ESCOLA DE PRIMAVERA DA MARATONA DE PROGRAMAÇÃO

PROMOÇÃO:

CEFET-MG

UFMG

UFV

UNIFEI

USP Universidade de São Paulo

UFU

APOIO:

beecrowd

SBC Sociedade Brasileira de Computação

Grupo de Computação Competitiva

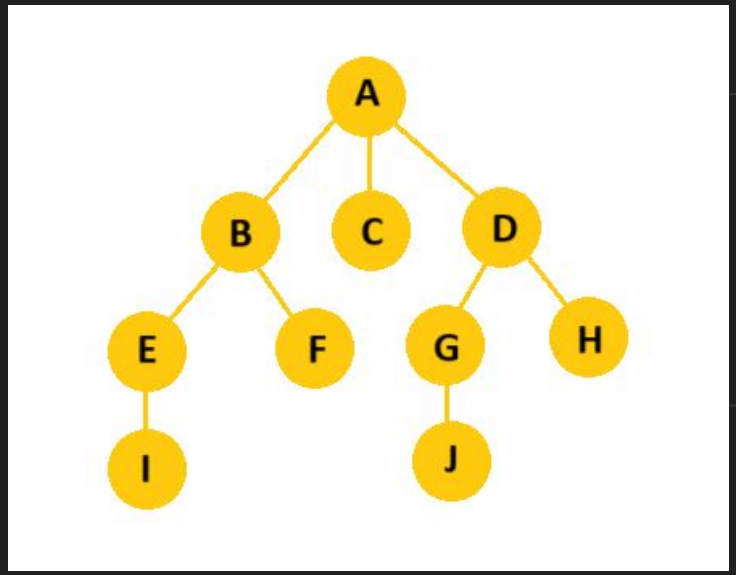
ÁRVORE BINÁRIA

>_

Por: *Milena Bueno Maciel*

CONTEÚDOS

01 - DEFINIÇÃO DE ÁRVORES



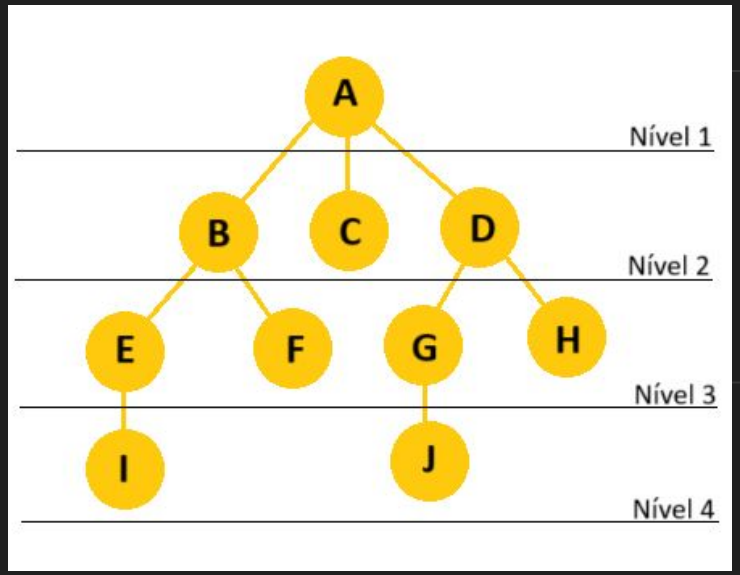
Árvores são um tipo especial de grafo conexo e sem ciclos, ou seja, existe um único caminho entre qualquer par de vértices

O nó A é chamado de raiz pois é o ponto de partida da estrutura. A partir dele, existem os nós filhos que podem ter seus próprios descendentes, formando subárvores.

Os nós folhas são aqueles que não possuem filhos.

Exemplo: I, F,C, J e H.

