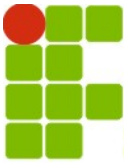

INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: ALGORITMOS E ESTRUTURAS DE DADOS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: ÁRVORE BINÁRIA DE BUSCA (BST)

Prática 08

Parte 1: Preparação

Passo 1: Crie um novo projeto e adicione os arquivos que acompanham a prática ao projeto.

> **main.cpp**: contém a função main, que testa as árvores binária a AVL. É possível ajustar que testes serão realizados durante o desenvolvimento (parâmetros da função de teste), mas ao final todos os testes devem estar setados.
>
> **bst.h** e **bst.cpp:** declaração e implementação da árvore binária de busca.
>
> **avl.h** e **avl.cpp:** declaração e implementação da árvore AVL. **(Prática 09)**

Passo 2: Compile e rode o código para verificar se não há erros.

> Neste ponto deve apenas compilar e rodar, mas não deve passar nos teste já que algumas funções não estão implementadas.

Passo 3: Estude o código para se familiarizar.

> Veja o material de aula se necessário. Faça comentários nos trechos mais complicados. Refatore o código se achar que vai ajudar seu trabalho. Teste novamente antes de fazer modificações mais profundas.

Parte 2: Implementando funções na Árvore Binária de Busca (BST)

Passo 1: Implemente a função de inserção na BST.

> A função a ser implementado é a versão privada, cujo corpo está em **bst.cpp**. A implementação mais direta é recursiva. A função é chamada passando um nó da árvore (inicialmente a raiz, depois esquerdo ou direito), e o retorno permite atualizar os ponteiros do nó pai (ou a raiz, quando for a primeira inserção). Depois de inserir o novo elemento, a altura do nó deve ser atualizada (chame `updateH()` ao final da inserção); isso será usado pela árvore AVL.
>
> **Importante:** caso a chave já exista na árvore, não faça nada. Duplicações na árvore quebram o funcionamento da árvore AVL.

Passo 2: Implemente a função de exibição da BST.

> A função `show()` privada deve varrer e exibir os nós da árvore em ordem (ver "Percurso" no material). A função deve exibir não só o valor da chave e mas também a altura do nó no formato "(chave, altura)".

Passo 3: Implemente a função de busca na BST.

> Há duas versões da função `search()`: uma pública e outra privada ; você deve trabalhar na versão privada (ela é usada pela pública). Siga o material da aula.

Passo 4: Implemente a função `successor()` em **bst.cpp**.

Em uma árvore de busca, o predecessor de um valor X presente na árvore é o maior valor Y também presente na árvore que é menor que X. Isto é, se pegássemos os elementos da árvore em ordem, Y seria o valor que viria imediatamente antes de X (se houver). O método `predecessor()` realiza essa busca.

Simetricamente, o sucessor de X é o valor W que viria logo após X na sequência de elementos presentes na árvore. Implemente o método `successor()` se baseando na implementação de `predecessor()`. Estude esse método para fazer os modificações necessárias.

Passo 5: Compile e teste a aplicação.

Rode a aplicação, verificando se passa nos testes.