Aula prática 7

Esta aula tem como objetivo estudar a utilização de árvores binárias de pesquisa (BST). São disponibilizadas implementações de algumas operações de manipulação destas estruturas de dados e pretende-se adicionar novas funcionalidades e comparar a performance de ambas em alguns casos de aplicação.

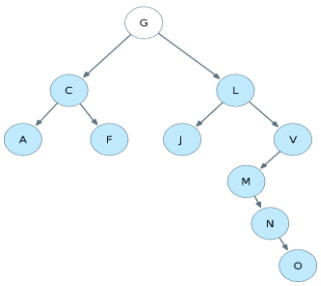
1. Considere a classe `Tree` que representa uma árvore binária de pesquisa (BST). São fornecidos os ficheiros ".cpp" e ".hpp" com as estruturas definidas e código parcialmente implementado:

- `bst.cpp` – implementação das árvores BST (para completar na aula);
- `bst.hpp` – estrutura das árvores BST (não alterar);
- `bst_test.cpp` – testes das funções (para completar na aula).

Estude cuidadosamente as estruturas de dados fornecidas, observando que os nós das árvores contêm um atributo *string* com a informação representada pelo nó e dois apontadores para os nós filhos (esquerdo e direito).

Analise o código fornecido, que inclui funções para:
- criar uma árvore vazia;
- inserir e eliminar elementos da árvore;
- identificar os elementos mínimo e máximo;
- pesquisar uma *string* e retornar o respetivo nó.

a) No ficheiro **`bst_test.cpp`**, implemente a inserção de elementos na árvore. O código deverá inserir os elementos do vetor `vtr` na árvore `obj_tree`. O resultado desta operação deverá ser uma árvore conforme a figura ao lado.



b) Implemente, no ficheiro **`bst.cpp`**, a função `bst_altura()`, que deverá retornar a altura de um determinado nó. Sugere-se a implementação desta função de forma recursiva, dada a definição também recursiva da altura de um nó (1 + altura do filho de maior altura). Teste a função com a raiz da árvore criada anteriormente e verifique que a sua altura é de 5.

c) Implemente, no ficheiro **`bst.cpp`**, a função `bst_preordem()`, que percorre uma árvore em pré-ordem e imprime as *strings* apontadas por cada nó. Repare que também esta função deve ser implementada de forma recursiva. Teste a função utilizando a árvore criada anteriormente. O resultado deverá ser o seguinte:

```
Travessia em pré-ordem da árvore BST: G C A F L J V M N O
```

d) Implemente, no ficheiro **`bst.cpp`**, a função `bst_posordem()`, que percorre uma árvore em pós-ordem e imprime as strings apontadas por cada nó. Esta função também pode ser implementada de maneira recursiva. Teste a função utilizando a árvore criada anteriormente.

O resultado deverá ser o seguinte:

```
Travessia em pos-ordem da árvore BST: A F C J O N M V L G
```

e) Analise agora o código da função `bst_remove()`. Da árvore criada na alínea a), no ficheiro **`bst_test.cpp`** utilize a função `bst_remove()` para remover o elemento "N". Neste caso, como ficaria a árvore final? Represente graficamente o resultado.

2. Considere a classe `AVLTree` que representa uma árvore binária de pesquisa (BST). São fornecidos os ficheiros ".cpp" e ".hpp" com as estruturas definidas e código parcialmente implementado.

- `avl.cpp` – implementação da árvore (para completar na aula);
- `avl.hpp` – estrutura da árvore (não alterar);
- `avl_test.cpp` – testes das funções.

a) Altere, no ficheiro **`avl.hpp`**, o *struct* `Node` para que também possa representar um nó de uma árvore AVL, nesse caso deve-se criar um novo atributo chamado `heigth` do tipo `int` para guardar a altura de cada nó.

b) Altere, no ficheiro **`avl.cpp`**, a função `insert(string v)`, para inserir os nós na árvore em que cada nó terá um valor `string` e também irá guardar a sua altura. O resultado deverá ser o seguinte:

```
G 5
C 1
A 0
F 0
L 4
J 0
V 3
M 2
N 1
O 0
```

c) Implemente, no ficheiro **`avl.cpp`**, a função `isbalanced(Node *treeNode)` que percorre a árvore a partir do nó raiz e verifica se a árvore está balanceada. O resultado deverá ser o seguinte:

```
A arvore não esta balanceada
```

3. Considere as class Car e Node com os seguintes atributos:

Car	Node
string brand;	Car* car;
string model;	Node* left;
int year;	Node* right;

Considere a class BST com nós da class Node. Em que ficam organizados na árvore em função do *brand* e do *model.* Á esquerda de qualquer nó só podem estar *brand* menores ou iguais e *model* menores. E à direita só podem estar *brand* maiores ou iguais e *model* maiores.

Analisa todo o código fornecido e implemente as seguintes funções:

a) Implemente a função da class BST int carsYear(Node* root,int cyear) que conta todos os carros de um determinado ano.

b) Implemente a função da class BST void printbrand(Node* root,string cbrand) que imprime todos os carros de uma determinada marca.