01 - DEFINIÇÃO DE ÁRVORES

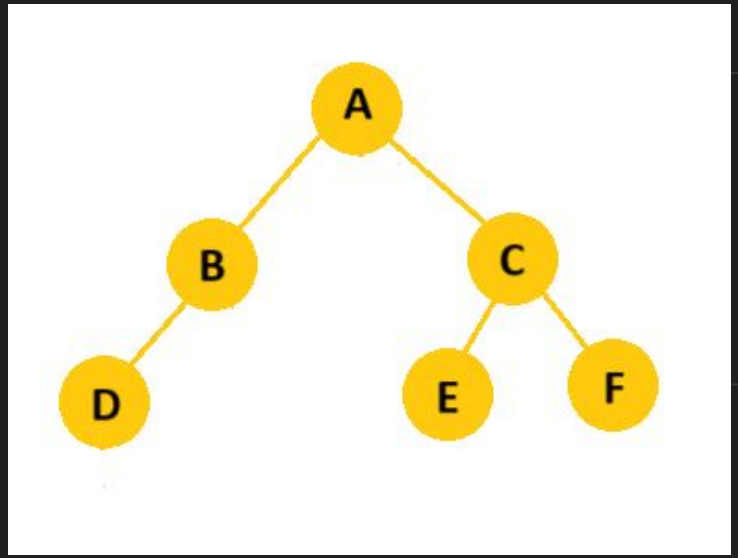


Nível: o nó raiz está localizado no nível 1. A partir dele, cada nó filho possui um nível a mais que o nível de seu pai.

Altura ou profundidade da árvore: É o máximo nível dos nós da árvore.

Grau de um nó: É o número de filhos do nó.

02 - ÁRVORE BINÁRIA DE BUSCA

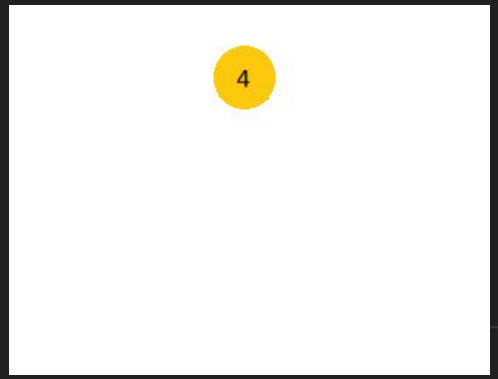


Uma Árvore Binária possui nós com grau máximo 2 e não possui uma regra específica de ordenação

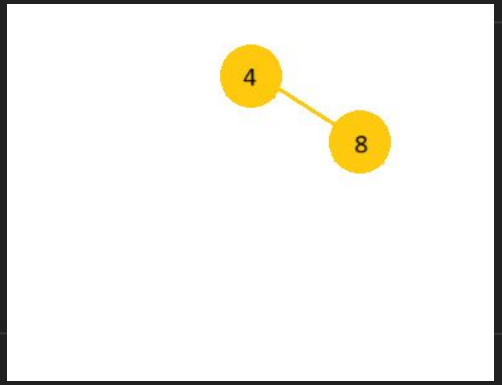
Uma Árvore Binária de Busca possui uma regra de ordenação, onde o nó da esquerda tem um valor menor que o valor do nó pai e o nó da direita tem um valor maior.

