UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

DOCUMENTAÇÃO ARVORE BINARIA COSTURADA

Pedro Ailan Silva de Oliveira
Vitor Hugo Ribeiro Tiburtino de Melo

São Cristóvão – SE
2020

UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

Pedro Ailan Silva de Oliveira
Vitor Hugo Ribeiro Tiburtino de Melo

DOCUMENTAÇÃO ARVORE BINÁRIA COSTURADA

Documentação relacionada a implementação da estrutura arvore binária costurada.

São Cristóvão – SE
2020

1. Introdução

Está arvore do inglês possui denominação Threared Binary Tree, que na tradução recebe o nome de Arvore Binária Encadeada. Também pode ser chamada de arvore binaria com fios. Ela é uma arvore derivada da Arvore Binária de Busca, porém os ponteiros dos nós folhas que antes eram nulos agora apontam para os predecessores e sucessores em ordem.

Tem como vantagem tornar a travessia em ordem mais rápida, no qual é feita sem a necessidade da utilização de pilhas ou recursão. Além do encadeamento ser útil para o acesso rápido de um nó ancestral. Por sua vez como desvantagem, na hora de realizar a travessia em preOrdem ou posOrdem é necessário a utilização de uma séria de tratamentos de If's para evitar que na hora da execução, ela entre em loop infinito.

Uma arvore binária costurada pode conter um encadeamento único: cujo apenas o ponteiro da direita é apontado para o sucessor em ordem (caso exista). Ou também pode possuir um encadeamento duplo: no qual os ponteiros NULL da esquerda e da direita são utilizados para apontar para o predecessor e o sucessor, respectivamente.

2. Descrição

Está árvore possui um comportamento e estrutura similar a arvore binária de busca, porém na estrutura desta arvore é utilizado dois flag do tipo booleano que informa false se o no aponta para um filho, e true para caso ele aponte para um predecessor(ponteiro da esquerda) ou para caso ele aponte para o sucessor(ponteiro da direita), e na hora de inserção e deleção, 3 cuidados são tomados para ambos.

Na inserção é necessário ajustar o encadeamento dos nós, após a inserção de cada elemento.

No primeiro caso, quando a inserção é numa arvore vazia, os ponteiros esquerdo e direito do nó são definidos como NULL e o novo nó se torna a raiz.

```
Raiz = novoNo;
novoNo->esq =NULL;
novoNo->dir = NULL;
```

No segundo caso, é quando um nó é inserido como filho esquerdo. Depois da inserção do nó deve ser feito com que os ponteiros esquerdo e direito do novo apontem para o antecessor e sucessor respectivamente.

```
novoNo->esq = pai->esq;
novoNo->dir = pai;
```

Após a inserção o ponteiro esquerdo do pai era um encadeamento, porém após a inserção ele deve apontar para o novoNo.

```
Pai->esqCostura = false;
```

```
Pai->esq = novoNo;
```

No terceiro caso, quando o nó é inserido como filho direito. O pai do novoNo é seu antecessor. O nó que antes era o sucessor em ordem do pai, agora é o sucessor em ordem do novoNo.

```
novoNo->esq = pai;

novoNo->dir = pai->dir;
```

Antes da inserção, o ponteiro direito do no pai apontava para seu sucessor (sendo um encadeamento) mas após a inserção ele aponta para o nó filho, o novoNo.

```
pai->dir        Costurada = false;

pai->dir = novoNo;
```

Para a deleção primeiro a chave a ser deletada é pesquisada, e em seguida é realizada a verificação de 3 casos:

Para o primeiro caso (Ou casa A no código), quando um nó folha é excluído. Quando um nó folha de uma arvore binária é excluído, os ponteiros esquerdo ou direito de seu pai é definido como NULL. Porém em uma arvore binária encadeada caso seja o filho esquerdo de seu pai, após a exclusão, o ponteiro esquerdo de seu pai deve apontar para seu predecessor

```
Pai->esqCosturada = true;

Pai->esq = noASerDeletado->esq;
```

Caso o nó a ser excluído for o filho da direita de seu pai, após a exclusão o ponteiro direito de seu pai, após a exclusão o ponteiro direito do pai deve apontar para o sucessor. O sucessor do nó que foi excluído passa a ser o sucessor do nó pai após a exclusão.

```
Pai->dirCosturada = true;

Pai->dir = noAserDeletado->dir;
```

