DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

AED - 2020-2021 - 1º Semestre
Algoritmos e Estruturas de Dados

Laboratório 6 - Árvores

Semanas de 7 de dezembro de 2020 (turno par) e 14 de dezembro de 2020 (turno ímpar)

Duração: 2 horas

Neste laboratório estuda-se o problema da representação e manuseamento de uma estrutura de dados em árvore binária. É fornecido código, incompleto, para processar uma árvore lida de um ficheiro, e alguns casos de teste.

Estude o código do programa que se encontra nos ficheiros `labt.c, tree.c, tree.h` e `queue.h`:

- Identifique a estrutura de dados utilizada para representar a árvore binária (usando ponteiros) onde a informação associada a cada nó é um inteiro.
- Os números inteiros associados aos nós da árvore são lidos de um ficheiro cujo nome deve ser fornecido na linha de comando; a construção da árvore é efectuada, de forma recursiva, pelo pseudo-código seguinte:

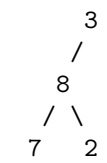
```
Construct ( ficheiro )
      Lê número inteiro do ficheiro
      no_actual = NewNode( número )
      se o no_actual foi criado
         filho1 = Construct ( ficheiro )
         filho2 = Construct ( ficheiro )
         liga filhos ao no_actual
      retornar no_actual
```

- A função `Construct( ficheiro )` recebe como argumento o ficheiro com números inteiros, e devolve um ponteiro para o nó raíz da árvore construída. Começa por ler do ficheiro um número inteiro e de seguida chama a função NewNode para criar um novo nó com o número lido. Caso o nó tenha sido efectivamente criado, são feitas chamadas recursivas à própria função para construir as sub-árvores relativas aos dois filhos.
- Observe no código a forma de implementação das funções de alocação (`AllocNode()`) e de inicialização (`NewNode( num )`) associadas. A `função NewNode( num )` começa por chamar a função `AllocNode()`, que aloca o espaço de memória para o novo nó. De seguida inicializa o valor do nó com o número passado como argumento, inicializa a `NULL` os ponteiros para os filhos do novo nó e retorna um ponteiro para o novo nó.
- No ficheiro de entrada devem ser inseridos números inteiros maiores ou iguais a zero, ou iguais a -1. Um número maior ou igual a zero significa que a árvore a construir tem um nó com esse número. O número -1 significa que um nó pai não tem um filho. Por exemplo, considere uma árvore binária com apenas um nó (raíz) com o valor 5. Tal árvore é representada por um ficheiro com os seguintes números:

```
5
-1
-1
```

O primeiro número -1 significa que o nó 5 não tem o filho da esquerda e o segundo -1 significa que o nó 5 também não tem o filho da direita.

Vejamos como segundo exemplo a seguinte árvore:



A árvore anterior é representada pelo seguinte ficheiro:

```
3                    "nó raiz"
8                    "filho da esquerda de 3"
7                    "filho da esquerda de 8"
-1                   "nó 7 não tem o filho da esquerda"
-1                   "nó 7 não tem o filho da direita"
2                    "filho da direita de 8"
-1                   "nó 2 não tem o filho da esquerda"
-1                   "nó 2 não tem o filho da direita"
-1                   "nó 3 não tem o filho da direita"
```

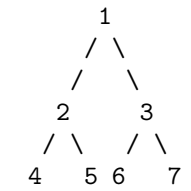
Note que a árvore é construída em profundidade da esquerda para a direita. Só quando se verifica que uma sub-árvore à esquerda não tem mais filhos é que se passa para a sub-árvore à direita.

No final da execução o programa faz uma impressão ("dump") da árvore introduzida com a estratégia pré-fixada, separando a raiz e os diferentes níveis em colunas verticais.

1. Indique qual a árvore gerada se introduzir no ficheiro a seguinte sequência de dados:

   1 2 3 -1 -1 4 -1 -1 5 6 -1 -1 7 -1 -1.

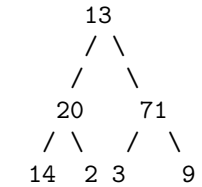
2. Qual deverá ser a sequência de dados para obter a seguinte árvore?



3. Complete o código para fazer o *dump* da árvore com as estratégias in-fixada e pós-fixada. Apresente a árvore escrevendo a sequência de nós separados por linhas e espaços correspondentes ao nível.

4. Complete o código para fazer o varrimento da árvore em profundidade (*sweep depth*) e em largura (*sweep breadth*). Apresente a árvore escrevendo a sequência de nós separados por linhas e espaços correspondentes ao nível, como no exercício anterior.

   Por exemplo, a árvore



   será representada, de acordo com o varrimento em profundidade, por

```
13
    20
        14
        2
    71
        3
        9
```

e de acordo com o varrimento em largura por

```
13
    20
    71
        14
        2
        3
        9
```

Para o varrimento da árvore em largura pode não ser vantajoso definir uma função recursiva, como foi feito para a estratégia pré-fixada. No ficheiro queue.h encontram-se assinaturas de funções que poderão ser úteis para a implementação do varrimento em largura.

5. Complete o código de modo a que o programa indique no écran se a árvore fornecida se encontra ordenada (função `isTreeOrdered()`).

6. Complete o código de modo a que o programa indique no écran se a árvore fornecida se encontra balanceada (função `isTreeBalanced()`).

7. Na directoria de trabalho referente a este laboratório encontrará 4 ficheiros de teste (teste01.txt a teste04.txt). Para cada um dos ficheiros, indique se a árvore respectiva se encontra ordenada e/ou balanceada.

8. **Pergunta surpresa**
   Haverá uma pergunta surpresa para cada grupo, relacionada com os algoritmos implementados, o código, etc. A pergunta surpresa poderá envolver adicionar ou alterar código, efetuar uma análise experimental adicional ou apenas uma análise teórica.