02 - ÁRVORE BINÁRIA DE BUSCA

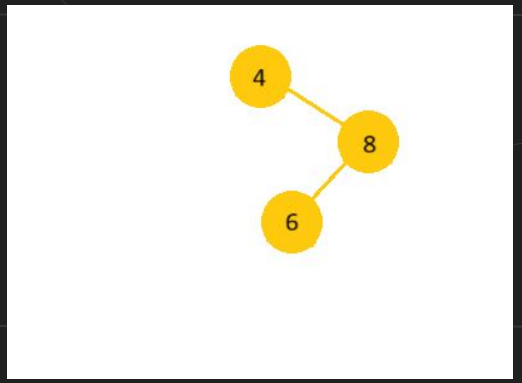
Valores inseridos: 4



Valores inseridos: 4 - 8

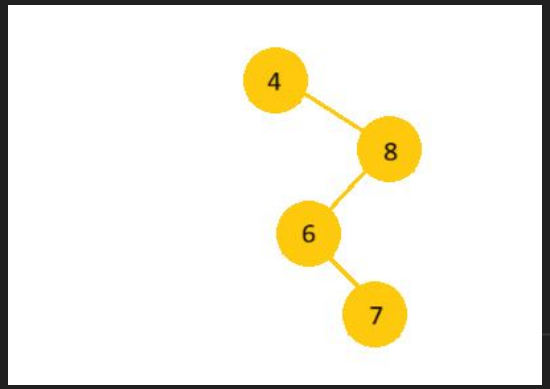


Valores inseridos: 4 - 8 - 6

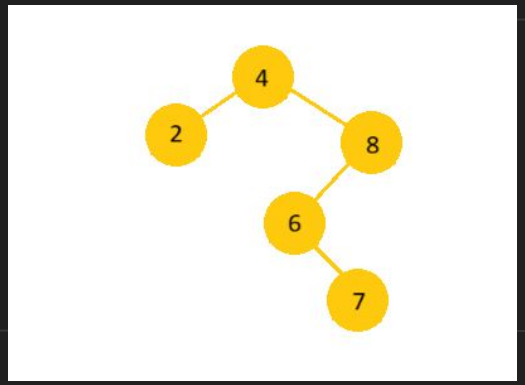


02 - ÁRVORE BINÁRIA DE BUSCA

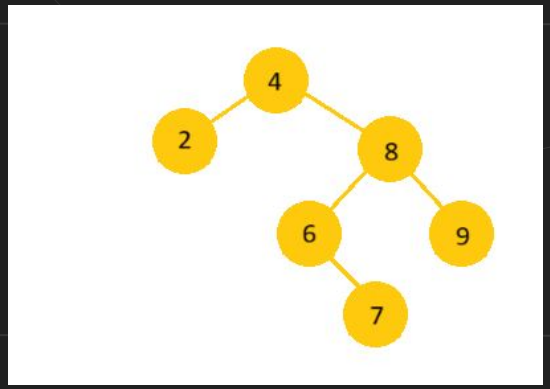
Valores inseridos: 4 - 8 - 6 - 7



Valores inseridos: 4 - 8 - 6 - 7 - 2



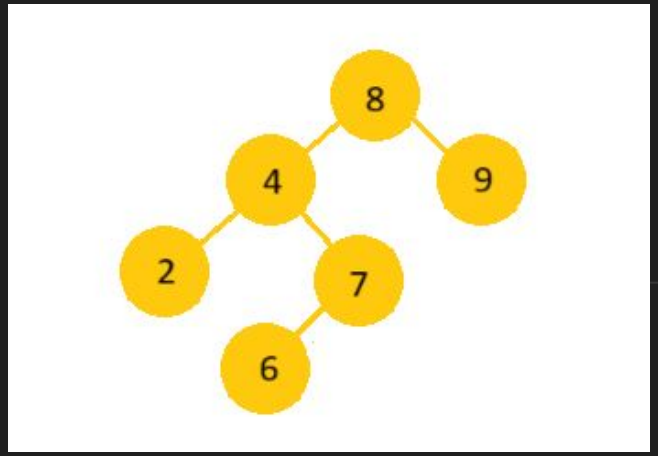
Valores inseridos: 4 - 8 - 6 - 7 - 2 - 9