No segundo caso (ou caso B no código), o nó a ser excluído possui apenas um filho. Após a exclusão o sucessor e o predecessor são descobertos.

```
Sucessor = sucessorEmOrdem(raiz);

Predecessor = predecessorEmOrdem(raiz);
```

Caso o nó a ser excluído possua uma subArvore a esquerda, após a exclusão o encadeamento direito do predecessor deverá apontar para o sucessor.

```
sucessor->dir = atual->dir;

predecessor->dir = sucessor;
```

Caso o nó a ser excluído possua uma subArvore a direita, após a exclusão, o encadeamento esquerdo do seu sucessor deve ser apontado para o seu predecessor.

```
Sucessor->esquerdo = predecessor;
```

Por fim o terceiro caso (ou caso C no código) o nó a ser excluído possua dois filhos, é buscado quem é o nó predecessor em ordem a ele, ou seja, encontrado o nó mais a direita do ponteiro a esquerda do nó a ser deletado. Depois a chave é

copiada deste no sucessor para o no a ser deletado, e chama o casoA ou casoB para deletar o nó encontrado. No caso quem é deletado é o nó que é deletado é o anterior a ele (em ordem), no qual sua chave é copiada.

3. Aplicação

Pode ser utilizada para substituir um arvore binária de busca e ganhar eficiência na travessia em ordem. Por possuir ponteiro folhas que apontem para seus predecessores e sucessores.

4. Referências Bibliográficas

- https://www.geeksforgeeks.org/threaded-binary-tree/

5. Implementação em C

5.1 RegistroArvoreCosturada.h

```
#ifndef REGISTROS_H_
#define REGISTROS_H_
#define true 1
#define false 0

typedef int TIPOCHAVE;
typedef int boolean;

typedef struct reg {
    TIPOCHAVE chave;
    struct reg *esq;
    struct reg *dir;
    boolean dirCostura, esqCostura;
} noh;

//=======================================Chamada das Funcoes=======================================
typedef noh* pont;
pont inicializarArvore();
```

```
boolean arvoreVazia(pont raiz);
pont criaNovoNoh(TIPOCHAVE ch);
//void buscar(pont raiz, TIPOCHAVE chave);
pont buscaNohParaInserir(pont raiz, TIPOCHAVE ch);
pont buscaNoh(pont raiz, TIPOCHAVE ch, pont *pai);
pont inserir(pont raiz, pont novoNoh);
pont removeNoh(pont raiz, TIPOCHAVE ch);
pont emOrdemSucessor(pont p);
pont ordemPredecessor(pont r);
pont nohMaisAEsquerda(pont raiz);
pont ordemSucessor(pont s);
void emOrdem(pont raiz, pont n);
void imprimePreOrdem(pont raiz);
void imprimePosOrdem(pont raiz);
pont casoA(pont raiz, pont pai, pont atual);
pont casoB(pont raiz, pont pai, pont atual);
pont casoC(pont raiz, pont pai, pont atual);
pont buscaNohParaInserir(pont raiz, TIPOCHAVE ch);

#endif
```

