)
        {
                if (valor < aux->informacao)
                {
                        return search(aux->esquerda, valor);
                }

                else if (valor > aux->informacao)
```

```
		{
			return search(aux->direita, valor);
		}
		else if (valor == aux->informacao)
		{
			cout << "Valor encontrado" << endl;
		}
		return aux;
	}
	else if (root == NULL)
	{
		cout << "Lista Vazia" << endl;
	}

	cout << "Nao encontrado" << endl;
	return 0;
}
```

Explicação:

A função do tipo void acessa Arv, busca search (Node acessa aux, int valor)

Se,
aux for diferente de NULL;
Se,
valor for menor que aux apontando para informacao;
retorna search(aux apontando a esquerda, valor);
Senão, se
valor for maior que aux apontando para informacao;
retorna search(aux apontando a direita, valor);
Senão, se,
valor for igual aux apontando a direita, valor;
Senão se,
valor for igual aux apontando para informacao

imprime na tela “Valor encontrado”
retorna aux;

Senão, se,
root for igual a NULL,
imprime na tela “Lista Vazia”
imprime na tela “Não encontrado”
retorna 0;

```
bool Arv::Pesquisar2(Node* aux, int valor){
```

```
   if(aux==NULL){

      return false;
      }

   else if(aux->informacao== valor){
      return true;
   }
else{

      if(valor<aux->informacao){
         Pesquisar2(aux->esquerda, valor);
      }
      else {
          Pesquisar2(aux->direita, valor);
      }

   }
```

Explicação:

Arv do tipo boleano busca Pesquisar2(Node acessa aux, int valor)
Se,
aux for igual a NULL
retorna falso;
Senão, se,
aux apontando para informacao for igual valor
retorna verdadeiro
Senão,
Se
valor for menor que aux apontando para informacao;
Pesquisae2(aux apontando a esquerda, valor)
Senão
Pesquisar2(aux apontando a esquerda, valor)
Senão,
Pesquisar2(aux apontando a direita, valor)

```
void Arv ::ImprimirPreOrdem(Node *aux)
{
        if (aux != NULL)
        {
                cout << "O elemento visitado foi: " << aux->informacao
```

```
                << " - Altura: " << aux->altura
                << " - Balanceamento: " << fatorBalanceamento(aux) << endl;
        ImprimirPreOrdem(aux->esquerda);
        ImprimirPreOrdem(aux->direita);
    }
}
```

Explicação:

Arv do tipo void busca ImprimirPreOrdem(Node acessa aux)
    Se,
    aux for diferente de NULL;
        imprime na tela “O elemento visitado foi: " << aux->informacao
            << " - Altura: " << aux->altura
            << " - Balanceamento: " << fatorBalanceamento(aux) <<
        ImprimirPreOrdem(aux apontando a esquerda)
        ImprimirPreOrdem(aux apontando a direita)

```
Node *Arv::libera(Node *root)
{
    if(root != NULL)
    {
        libera(root->esquerda);
        libera(root->direita);
        delete root;
    }
    return NULL;
}
```

Explicação:

Node acessa Arv, busca a função libera(Node acessa root)
    Se,
    root for diferente de NULL
        libera(root aponta a esquerda);
        libera(root aponta a direita);
        então deleta root (raíz)

```
int main()
{
    setlocale(LC_ALL, "Portuguese"); //para a leitura acentuada das palavras
```

