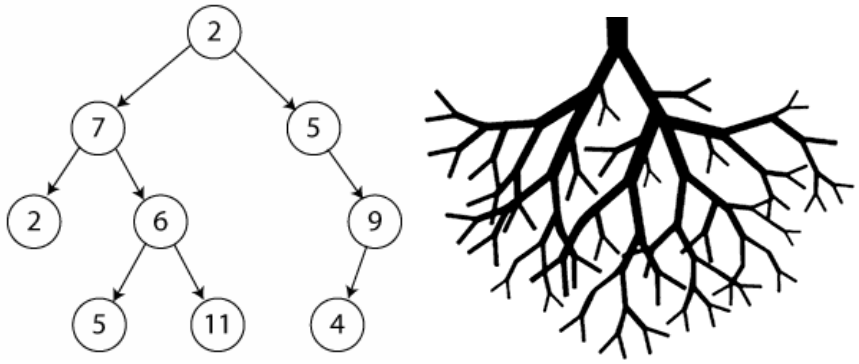
Árvores Binárias em C

Material de suporte para a vídeo aula da estrutura de dados árvore

1_ O que são Árvores Binárias?

- Árvores binárias são estruturas de dados que consistem em nós, onde cada nó tem, no máximo, dois filhos: um filho à esquerda e um filho à direita. Cada nó em uma árvore binária contém um valor e referências aos seus filhos. A estrutura é chamada "binária" porque cada nó tem, no máximo, dois filhos. As árvores binárias, quando balanceadas, tem uma complexidade de busca igual a O(log N), o que as torna ideais em situações onde é necessário retornar elementos de forma rápida em operações de busca.



2_ Estrutura da Árvore Binária:

- 2.1 NODO:

```
typedef struct Nodo {
   int info;
   struct Nodo* esq;
   struct Nodo* dir;
} Nodo;
```

Esta é a estrutura básica de uma árvore binária, nela se encontra um campo para armazenar o valor (info), sendo que tal campo pode ser de qualquer tipo e dois ponteiros do tipo Nodo, um apontando para o nó à sua esquerda e outro nó apontado à sua direita.

- 2.2 MÉTODO CRIAR RAIZ:

```
void criaRaiz(tree *t, elemento item) {
  ptrNodo no;
  no = (ptrNodo)malloc(sizeof(struct nodo));
  no->esq = NULL;
  no->dir = NULL;
  no->info = item;
  *t = no;
}
```

O método criaRaiz é responsável por instanciar a árvore binária, é passado por referência a árvore (tree *t) e o elemento que estará armazenado dentro da raiz da árvore. Dentro desse método, é criado um ponteiro (ptrNodo no), depois é feita uma alocação dinâmica de memória para o ponteiro. A seguir, é atribuído o valor null para o campo esq e dir, que são os ponteiros para o filho à esquerda e o filho à direita, o item, que é o valor que será armazenado no nó e, por fim, a árvore t passa a apontar para o nó criado.

- 2.3 MÉTODO DE INSERIR :

```
// Função para inserir um elemento na árvore
Tree insere(Tree t, int item) {
   if (t == NULL) {
      // Árvore vazia ou local de inserção encontrado
      Nodo* novoNodo = (Nodo*)malloc(sizeof(Nodo));
      if (novoNodo == NULL) {
         // Tratamento de erro: não foi possível alocar memória
         return NULL;
      }

      novoNodo->info = item;
      novoNodo->esq = NULL;
      novoNodo->dir = NULL;

      return novoNodo;
   }
```

Tree insere(Tree t, int item): Esta função recebe a raiz da árvore (t) e um elemento inteiro (item) que será inserido na árvore. A função retorna a raiz da árvore atualizada após a inserção.

if (t == NULL): Verifica se a árvore está vazia (quando a raiz é NULL) ou se o local de inserção foi encontrado.
Nodo* novoNodo = (Nodo*)malloc(sizeof(Nodo));: Aloca dinamicamente memória para um novo nó da árvore. Nodo parece ser o tipo de estrutura que representa um nó na árvore.
if (novoNodo == NULL) { return NULL; }: Verifica se a alocação de memória foi bem-sucedida. Se não, a função retorna NULL, indicando um erro na alocação.
novoNodo->info = item;: Atribui o valor do elemento (item) ao campo info do novo nó.

novoNodo->esq = NULL; e novoNodo->dir = NULL;: Inicializa os ponteiros para os filhos esquerdo e direito do novo nó como NULL. Isso é importante para garantir que o novo nó seja uma folha (sem filhos) no momento da inserção.

return novoNodo;: Retorna o ponteiro para o novo nó, que agora se tornou a raiz da árvore ou foi inserido como filho de um nó existente.

