Aula prática 7

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

1. Considere as bibliotecas de árvores binária de pesquisa (BST) e árvores AVL, disponibilizadas no ficheiro **PROG2_1718_P07.zip**. São fornecidos os ficheiros ".c" e ".h" com as estruturas definidas e código parcialmente implementado:

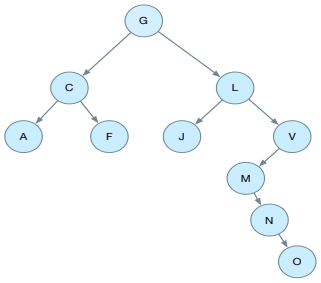
- `bst` – estrutura e código associado das árvores BST (para completar na aula);
- `avl` – estrutura e código associado das árvores AVL (para completar na aula);

Estude cuidadosamente as estruturas de dados fornecidas, observando que os nós das árvores contêm em ambos os casos um apontador para a *string* com a informação representada pelo nó, para além dos apontadores para os dois filhos (esquerdo e direito). Verifique ainda que no caso das árvores AVL se guarda também em cada nó a sua própria altura.

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) Num ficheiro distinto dos fornecidos (por exemplo **`arvores-teste.c`**), crie um programa que utilize as bibliotecas fornecidas e no qual deverá testar o código a implementar nesta e nas alíneas seguintes. Comece por criar uma árvore do tipo BST, inserindo os seguintes elementos pela ordem indicada: "G", "C", "A", "F", "L", "J", "V", "M", "N" e "O". O resultado desta operação deverá ser uma árvore conforme a figura ao lado.



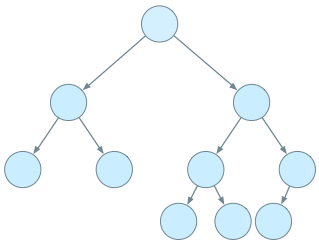
b) Implemente, no ficheiro **`bst.c`**, a função `bst_altura()`, que deverá retornar a altura de um determinado nó. Sugere-se a implementação desta função de forma recursiva, dada a definição também recursiva da altura de um nó (1 + altura do filho de maior altura). Teste a função com a raiz da árvore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

c) Implemente, no ficheiro **`bst.c`**, a função `bst_preordem_impl()`, 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 e que a mesma é chamada pela função (já implementada) `bst_imprime()`. 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) Verifique que no ficheiro **`avl.c`**, as funções `avl_altura()` e `avl_preordem_impl()` já se encontram implementadas. Existem diferenças entre estas funções e as homónimas para o caso das árvores BST? Em caso afirmativo, explique-as.

e) Analise cuidadosamente o código da função `avl_insere()`. Crie no programa de testes uma árvore AVL, introduzindo os mesmos elementos da alínea a) e pela mesma ordem. A árvore resultante deverá ter a estrutura da figura ao lado. Valide esse facto utilizando o *debugger* para inspecionar cada nó da árvore. Complete a figura ao lado com o conteúdo dos nós.



f) Qual é a altura da árvore resultante? Verifique o resultado no programa de testes. Compare esta árvore com a árvore BST da alínea a).

g) Analise agora o código da função `avl_remove()`. Da árvore criada na alínea e) remova o elemento "N". Observe que se trata de uma remoção que não obriga ao balanceamento de qualquer sub-árvore. Represente graficamente o resultado.

h) Remova agora, da árvore resultante, o elemento "V". Neste caso é necessária alguma rotação para balancear a árvore? De que tipo? Represente graficamente o resultado.

2. Pretende-se estudar a performance de algumas operações em árvores binárias de pesquisa e em árvores AVL. Em concreto, pretende-se comparar os tempos de execução das operações de inserção e pesquisa.

Nota: Para cronometrar a execução de funções pode usar a biblioteca time (#include <time.h>) para obter a hora atual. Use o exemplo seguinte como referência:

```
#include <time.h>
#include <stdio.h>

int main()
{
    clock_t inicio, fim;
    double tempo;

    inicio = clock()

    /* tarefa a verificar */

    fim = clock();
    tempo = (double)(fim - inicio) / CLOCKS_PER_SEC;
    printf("tempo em segundos: %lf\n", tempo);
}
```

a) O ficheiro de texto **`cidades.txt`** contém uma cidade por linha, acompanhada do respetivo país (separado por vírgula). Compare e explique a eventual diferença de tempo de execução entre a inserção da totalidade das cidades do ficheiro numa árvore BST e numa árvore AVL.

b) Compare o tempo de execução da pesquisa da cidade "Zywiec,Poland" em cada um dos tipos de árvores. Como justifica as diferenças?

c) Repita o exercício da alínea a), utilizando agora o ficheiro **`cidades_sorted.txt`**, que contém a mesma lista de cidades, embora neste caso ordenada alfabeticamente. Explique as diferenças encontradas.

3. Pretende-se agora implementar um contador de palavras que utilize árvores AVL para guardar o número de ocorrências de cada uma das *strings* contidas num ficheiro de texto. De cada vez que se tenta inserir uma *string* na árvore deve verificar-se se a mesma já existe e, em caso afirmativo, incrementar o contador associado ao nó respetivo, ao invés de criar um novo nó.

a) Altere, no ficheiro **`avl.h`**, a estrutura dos nós das árvores por forma a que os mesmos incluam um contador (valor inteiro). Altere também todas as funções necessárias por forma a que o contador associado a um nó contenha o número de vezes que a respetiva *string* foi adicionada à árvore.

b) Implemente o código necessário para, recorrendo a uma árvore AVL, realizar a contagem do número de cidades existentes em cada um dos países. Repare que não é necessário guardar cada cidade na árvore, basta guardar o seu país (sugestão: utilize a função `strtok()` para extrair da *string* lida o nome do país). Verifique o resultado para os países Portugal, Spain e Russia, que deverá ser de acordo com o seguinte:

```
Portugal – 460
Spain – 8107
Russia – 4571
```