Departamento de Engenharias e Tecnologia
Algoritmos e Estruturas de Dados II

Prof. Bruno Xavier
bruno.xavier@ufersa.edu.br

Desafios de Programação

2025.2

1. Encontre o elemento com a menor chave e sucessor em ordem de uma BST dada.

```
Node* getMin(Node* root);
Node* getSuccessor(Node* root);
```

2. Escreva três funções que devem retornar o tamanho e a altura de uma árvore binária, além da mediana das chaves de todos os nós.

```
int getSize(Node* root);
int getHeight(Node* root);
int getMedian(Node* root);
```

3. Implemente um método para remover o nó com a menor chave de uma BST.

```
Node* removeMin(Node* tree);
```

4. Dada uma BST, realiza a mudança de uma chave.

```
Node* changeKey(Node* root, int oldVal, int newVal);
```

5. Dada uma árvore binária, cheque se ela é ou não uma BST.

```
bool isBST(Node* root);
```

6. Implemente uma função de inserção em uma BST que seja iterativa (com `for` ou `while`).

7. Implemente uma função de inserção em uma BST que simule a passagem por referência.

```
void insertNode (Node** node, int key)
```

   Ou seja, `insertNode` não possui retorno, mas deve atualizar `node` (a árvore passada como parâmetro) internamente. Exemplo de uso:

```
Node* root = NULL;
insertNode(&root, 50);
insertNode(&root, 30);
insertNode(&root, 70);
```

8. Implemente uma função de busca em uma BST que seja iterativa (com `for` ou `while`).

9. Escreva uma função `traverse` que terá como parâmetros uma árvore, uma flag (`PRE`, `IN` ou `POS`) que determinará o percurso em pré-ordem, ordem ou pós-ordem e um ponteiro para uma função. A função `traverse` deverá retornar o percurso correspondente da árvore.

```
enum order {
  PRE, // Pre-ordem
  IN,  // Ordem
  POS, // Pos-ordem
};
typedef enum order Order;

void printNode(Node* node){
  if (node)
    printf("%d ", node->key);
}

void traverse(Node* node, Order ord, void (*visit)(Node* node));
```

A sugestão é usar *switch-case* sobre `ord` para implementar a lógica de qual percurso acionar. Em cada caso do *switch* você deve implementar a travessia corresponde usando a própria função recursivamente e a função `visit` para processar o nó. Exemplo de uso:

```
printf("Percurso em Pre-ordem: ");
traverse(root, PRE, printNode);
printf("\n");

printf("Percurso em Ordem: ");
traverse(root, IN, printNode);
printf("\n");

printf("Percurso em Pos-ordem: ");
traverse(root, POS, printNode);
printf("\n");
```

10. Escreva um programa que abra e leia um arquivo de texto e registre quantas vezes cada palavra ocorre no arquivo. Use uma BST modificada para armazenar tanto uma palavra quanto o número de vezes que ela ocorre. Depois que o programa tiver lido o arquivo, ele deve oferecer um menu com três opções. A primeira é listar todas as palavras junto com o número de ocorrências. A segunda é permitir que você digite uma palavra, com o programa relatando quantas vezes a palavra ocorreu no arquivo. A terceira opção é sair.

    O nó da árvore precisará de três campos:

    - `char* word`: Uma palavra.
    - `int count`: Um contador para o número de ocorrências.
    - `Node* left, Node* right`: Ponteiros para os nós filhos.

    O fluxo principal do programa seria:

    - **Abrir o arquivo**: Use `fopen` para abrir o arquivo de texto em modo de leitura.
    - **Ler e processar o arquivo**:
        - Leia o arquivo palavra por palavra.
        - Para cada palavra, use uma função de inserção na BST:
            * Se a palavra já existir na árvore, apenas incremente o contador (`count`) do nó correspondente.
            * Se a palavra não existir, adicione um novo nó à árvore com o contador inicializado em 1.
    - **Apresentar o menu**: Depois de ler todo o arquivo, entre em um loop que exibe as três opções e espera pela escolha do usuário.
    - **Executar as opções**:
        - **Opção 1 (Listar tudo)**: Use uma travessia em ordem na BST. Essa travessia é ideal porque ela listará as palavras em ordem alfabética. Para cada nó, imprima a palavra e sua contagem.
        - **Opção 2 (Buscar palavra)**: Crie uma função de busca na BST que recebe uma palavra como argumento. A função deve retornar o nó se a palavra for encontrada. Se o nó for encontrado, imprima a contagem. Caso contrário, informe que a palavra não foi encontrada.
        - **Opção 3 (Sair)**: Saia do loop e termine o programa. Lembre-se de liberar toda a memória alocada para a árvore antes de fechar o programa para evitar vazamentos de memória.