- 2.4 MÉTODO DE BUSCA:

```
// Função para buscar um elemento na árvore
Tree busca(Tree t, int chave) {
    // Caso base: árvore vazia ou chave encontrada
    if (t == NULL || t->info == chave) {
        return t;
    }

    // Se a chave for menor que o valor do nó, busca na subárvore esquerda
    if (chave < t->info) {
        return busca(t->esq, chave);
    }
    // Se a chave for maior que o valor do nó, busca na subárvore direita
    else {
        return busca(t->dir, chave);
    }
}
```

Tree busca(Tree t, int chave): A função recebe a raiz da árvore (t) e a chave que está sendo buscada. Retorna um ponteiro para o nó que contém a chave ou NULL se a chave não for encontrada.

if (t == NULL || t->info == chave) { return t; }: Este é o caso base da recursão. Se a árvore estiver vazia (t == NULL) ou se a chave for encontrada no nó atual (t->info == chave), o ponteiro para o nó atual é retornado.

if (chave < t->info) { return busca(t->esq, chave); }: Se a chave for menor que o valor do nó atual, a busca continua na subárvore esquerda, chamando recursivamente a função busca com a subárvore esquerda como nova raiz.

else { return busca(t->dir, chave); }: Se a chave for maior que o valor do nó atual, a busca continua na subárvore direita, chamando recursivamente a função busca com a subárvore direita como nova raiz.

Essa função utiliza a propriedade de árvores binárias, onde todos os elementos na subárvore esquerda de um nó são menores que o valor do nó e todos os elementos na subárvore direita são maiores. Portanto, ela realiza uma busca eficiente na árvore binária para encontrar o elemento desejado

- 2.5 MÉTODO ENCONTRAR MENOR:

```
// Função para encontrar o nó mais à esquerda em uma subárvore
Tree encontraMenor(Tree t) {
   while (t->esq != NULL) {
      t = t->esq;
   }
   return t;
}
```

Esta função encontra e retorna o nó mais à esquerda em uma subárvore. A lógica é simples: ela percorre os nós à esquerda da árvore até encontrar o nó mais à esquerda, que será o menor nó na subárvore.

while (t->esq != NULL): Enquanto houver um filho à esquerda, atualiza o ponteiro t para apontar para o filho à esquerda.

return t;: Quando não houver mais filhos à esquerda, o nó atual (t) é o nó mais à esquerda na subárvore, então esse nó é retornado.

- 2.6 REMOVER:

```
Tree removeNo(Tree t, int chave) {
   if (t == NULL) {
      return t;
   }

   // Procura o nó a ser removido
```

```
    if (chave < t->info) {
        t->esq = removeNo(t->esq, chave);
    } else if (chave > t->info) {
        t->dir = removeNo(t->dir, chave);
    } else {
        // Caso 1: Nó sem filhos ou com apenas um filho
        if (t->esq == NULL) {
            Tree temp = t->dir;
            free(t);
            return temp;
        } else if (t->dir == NULL) {
            Tree temp = t->esq;
            free(t);
            return temp;
        }

        // Caso 2: Nó com dois filhos
        Tree temp = encontraMenor(t->dir);
        t->info = temp->info;
        t->dir = removeNo(t->dir, temp->info);
    }

    return t;
}
```

A função removeNo é responsável por remover um nó com uma chave específica da árvore binária.

`if (t == NULL) { return t; }`: Se a árvore (ou subárvore) estiver vazia, não há nada para remover, então a função retorna a árvore como está.

A estrutura condicional seguinte `(if (chave < t->info) { ... } else if (chave > t->info) { ... } else { ... })` é usada para encontrar o nó a ser removido.

Caso 1: Se o nó a ser removido não tem filhos ou tem apenas um filho, o código libera a memória do nó a ser removido e ajusta os ponteiros dos pais para não apontarem mais para esse nó.

Caso 2: Se o nó a ser removido tem dois filhos, ele encontra o menor nó na subárvore direita (usando a função encontraMenor), copia o valor desse nó para o nó a ser removido e, em seguida, remove recursivamente esse menor nó.

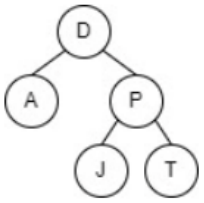
`return t;`: Retorna a árvore modificada após a remoção.

EXERCÍCIOS:

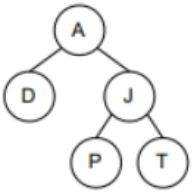
Agora com base na vídeo aula e no material de suporte, implemente as funções básicas (você pode melhorar seu TAD 🙂) e responda os exercícios abaixo:

1. Uma árvore consiste em uma estrutura de dados que contém nós e arcos e pode ser utilizada para organizar objetos de forma hierárquica. Uma árvore binária de busca utiliza a estrutura de árvore e possui propriedades específicas. Assinale a alternativa que apresenta uma árvore binária de

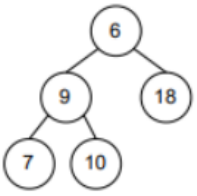
►a)



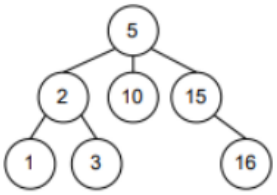
b)



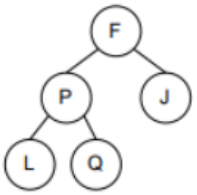
c)



d)



e)



2. Suponha que você tenha uma árvore binária de busca com n nós. Qual é a complexidade de tempo de busca se a árvore binária for balanceada ?

A O(n)

B O(log n)

C O(n log n)

D O(n^2)

E O(log^2 n)

3. Implemente uma função para calcular a altura de uma árvore binária.

4. Pergunta: Escreva uma função para contar o número de nós em uma árvore binária.(Dica: a altura da árvore é a distância entre a raiz e o nó folha mais distante)

5. Nas estruturas conhecidas como árvores, o nó do topo da árvore, do qual descendem os demais nós, denomina-se nó

A interior.

B terminal.

C raiz.

D exterior.

E filho.