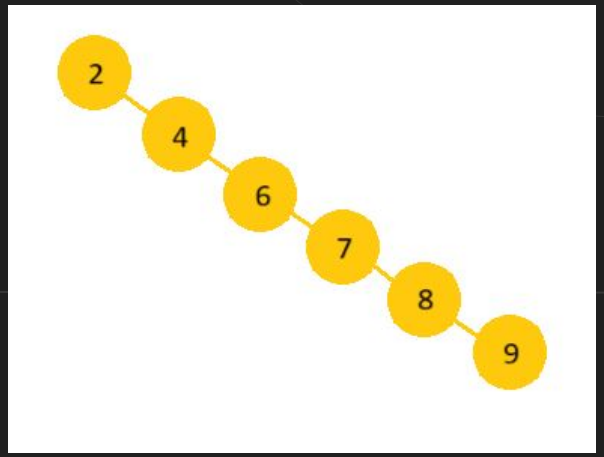
02 - ÁRVORE BINÁRIA DE BUSCA

Diferentes ordens de inserção dos mesmos números, geram diferentes árvores binárias

8 - 9 - 4 - 7 - 6 - 2



2 - 4 - 6 - 7 - 8 - 9



03 - FUNÇÕES BÁSICAS - CRIAR NÓS

A struct Node possui todas as características

que os nós da árvore precisam ter:

- Um valor (key)
- Um ponteiro para seu filho da direita
- Um ponteiro para seu filho da esquerda

```
7   struct Node {
8       int key;
9       Node* left;
10      Node* right;
11      Node(int item) {
12          key = item;
13          left = NULL;
14          right = NULL;
15      }
16  };
17  int main() { _
18      Node* root = new Node(100);
19      return 0;
20  }
```

03 - FUNÇÕES BÁSICAS - INSERT

Percorre a árvore a partir da raiz comparando os valores dos nós até encontrar um nó vazio.

Cria um novo nó com o valor inserido.

```
17  Node* insert(Node* node, int key) {
18      if (node == NULL)
19          return new Node(key);
20      if (node->key <= key)
21          node->right = insert(node->right, key);
22      else
23          node->left = insert(node->left, key);
24      return node;
25  }
26  int main() { _
27      Node* root = NULL;
28      root = insert(root, 10);
29      root = insert(root, 20);
30      root = insert(root, 200);
31      return 0;
32  }
```

03 - FUNÇÕES BÁSICAS - SEARCH

Percorre a árvore a partir da raiz comparando os valores dos nós até encontrar o valor solicitado

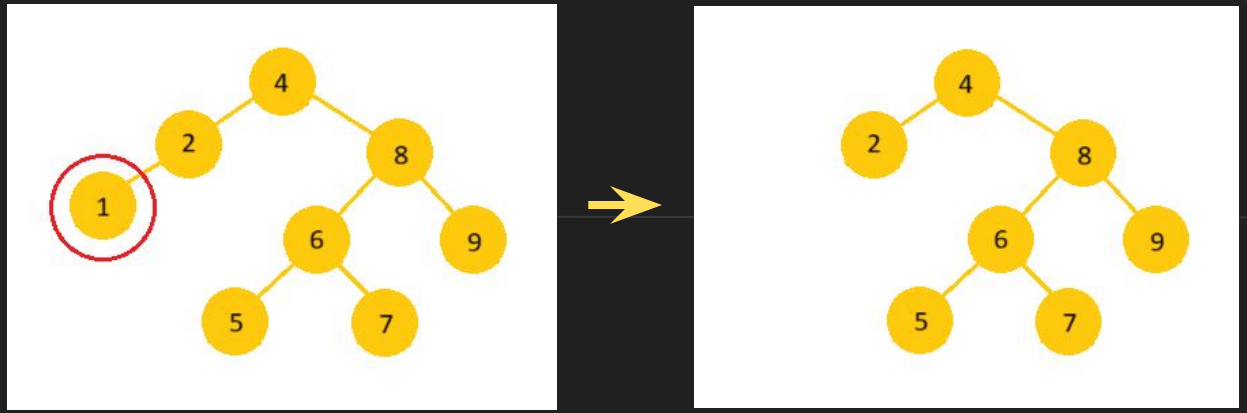
Se a função não encontrar o valor na árvore, ela retornará um ponteiro nulo

```
27  Node* search(Node* root, int key) {
28      if (root == NULL || root->key == key)
29          return root;
30      if (root->key < key)
31          return search(root->right, key);
32      return search(root->left, key);
33  }
34  int main() { _
35      Node* root = NULL;
36      root = insert(root, 10);
37      if(search(root, 10) != NULL)
38          cout << "Found" << endl;
39      else
40          cout << "Not Found" << endl;
41      return 0;
42  }
```