```
Arv avl;

int op;
int valor_inserir, valor_search, valor_excluir;

int Nodes_iniciais[] = {30, 40, 24, 58, 48, 26, 11, 13, 14, 20};
cout << "Valores iniciais da AVL:" << endl;
for(int i = 0; i < 10; i++)
{
	avl.root =
		avl.Inserir(avl.root, Nodes_iniciais[i]);
	cout << Nodes_iniciais[i] << ",";
}


while (op != 5)
{

	cout << "\n\nSeja bem-vindo ao Menu Árvore AVL" << endl;

	cout << "\n1 - Inserir valor na árvore"
		 << "\n2 - Imprimir Pré-Ordem"
		 << "\n3 - Remover nó"
		 << "\n4 - Pesquisar nó"
		 << "\n5 - Sair" << endl;

	cout << "\Opção: ";
	cin >> op;

	switch (op)
	{
	case 1:
		cout << "\nDigite valor que deseja inserir: ";
		cin >> valor_inserir;
		avl.root =
			avl.Inserir(avl.root, valor_inserir);
		cout << "Valor inserido" << endl;
		system("cls");
		break;
	case 2:
		avl.ImprimirPreOrdem(avl.root);
		system("pause");
		system("cls");
		break;
```

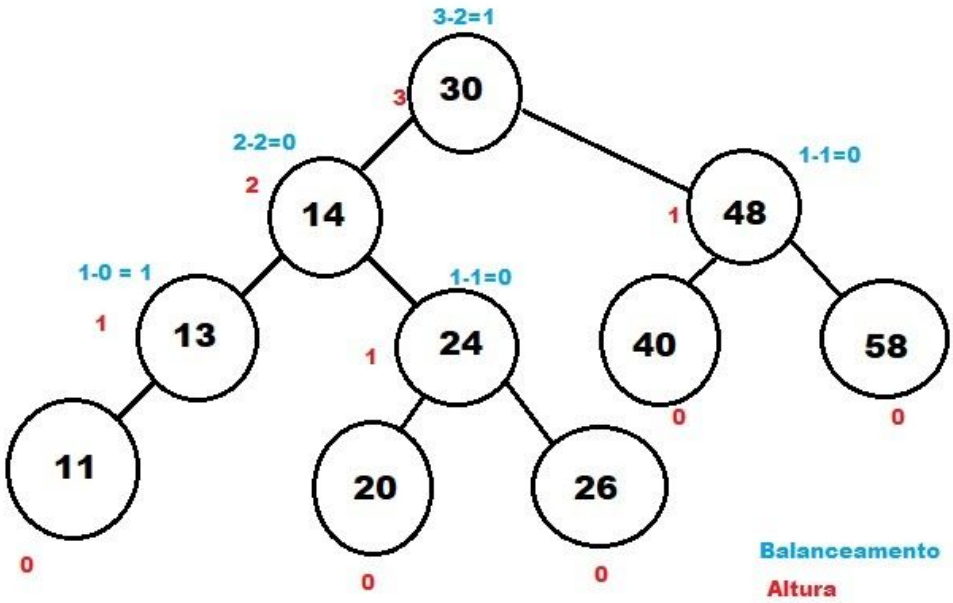
```
            case 3: // validar se o valor existe ou não
                        cout << "\nDigite valor que deseja remover: ";
                        cin >> valor_excluir;
                        if(avl.Pesquisar2(avl.root, valor_excluir)==true){
        avl.root =avl.Remover(avl.root, valor_excluir);
        cout << "Elemento removido" << endl;
                        }
                        else{
        cout<< "Elemento não encontrado"<<endl;
                        }
                        system("pause");
                        system("cls");
                        break;
            case 4:
                        cout << "\nDigite valor que deseja pesquisar: ";
                        cin >> valor_search;
                        if(avl.Pesquisar2(avl.root, valor_search)==true){
     cout<<"valor encontrado"<<endl;
                        }
                        else{
        cout<<"valor nao encontrado"<<endl;
                        }
        system("pause");
                        system("cls");
                        break;
            default:
                        break;
            }
        }
        avl.root = avl.libera(avl.root);