5.2 funcoes.h

```
#ifndef FUNCOES_H_
#define FUNCOES_H_
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "registroArvoreCosturada.h"
#define true 1
#define false 0

//==============================Funcao de inicializar e  testa
vazio==============================
/* Função que inicializa a arvore como NULL */
pont inicializarArvore(){
```

```
    return NULL;
}

/* Função que testa se a arvore passada é vazia, returna true caso for */
boolean arvoreVazia(pont raiz){
    if(raiz == NULL) return true;
    else return false;
}
//=============================================================================


//==================================Funcao que cria um  Novo Noh==================================
/* Retorna um novo noh, no qual os ponteiros do noh a esquerda e direita inicialmente apontam para o NULL,
e seta os verificadores da esquerda e direita como True. */

pont criaNovoNoh(TIPOCHAVE ch){
   pont novoNoh = (pont)malloc(sizeof(noh));
   novoNoh->esq = NULL;
   novoNoh->dir = NULL;
   novoNoh->chave = ch;
   novoNoh->dirCostura = true;
   novoNoh->esqCostura = true;
   return novoNoh;
}
//=============================================================================


//=======================================Funcoes de Busca=======================================

/*  void buscar(pont raiz, TIPOCHAVE chave) {
   pont atual = raiz;
   printf("test");
```

```
    //verifica se a raiz e null
    //if(raiz == NULL) printf("Arvore vazia");
    //verifica se o valor da raiz e igual ao valor passado, se for retorna a raiz
    if(atual->chave == chave) printf("Chave %d encontrada\n", atual->chave);
    //verifica se o valor da chave da raiz e maior que o passado, se for faz a
recursividade e chama a subarvore esquerda
    else if(atual->esqCostura == false && atual->chave > chave) buscar(atual-
>esq, chave);
    else if(atual->dirCostura == false && atual->chave < chave) buscar(atual-
>dir, chave);
    else printf("Chave nao existe na arvore\n");
}  */


void buscar(pont raiz, TIPOCHAVE ch) {
    pont atual = raiz;


    while(raiz != NULL){
        //primeiro teste para evitar q haja chaves iguais na arvore
        if(raiz->chave == ch) {
            printf("Chave %d Encontrada\n", raiz->chave);


        }


        //atualiza o ponteiro "pai"


        //realiza a primeira comparacao de chaves
        //caso a chave inserida seja menor que a da raiz, ele entra nesse if
        //para poder andar pela subarvore da esquerda
        if(ch < raiz->chave){


            if(raiz->esqCostura == false) raiz = raiz->esq;
            else break;
        }else {
            if(raiz->dirCostura == false) raiz = raiz->dir;
            else break;
        }
        }
```

```
}

//Nesta funcao e buscado onde sera inserido o novoNoh
//Ele deixa salvo o pai do no que será inserido
pont buscaNohParaInserir(pont raiz, TIPOCHAVE ch){
    pont pai = NULL;

    //while para percorrer a arvore
    while(raiz != NULL){
        //primeiro teste para evitar q haja chaves iguais na arvore
        if(raiz->chave == ch) {
            printf("Chave Duplicada nao pode ser inserida");
            //retorna NULL para ser tratado na funcao que o chamou
            return NULL;
        }

        //atualiza o ponteiro "pai"
        pai = raiz;

        //realiza a primeira comparacao de chaves
        //caso a chave inserida seja menor que a da raiz, ele entra nesse if
        //para poder andar pela subarvore da esquerda
        if(ch < raiz->chave){

            if(raiz->esqCostura == false) raiz = raiz->esq;
            else break;
        }else {
            if(raiz->dirCostura == false) raiz = raiz->dir;
            else break;
        }
    }

    return pai;
}
```

```
//=========================================================
=======================================

//=====================================Funcao Inserir e
Deletar=====================================
//Funcao para inserir, que respeita os 3 casos de inserção,
// pois cada encadeamento tem que ser ajustado após a inserção
// Cujo primeiro Caso é para a inserção seja numa arvore vázia
// Sendo o Segundo quando o noh será inserido no filho esquerdo
// E o terceiro quando o noh é inserido como filho esquerdo.

pont inserir(pont raiz, pont novoNoh){
    //se a raiz for null, inserimos la
    if(arvoreVazia(raiz)) return novoNoh;
    pont no;
    no = buscaNohParaInserir(raiz, novoNoh->chave);
    if(no == NULL) return NULL;
    //se a chave do elemento a ser inserido for menor que a da raiz, insere na
subarvore a esquerda
    else if(novoNoh->chave < no->chave){
        novoNoh->esq = no->esq;
        novoNoh->dir = no;
        no->esqCostura = false;
        no->esq = novoNoh;
        //se a chave do elemento a ser inserido for maio que a da raiz, insere na
subarvore a direita.
    } else {
        novoNoh->esq = no;
        novoNoh->dir = no->dir;
        no->dirCostura = false;
        no->dir = novoNoh;
    }

    return raiz;
}
// Função para remoção, no qual a chave a ser excluida é pesquisada,
// em seguida é efetuada a remoção, respeitando os 3 casos de remoção:
```

```
// No primeiro caso é quando o noh de uma folha precisa ser Excluído.
// Em seu segundo caso o noh a ser excluído possui apenas um filho
// E no ultimo caso o noh a ser escolhido possui dois filhos.
pont removeNoh(pont raiz, TIPOCHAVE ch) {

    //Inicializa o pai como NULL, e atual recebe a rauz
    pont pai = NULL, atual = raiz;

    int achou = 0;

    //busca o noh que contenha a chave que sera deletado
    while (atual != NULL) {
        //caso encontre quebra o laco e retorna true
        if (ch == atual->chave) {
            achou = 1;
            break;
        }
        //seta o valor do pai como o atual para n perder a referencia
        pai = atual;

        //verifica se a chave e menor que o qua chave do noh atual,
        if (ch < atual->chave) {
            //caso seja verifica se o ponteiro esquerdo e um encademanto(se e um
noh folha)
            //se n for o ponteiro atual recebe o esquerdo dele,
            if (atual->esqCostura == false) atual = atual->esq;
            //caso nao seja ele quebra o enlace
            else break;