03 - FUNÇÕES BÁSICAS - DELETE

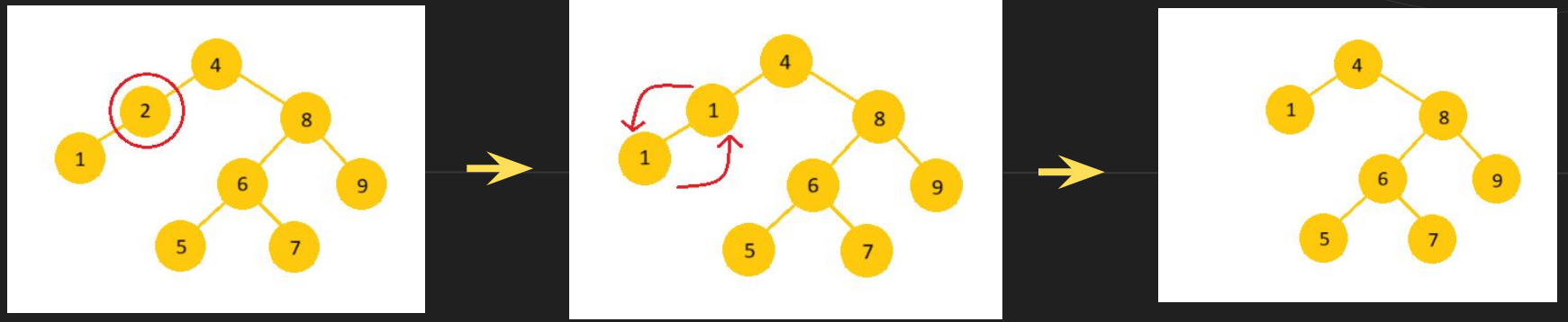
Para excluir um nó de uma árvore binária existem 3 situações possíveis:

1. Nó sem filhos: basta deletar aquele nó sem realizar nenhuma modificação na árvore.



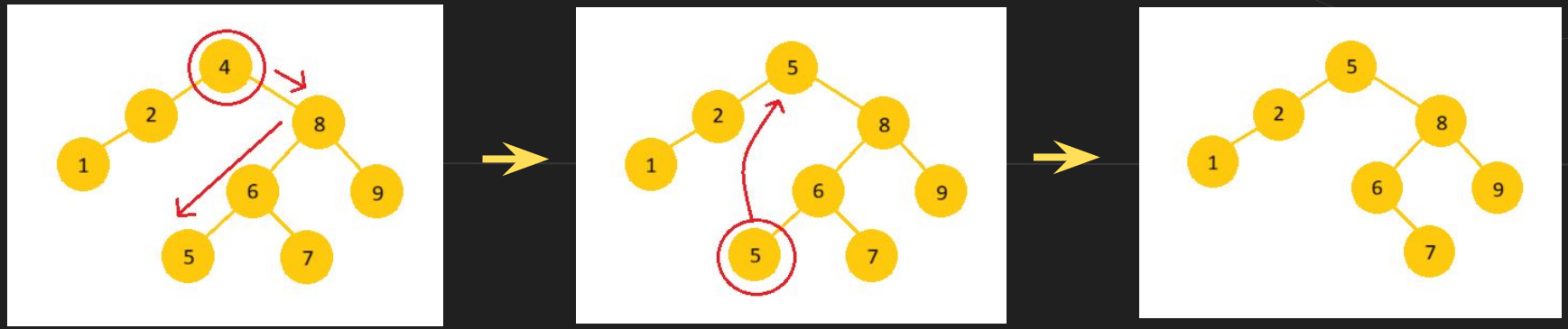
03 - FUNÇÕES BÁSICAS - DELETE

2. Nó com um filho: será necessário replicar o nó filho no lugar do nó que será excluído e depois excluí o nó filho original, que agora está duplicado, como no caso 1.