        return 0;
}
```

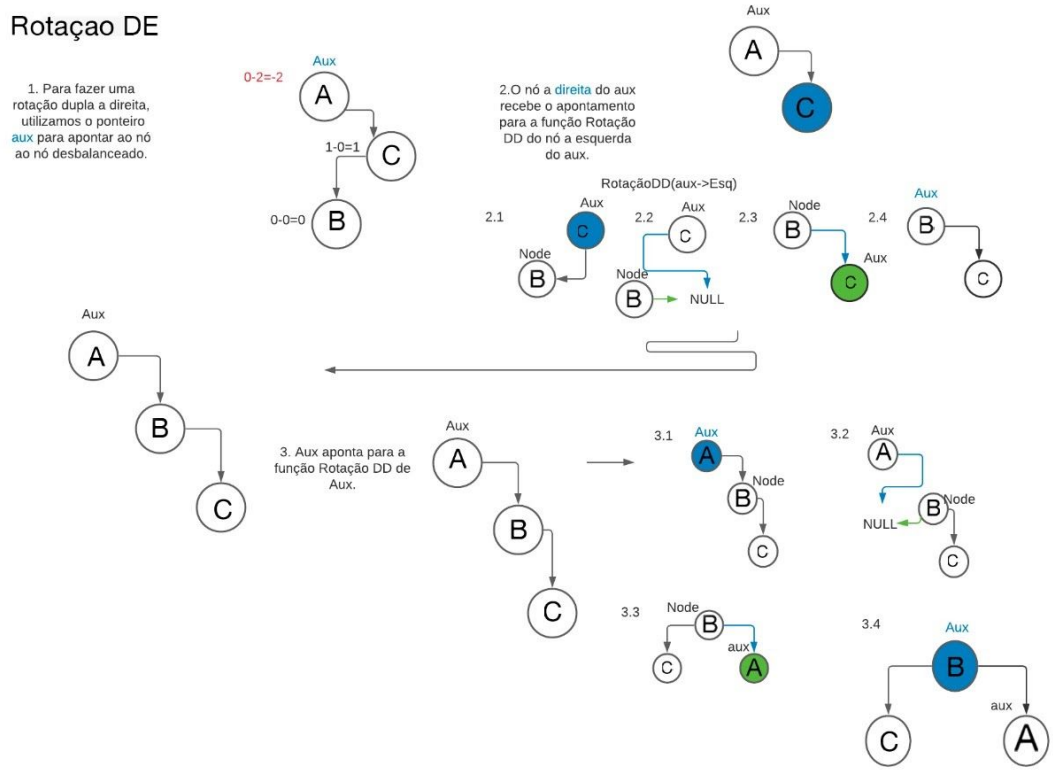
A função main serve como o ponto de partida para a execução do programa. Em geral, ela controla a execução direcionando as chamadas para outras funções no programa. Normalmente, um programa para de ser executado no final de main, embora possa terminar em outros pontos no programa por diversos motivos.

FIM.

Agora, para melhor entendimento, colocamos abaixo as rotações da árvore apresentada na codificação.

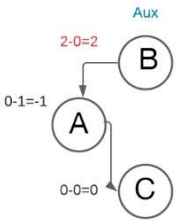

3-2=1
30
3
2-2=0
2
14
1
48
1-1=0
1-0 = 1
1
13
1-1=0
1
24
40
58
0
0
11
20
26
0
0
0
Balanceamento
Altura

Rotações:

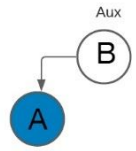

Rotaçao DE
1. Para fazer uma rotação dupla a direita, utilizamos o ponteiro aux para apontar ao nó ao nó desbalanceado.
0-2=-2
Aux
A
1-0=1
C
0-0=0
B
2.O nó a direita do aux recebe o apontamento para a função Rotação DD do nó a esquerda do aux.
Aux
A
C
RotaçãoDD(aux->Esq)
2.1
Aux
C
Node
B
2.2
Aux
C
Node
B
NULL
2.3
Node
B
Aux
C
2.4
Aux
B
C
Aux
A
B
C
3. Aux aponta para a função Rotação DD de Aux.
Aux
A
B
C
3.1
Aux
A
Node
B
C
3.2
Aux
A
Node
B
NULL
C
3.3
Node
B
aux
C
A
3.4
Aux
B
aux
C
A

Rotação ED

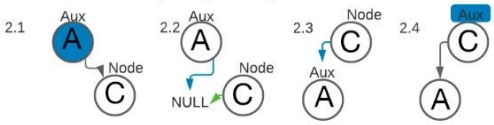
1. Para fazer uma rotação dupla a esquerda, utilizamos o ponteiro aux para apontar ao nó ao nó desbalanceado.

