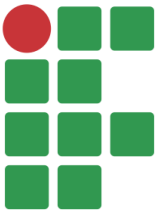

INSTITUTO FEDERAL DA PARAÍBA
CAMPUS CAMPINA GRANDE
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO
DISCIPLINA DE ESTRUTURAS DE DADOS
PROF. VICTOR ANDRÉ PINHO DE OLIVEIRA

Estruturas de Dados

Aula 9 - ED - Árvores Binárias

Introdução

Olá,

Na semana anterior, nós implementamos a Estrutura de Dados Fila Sequencial Circular. O grande diferencial dessa Estrutura, se comparada com a Fila Sequencial, está na otimização do uso do array, fazendo-o parecer circular, o que torna o custo computacional da operação dequeue mais baixo.

Com a implementação da Fila Sequencial Circular nós encerramos as Estruturas Lineares. A partir da aula de hoje, nós veremos a implementação de Estruturas não-Lineares. Veremos duas dessas Estruturas: Árvores Binárias e Matrizes Esparsas. Nesta aula, dedicaremos nossa atenção às **Árvores Binárias**.

Certifique-se de estar com o estudo de recursividade em dia!

Árvores Binárias

Como já mencionado, as **Árvores Binárias** fazem parte do conjunto de Estruturas categorizadas como não-Lineares. Se você recordar, mencionamos que uma Estrutura é dita Linear quando os elementos estão dispostos em uma sequência, isto é, cada elemento tem apenas um antecessor e um sucessor. Por outro lado, Estruturas ditas não-Lineares não seguem esse padrão, não existindo a ideia de antecessor e sucessor.

Uma **Árvore** é um tipo de Estrutura onde cada elemento é denominado **nó**, sendo que o primeiro nó da Árvore é chamado de nó **raiz**. Cada nó poderá ramificar (apontar) para vários outros nós. Uma ramificação também é chamada de subárvore, pois é também vista como uma árvore. Se um nó ramifica, dizemos que o nó tem filhos. Se um nó não tem filhos, este é chamado de nó **folha**. Uma **Árvore é dita Binária** quando cada nó tem um máximo de dois nós filhos.

Observe o diagrama a seguir.

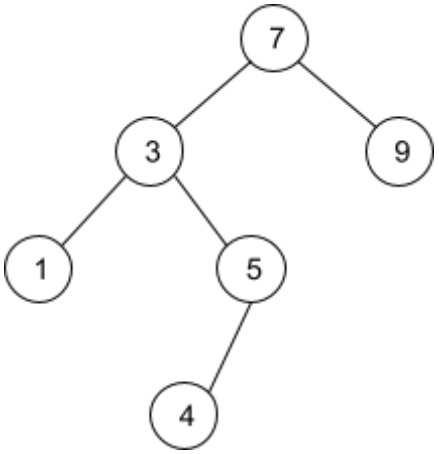

Diagrama 1 - Exemplo de uma Árvore Binária

O Diagrama acima ilustra um exemplo de Árvore Binária. Se você for um leitor atento, deve ter notado que cada nó da Árvore acima obedece o seguinte padrão:

- os filhos da subárvore esquerda são todos menores que ele
- os filhos da subárvore direita são todos maiores que ele

Árvores Binárias que seguem o padrão supracitado são chamadas de **Árvores Binárias Ordenadas** ou **Árvores Binárias de Busca**. É exatamente essa especialização de Árvores Binárias que iremos implementar nesta aula.

Implementação

Existem várias formas de implementar Árvores Binárias. Para esta aula, foi escolhido utilizar uma implementação recursiva. Por favor, reveja o assunto de recursividade que foi ministrado algumas aulas atrás.