03 - FUNÇÕES BÁSICAS - DELETE

3. Nó com dois filhos: nesse caso, pegamos o nó com menor valor da subárvore da direita e colocamos no lugar do nó que será excluído.



03 - FUNÇÕES BÁSICAS - DELETE

Primeiramente, será necessário criar a função getSuccessor, responsável por encontrar o elemento que será substituído pelo nó deletado no terceiro caso.

```
27  Node* getSuccessor(Node* node)
28  {
29      node = node->right;
30      while (node != NULL && node->left != NULL)
31          node = node->left;
32      return node;
33  }
```

03 - FUNÇÕES BÁSICAS - DELETE

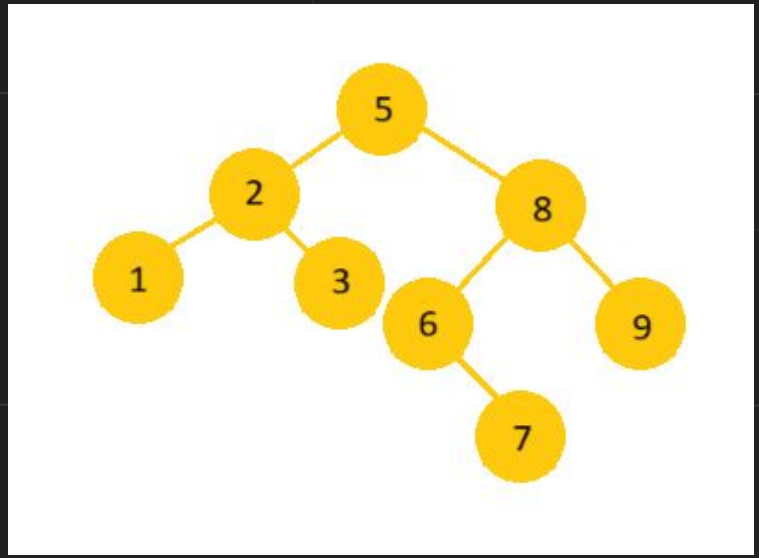
```
Node* delNode(Node* root, int val)
{
    if (root == NULL)
        return root;

    if (root->key > val)
        root->left = delNode(root->left, val);
    else if (root->key < val)
        root->right = delNode(root->right, val);

    else {
        if (root->left == NULL) {
            Node* temp = root->right;
            delete root;
            return temp;
        }
        if (root->right == NULL) {
            Node* temp = root->left;
            delete root;
            return temp;
        }

        Node* succ = getSuccessor(root);
        root->key = succ->key;
        root->right = delNode(root->right, succ->key);
    }
    return root;
}
```

04 - PERCORRENDO A ÁRVORE - INORDER

A travessia inorder fornece os valores da árvore binária em ordem crescente.

A ordem de prioridade na leitura é: subárvore da esquerda - raiz - subárvore da direita