        //caso para quando o noh e maior que a chave do atual
        }else {
            //caso seja verifica se o ponteiro direito e um encademanto(se e um noh
folha)
            if (atual->dirCostura == false) atual = atual->dir;
            //caso nao seja ele quebra o enlace
            else break;
        }
```

```
    }

    //verifica se o noh esta contido na arvore
    if (achou == 0) printf ("Esta chave nao possui na arvore\n");
    //caso para quando o noh a ser excluido possui dois filhos
    else if (atual->esqCostura == false && atual->dirCostura == false) raiz =
casoC(raiz, pai, atual);
    //caso para quando o noh a ser excluido possui um filho a esquerda
    else if (atual->esqCostura == false) raiz = casoB(raiz, pai, atual);
    //caso para quando o noh a ser excluido possui um filho a direita
    else if (atual->dirCostura == false) raiz = casoB(raiz, pai, atual);
    //caso para quando o noh a ser excluido nao possui nenhum filho
    else raiz = casoA(raiz, pai, atual);

    return raiz;

}

//=============================================================
=======================================

//===================================Funcoe de Busca para um
noh=================================

//Utilizada em na função que printa em Ordem.
pont emOrdemSucessor(pont p){
    if(p->dirCostura == true) return p->dir;
    p = p->dir;
    while(p->esqCostura == false) p = p->esq;

    return p;
}

pont posOrdemSucessor(pont p){
    if(p->dirCostura == true) return p->dir->dir;
    p = p->dir;
    while(p->esqCostura == false) p = p->esq;
```

```
    return p;
}
//Função para encontrar o noh mais a esquerda,
pont nohMaisAEsquerda(pont raiz){
    pont p = raiz;
    while(p->esqCostura == false) p = p->esq;
    return p;

}

//Funções Utilizadas pelo métodod de remoção
//Encontra o Predecessor em ordem e retorna o este noh
pont ordemPredecessor(pont r){
    if(r->esqCostura == true) return r->dir;
    if(r->esq == NULL) return NULL;
    r = r->esq;
    //CONFERIR
    while (r->dirCostura == false && r->dir != NULL) r = r->dir;
    return r;
}

//Encontra o Sucessor em ordem e retorna o este noh
pont ordemSucessor(pont s) {
    if(s->dirCostura == true) return s->dir;

    if(s->dir == NULL) return NULL;
    s = s->dir;
    while(s->esqCostura == false) s = s->esq;

    return s;
}

//=================================================================
=======================================
```

```
//=======================================Funcoes de
Impressao=======================================

//imprime esquerda, raiz, direita
void printEmOrdem(pont raiz, pont n){
    if(arvoreVazia(raiz)) printf("Arvore vazia");

    //while(p->esqCostura == false) p = p->esq;
    if(!arvoreVazia(n)){
        printf("%d ", n->chave);
        printEmOrdem(raiz, emOrdemSucessor(n));
    }

    printf("\n");
}

//imprime raiz, esquerda, direita
void printPreordem(pont r)
{
    pont atual = r;

    while(atual != NULL)
    {
        printf("<");
        printf("%d", atual->chave);

        if(atual->esq != NULL && atual->esqCostura == false ) atual = atual->esq;

        else if(atual->dirCostura == 0 ) atual = atual->dir;

        else
        {
            while(atual->dir != NULL && atual->dirCostura == true) atual = atual-
>dir;

            if(atual->dir == NULL) {
```

```
            printf(">");
            break;
        }
        else atual = atual->dir;

    }
    printf(">");
  }

  printf("\n");
}


//================================================Casos de Exclusao================================================
pont casoA(pont raiz, pont pai, pont atual){
  //caso o noh a ser deletado seja a raiz;
  if (pai == NULL)  raiz = NULL;
  //caso o noh a ser deletado esteja ao lado esquerdo de seu pai
  else if (atual == pai->esq) {
    pai->esqCostura = true;
    pai->esq = atual->esq;
  //caso o noh a ser deletado esteja ao lado direito de seu pai
  }else {
    pai->dirCostura = true;
    pai->dir = atual->dir;
  }

  //da um free na memoria no ponteiro atual e retorna a raiz
  free(atual);
  return raiz;
}

pont casoB(pont raiz, pont pai, pont atual){
  pont filho;
```

```
    /* //inicializa o noh filho com o noh a esquerda do noh a ser deletado
    if(atual->esqCostura == false) filho = atual->esq;
    //inicializa o noh filho com o noh a direita do noh a ser deletado
    else filho = atual->dir;

    //caso o noh a ser deletado seja a raiz
    if(pai == NULL) raiz = filho;
    else if(atual == pai->esq) pai->esq = filho;
    else pai->dir = filho; */

    //encontra o sucessor e o predecessor
    pont sucessor = ordemSucessor(atual);
    pont predecessor = ordemPredecessor(atual);

    //caso o noh atual possua a subArvore esquerda
    if(atual->esqCostura == false) {
        if(sucessor == NULL) {
            pai->dir = atual->esq;
            sucessor = atual->esq;

            while(sucessor->dirCostura == false) sucessor = sucessor->dir;
            sucessor->dir = NULL;
        }else {
            sucessor->dir = atual->dir;
            predecessor->dir = sucessor;
        }
    }

    //caso o noh atual possua a subArvore direita
    else if(atual->dirCostura == false) {
        if(predecessor == NULL) {
            pai->esq = atual->dir;
            predecessor = atual->dir;
            while(predecessor->esqCostura == false) predecessor = predecessor-
>esq;
```

```
            predecessor->esq = NULL;
        }
        pai->esq = atual->dir;
        sucessor->esq = atual->esq;
    }


    free(atual);

    return raiz;

}

pont casoC(pont raiz, pont pai, pont atual) {

    //Encontra o sucessor em ordem e seu pai
    pont paiSucessor = atual;
    pont sucessor = atual->esq;

    //encontra o filho mais a esquesda do sucessor
    while(sucessor->dirCostura == false && sucessor->dir != NULL) {
        paiSucessor = sucessor;
        sucessor = sucessor->dir;
    }

    //faz com que a chave a atual recebe a chave do mais a esquerda
    atual->chave = sucessor->chave;

    //verifica se ha a necessidade
    if(sucessor->esqCostura == true && sucessor->dirCostura == true) raiz =
casoA(raiz, paiSucessor, sucessor);
    else raiz = casoB(raiz, paiSucessor, sucessor);

    return raiz;
}
#endif
```