Código

Segue abaixo o código.

```c
#include <stdio.h>
#include <stdlib.h>

struct sNODE{
  int dado;
  struct sNODE *esq, *dir;
};

struct sNODE *raiz = NULL;

struct sNODE *inserir(struct sNODE *no, int dado);
struct sNODE *remover(struct sNODE *no, int dado);

void emOrdem(struct sNODE *no);
void preOrdem(struct sNODE *no);
void posOrdem(struct sNODE *no);

struct sNODE *buscar(struct sNODE *no, int dado);
int obter(struct sNODE *no);

struct sNODE *apagar(struct sNODE *no);


int main(){
  return 0;
}

struct sNODE *inserir(struct sNODE *no, int dado){
  if (!no) {
      no = (struct sNODE*) malloc(sizeof(struct sNODE));
      no->dado = dado;
      no->esq = no->dir = NULL;
  } else if (dado < no->dado)
      no->esq = inserir(no->esq,dado);
  else
      no->dir = inserir(no->dir,dado);

  return no;
}

struct sNODE *remover(struct sNODE *no, int dado){
```

```
  struct sNODE *aux = NULL, *aux2 = NULL;

  if (no) {
     if (no->dado == dado) {
        if (no->esq == no->dir) {
           free(no);
           return NULL;
        }
        else if (!no->esq) {
           aux = no->dir;
           free(no);
           return aux;
        }
        else if (!no->dir) {
           aux = no->esq;
           free(no);
           return aux;
        } else {
           aux = aux2 = no->dir;
           while (aux->esq)
              aux = aux->esq;
           aux->esq = no->esq;
           free(no);
           return aux2;
        }
     }
     else {
        if (dado < no->dado)
           no->esq = remover(no->esq, dado);
        else
           no->dir = remover(no->dir, dado);
     }
  }
  return no;
}

void emOrdem(struct sNODE *no){
  if (no) {
     emOrdem(no->esq);
     printf("%d ",no->dado);
     emOrdem(no->dir);
```

```
  }
}

void preOrdem(struct sNODE *no){
  if (no) {
      printf("%d ",no->dado);
      preOrdem(no->esq);
      preOrdem(no->dir);
  }
}

void posOrdem(struct sNODE *no){
  if (no) {
      posOrdem(no->esq);
      posOrdem(no->dir);
      printf("%d ",no->dado);
  }
}

struct sNODE *buscar( struct sNODE *no, int dado){
  if (no) {
      if (no->dado == dado)
         return no;
      else if (dado < no->dado)
         return buscar(no->esq, dado);
      else
         return buscar(no->dir, dado);
  }

  return NULL;
}

int obter(struct sNODE *no) {
  if (no)
      return no->dado;
  else {
      printf("Nenhum dado para retornar.");
      exit(0);
  }
}
```

```
struct sNODE *apagar(struct sNODE *no){
  if (no) {
      no->esq = apagar(no->esq);
      no->dir = apagar(no->dir);
      free(no);
  }
  return NULL;
}
```

Para implementar uma Árvore Binária, vamos precisar de:

- um registro (sNODE) para armazenar as informações de um nó:
  - campo **dado**: vai armazenar o elemento do nó
  - campo **esq**: vai armazenar o endereço do nó filho esquerdo
  - campo **dir**: vai armazenar o endereço do nó filho direito.
- duas funções base:
  - **inserir**: para inserir um elemento na Árvore. Note que, em Árvores Binárias de Busca (ou Ordenadas), a inserção de elementos repetidos é desencorajada
  - **remover:** para remover algum elemento da Árvore, caso exista.
- algumas funções de consulta:
  - **emOdem**: na consulta em Ordem, primeiro os nós da subárvore esquerda de um nó são visitados, em seguida é visitado o nó e depois os nós da subárvore direita
  - **preOrdem**: na consulta em pré-Ordem, primeiro é visitado o nó, depois os nós da subárvore esquerda, seguido dos nós da subárvore direita
  - **posOrdem**: na consulta em pós-Ordem, primeiro os nós da subárvore esquerda são visitados, depois os nós da subárvore direita e, por fim, o nó é visitado.
- algumas funções auxiliares:
  - **buscar**: procura por um elemento na Árvore
  - **obter**: retorna o dado presente no nó
  - **apagar**: apaga todos os nós da Árvore.

Precisaremos também de um ponteiro para o nó raiz. Ele pode ficar de forma global ou local, na função main. Optamos por deixar como variável global.

Vamos entender como cada função funciona.

Funções base

Começaremos pelas funções base da Estrutura.

Função inserir

A função inserir tem o objetivo de inserir um elemento na Árvore.

```
struct sNODE *inserir(struct sNODE *no, int dado){
  if (!no) {
      no = (struct sNODE*) malloc(sizeof(struct sNODE));
      no->dado = dado;
      no->esq = no->dir = NULL;
  } else if (dado < no->dado)
      no->esq = inserir(no->esq,dado);
  else
      no->dir = inserir(no->dir,dado);

  return no;
}
```

A função inserir, bem como as demais funções que modificam a Árvore, retorna um ponteiro para sNODE e recebe um ponteiro para sNODE. Optamos por fazer dessa forma em razão da implementação recursiva.

A função inserir, além de um ponteiro para sNODE, recebe o dado a ser inserido.

De maneira geral, a lógica da função é a seguinte: a partir do nó raiz, verifico se o dado que quero inserir é menor que o dado presente no nó em questão. Se sim, ele será inserido em algum ponto da subárvore esquerda; se for maior, será inserido em algum ponto da subárvore direita. Na medida em que as visitas aos nós vão se aprofundando, vai ocorrer de chegarmos a um "nó NULL" - o ponto exato em que o novo nó deverá ser inserido. Quando chegar nessa situação - a condição de parada - alocamos espaço para o novo nó, preenchemos o registro e, por fim, retornamos o ponteiro do nó recém-alocado.