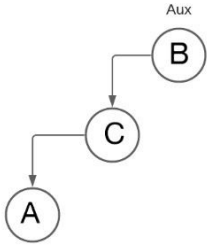


2.O nó a esquerda do aux recebe o apontamento para a função Rotação DD do nó a esquerda do aux.

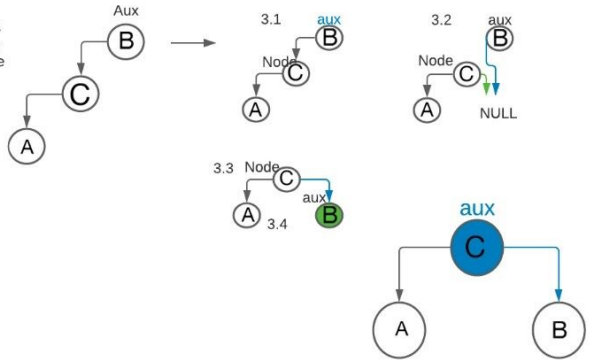


RotaçãoDD(aux->Esq)





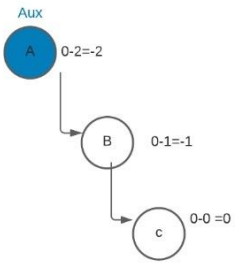
3. Aux aponta para a função Rotação EE de Aux.



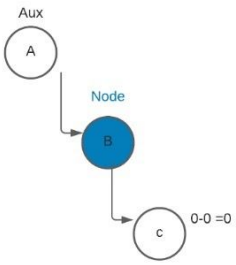
Rotação DD

O nó desbalanceado sofrerá uma rotação a esquerda.

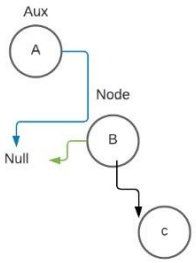
1.Para fazer a rotação utilizamos o pontiro aux para apontar ao nó desbalanceado.



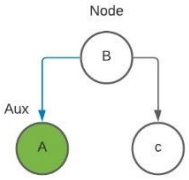
Criamos um ponteiro Node que apontara o no a direita do Aux.



3. A direita do no Aux aponta o no a esquerda do NODE

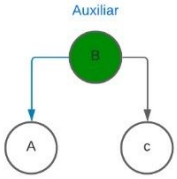


4.A direita de NODE aponta para Aux e a raiz aponta para Node.



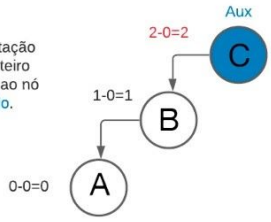
Digite alguma coisa

5.Auxiliar aponta para Node.

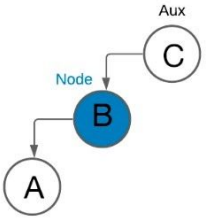


Rotação EE

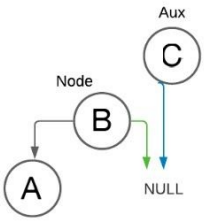
1.Para fazer a rotação utilizamos o ponteiro aux para apontar ao nó desbalanceado.



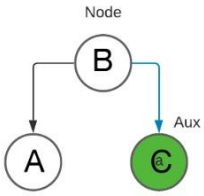
2.Criamos um ponteiro Node que apontara o no a esquerda do Aux.



3. A esquerda do no Aux aponta o no a direita do NODE.



4.A direita de NODE aponta para Aux e a raiz aponta para Node.



5.Auxiliar aponta para Node