5.3 main.c

```
#include <stdlib.h>
#include <stdio.h>
#include "funcoes.h"
#define true 1
#define false 0

typedef int TIPOCHAVE;

void main(){

   boolean verificador = true;
   int opcao;
   TIPOCHAVE chave;
   pont r = inicializarArvore();
   pont noh = NULL;
   while (verificador){
      //system("clear || cls");
      printf("================================ Bem Vindo ao menu Da Arvore Costurada Binaria de Busca Duplamente Encadeada ================================\n");
      printf("1- Inserir na Arvore\n");
      printf("2- Buscar na Arvore\n");
      printf("3- Remover da Arvore\n");
      printf("4- Visualizar Arvore em PreOrdem (Raiz, Esquesda, Direita)\n");
      printf("5- Visualizar Arvore em Ordem(Esquesda, Direita, Raiz)\n");
      printf("6- Sair do Menu\n");

      printf("Digite a opcao desejada: ");
      scanf("%d",&opcao);

      switch(opcao)
         {
            case 1:
               printf("Digite o valor da chave a ser inserida: ");
```

```
                scanf("%d", &chave);
                noh = criaNovoNoh(chave);
                r = inserir(r, noh);
                break;

            case 2:
                printf("Digite o valor da chave a ser buscada: ");
                scanf("%d", &chave);
                buscar(r, chave);
                break;
            case 3:
                printf("Digite o valor da chave a ser removida: ");
                scanf("%d", &chave);
                removeNoh(r, chave);
                break;
            case 4:
                printf("Arvore Em PreOrdem: ");
                printPreordem(r);
                break;
            case 5:
                printf("Arvore Em Ordem: ");
                 pont n = nohMaisAEsquerda(r);
                printEmOrdem(r, n);
                break;
            case 6:
                printf("Voce saiu da arvore, reinicia o projeto para ter acesso ao
Menu de novo");
                verificador = false;
        }
    }


}
```