Função remover

A função remover tem o objetivo de remover um elemento da Árvore.

```
struct sNODE *remover(struct sNODE *no, int dado){
  struct sNODE *aux = NULL, *aux2 = NULL;

  if (no) {
      if (no->dado == dado) {
         if (no->esq == no->dir) {
            free(no);
            return NULL;
         }
         else if (!no->esq) {
            aux = no->dir;
            free(no);
            return aux;
```

```
        }
        else if (!no->dir) {
           aux = no->esq;
           free(no);
           return aux;
        } else {
           aux = aux2 = no->dir;
           while (aux->esq)
              aux = aux->esq;
           aux->esq = no->esq;
           free(no);
           return aux2;
        }
     }
     else {
        if (dado < no->dado)
           no->esq = remover(no->esq, dado);
        else
           no->dir = remover(no->dir, dado);
     }
  }
  return no;
}
```

Além de retornar e receber um ponteiro sNODE, a função remover recebe o dado a ser removido da Árvore.

A lógica da função remover é a seguinte: a partir do nó raiz, verificamos se o dado sendo buscado é igual ao nó sendo visitado. Se não for igual, verificarmos se o dado é menor que o dado do nó - caso em que o elemento que queremos remover se encontra na subárvore esquerda - ou se o dado é maior que o dado do nó - caso em que o elemento que queremos remover se encontra na subárvore direita. Contudo, caso o nó sendo visitado seja igual ao dado que queremos remover, podemos cair em uma das quatro situações:

1. O nó a ser removido é um nó folha (seus filhos são iguais e iguais a NULL): nesse caso, basta removermos o nó e retornar NULL;
2. O nó a ser removido não tem o filho esquerdo: nesse caso, removemos o nó e retornamos no->dir para assumir o lugar do pai
3. O nó a ser removido não tem o filho direito: nesse caso, removemos o nó e retornamos no->esq para assumir o lugar do pai
4. O nó a ser removido tem filho esquerdo e direito: nesse caso, atribuímos a subárvore esquerda (no->esq) como filho do nó esquerdo mais profundo da subárvore direita antes de remover o nó.

Funções de consulta

E agora, veremos as funções de consulta.

Função emOrdem

Visita os nós da Árvore em Ordem.

```
void emOrdem(struct sNODE *no){
  if (no) {
      emOrdem(no->esq);
      printf("%d ",no->dado);
      emOrdem(no->dir);
  }
}
```

A partir do nó raiz, uma visita “em Ordem” visita primeiro a subárvore esquerda, depois visita o nó em questão, e só depois visita a subárvore direita.

Uma visita em Ordem na Árvore do Diagrama 1 resultaria em: 1 3 4 5 7 9.

Função preOrdem

Visita os nós da Árvore em pré-Ordem.

```
void preOrdem(struct sNODE *no){
  if (no) {
      printf("%d ",no->dado);
      preOrdem(no->esq);
      preOrdem(no->dir);
  }
}
```

A partir do nó raiz, uma visita “em pré-Ordem” visita primeiro o nó em questão, depois a subárvore esquerda, seguido da subárvore direita.

Uma visita em pré-Ordem na Árvore do Diagrama 1 resultaria em: 7 3 1 5 4 9.

Função posOrdem

Visita os nós da Árvore em pós-Ordem.

```
void posOrdem(struct sNODE *no){
  if (no) {
      posOrdem(no->esq);
      posOrdem(no->dir);
      printf("%d ",no->dado);
  }
}
```

A partir do nó raiz, uma visita “em pós-Ordem” visita primeiro a subárvore esquerda, seguido da subárvore direita, e só depois visita o nó em questão.

Uma visita em pós-Ordem na Árvore do Diagrama 1 resultaria em: 1 4 5 3 9 7.

Funções de auxiliares

E agora, veremos as funções auxiliares.

Função buscar

Procura por um dado na Árvore.

```
struct sNODE *buscar(struct sNODE *no, int dado){
  if (no) {
      if (no->dado == dado)
         return no;
      else if (dado < no->dado)
         return buscar(no->esq, dado);
      else
         return buscar(no->dir, dado);
  }

  return NULL;
}
```

A lógica da função buscar é a seguinte: se o nó que estou visitando contém o dado que estou buscando, então retorno o endereço do nó. Se não, o dado buscado for menor que o dado do nó, então vou buscar na subárvore esquerda, se não, vou buscar na subárvore direita. Se durante as visitas chegarmos a um nó folha e ainda assim não encontrarmos o dado, então o dado não está na Árvore e retornamos NULL.

Função obter

Obtém o dado de um nó da Árvore.

```
int obter(struct sNODE *no) {
  if (no)
      return no->dado;
  else {
      printf("Nenhum dado para retornar.");
      exit(0);
  }
}
```

Retorna o dado a partir de um nó da Árvore.

Função apagar

Apaga todos os nós da Árvore.

```
struct sNODE *apagar(struct sNODE *no){
  if (no) {
      no->esq = apagar(no->esq);
      no->dir = apagar(no->dir);
      free(no);
  }
  return NULL;
}
```

A lógica da função apagar é semelhante à lógica da visita em pós-Ordem, pois a ideia é apagar toda a subárvore esquerda, depois toda a subárvore direita, para depois apagar o nó em questão.

Na aula de hoje demos início ao estudo das Estruturas não-Lineares. Implementamos uma Árvore Binária Ordenada.

Na próxima aula veremos Matrizes Esparsas.

🤗

Bons estudos e até a próxima!