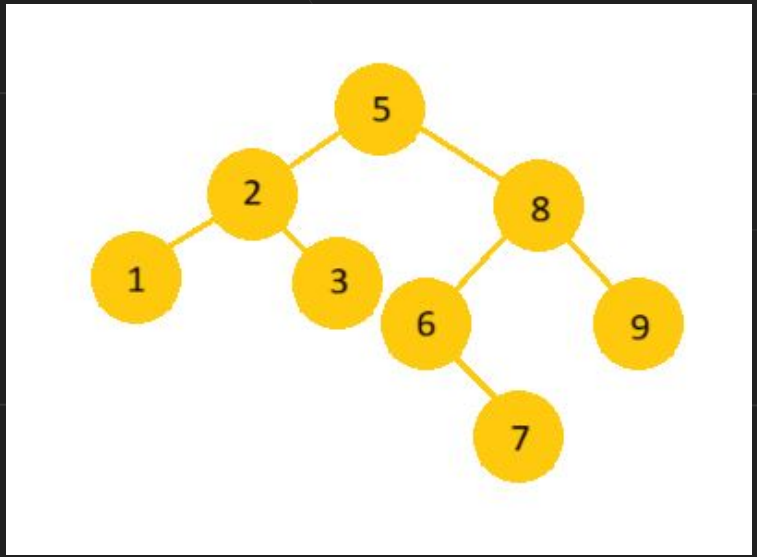
Saída: 1, 2, 3, 5, 6, 7, 8, 9



04 - PERCORRENDO A ÁRVORE - PREORDER

A ordem de prioridade na leitura é: raiz - subárvore da esquerda - subárvore da direita

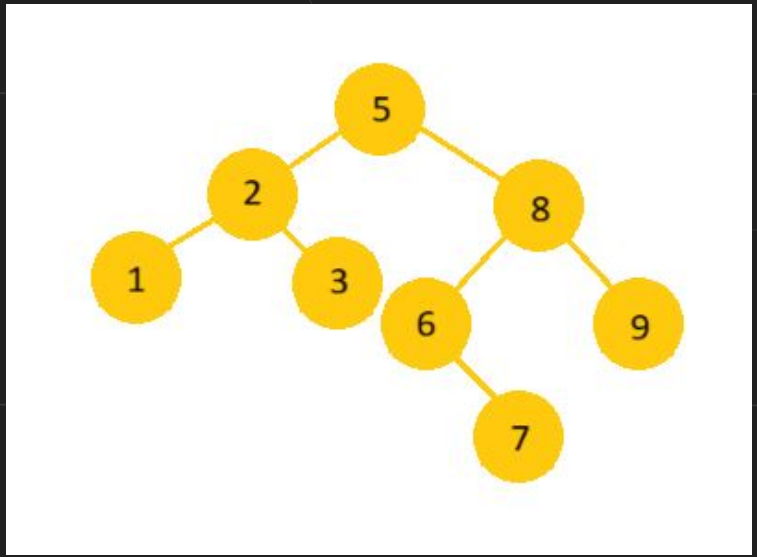
Saída: 5, 2, 1, 3, 8, 6, 7, 9



04 - PERCORRENDO A ÁRVORE - POSTORDER

A ordem de prioridade na leitura é: subárvore da esquerda - subárvore da direita - raiz

Saída: 1, 3, 2, 7, 6, 9, 8, 5



04 - PERCORRENDO A ÁRVORE

```
void printInorder(Node* node)
{
    if (node == NULL)
        return;

    printInorder(node->left);

    cout << node->key << " ";

    printInorder(node->right);
}
```

```
void printPreOrder(Node* node)
{
    if (node == NULL)
        return;
    cout << node->key << " ";

    printPreOrder(node->left);
    printPreOrder(node->right);
}
```

```
void printPostOrder(Node* node)
{
    if (node == NULL)
        return;
    printPostOrder(node->left);
    printPostOrder(node->right);

    cout << node->key << " ";
}
```

05 - LINK ALGORITMO COMPLETO

https://github.com/MilenaBMaciel/TemplateArvoreBinaria/blob/main/arvBin.cpp

06 - QUESTÃO 1200 - BEECROWD

https://judge.beecrowd.com/pt/problems/view/1200

06 - QUESTÃO 1200 - BEECROWD

Para esse problema, bastou utilizar o template padrão de árvore binária com pequenas alterações, como a troca da key de int para char e a lógica extra para printar os elementos respeitando os critérios de espaçamentos. Além disso, foi necessário uma lógica simples para seleção da operação que seria utilizada.

OBRIGADO PELA ATENÇÃO

Grupo de Computação Competitiva

gcc-cefetmg.github.io/Site-GCC

@gcc.cefetmg