Ficha 10

Programação Imperativa

Árvores binárias de procura

Use, se achar necessário, o projecto `https://codeboard.io/projects/244195` para responder aos problemas propostos.
Considere o seguinte tipo para representar árvores binárias de inteiros.

```
typedef struct nodo {
    int valor;
    struct nodo *esq, *dir;
} * ABin;
```

1. Uma forma de remover um elemento de uma árvore binária de procura sem aumentar a altura da árvore consiste em substituir o nodo onde esse elemento se encontra pelo menor elemento que se encontra do seu lado direito.

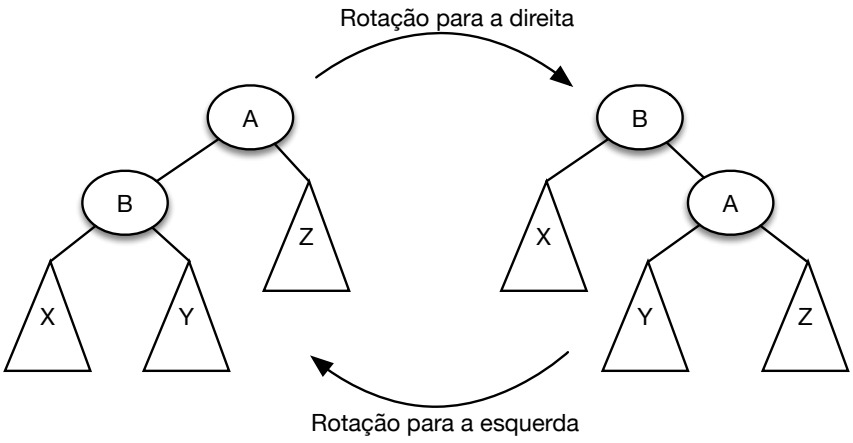
   Apresente definições das seguintes funções sobre árvores binárias de procura.

   (a) `ABin removeMenor (ABin *a)` que remove o nodo mais à esquerda de uma árvore (retornando esse nodo).

   (b) `void removeRaiz (ABin *a)` que remove a raiz de uma árvore não vazia (libertando o correspondente espaço).

   (c) `int removeElem (ABin *a, int x)` que remove um elemento de uma árvore binária, libertando o espaço correspondente. A função deverá retornar 0 se o elemento existia na árvore.

2. Considere as seguintes definições que *rodam* uma árvore. Note que ambas as operações preservam a ordem dos elementos (i.e., se forem efectuadas sobre uma árvore de procura, o resultado continua a ser uma árvore de procura).



```
void rodaEsquerda (ABin *a){
  ABin b = (*a)->dir;
  (*a)->dir = b->esq;
  b->esq = (*a);
  *a = b;
}
```

```
void rodaDireita (ABin *a){
    ABin b = (*a)->esq;
    (*a)->esq = b->dir;
    b->dir = *a;
    *a = b;
}
```

Note ainda que ao efectuar uma destas rotações, o elemento que está na raiz passa para o nivel 1 enquanto que um dos elementos que está no nível 1 passa para o nível 0. Neste caso dizemos que este último elemento foi promovido.

Usando estas funções defina as seguintes operações sobre árvores binárias de procura.

(d) `void promoveMenor (ABin *a)` que promove o menor elemento de uma árvore para o nível 0. A árvore resultante não deve aumentar a altura da árvore em mais do que uma unidade.

(e) `void promoveMaior (ABin *a)` que promove o maior elemento de uma árvore para o nível 0. A árvore resultante não deve aumentar a altura da árvore em mais do que uma unidade.

(f) Apresente uma definição alternativa da função `removeMenor` descrita na questão 1.

3. Uma árvore diz-se equilibrada sse, em cada nodo, o número de nodos à esquerda e à direita não difere em mais do que uma unidade.

   Uma forma de equilibrar uma árvore consiste em começar por a transformar numa *espinha* (i.e., uma árvore em que todos os nodos têm a sub-árvore da esquerda vazios) e depois equilibrar essa árvore.

   Nas funções que se descrevem abaixo **não deve ser feita qualquer alocação de memória**; deve-se reorganizar os nodos da árvore de forma a obter o resultado pretendido

   (g) Defina uma função `int constroiEspinha (ABin *a)` que transforma a árvore `*a` numa espinha. A função deve retornar o número de nodos da árvore.

   Sugestão: de forma a tornar esta função mais eficiente, comece por definir uma função `int constroiEspinhaAux (ABin *a, ABin *ult)` que também coloca e `*ult` o endereço do nodo mais à direita da árvore produzida.

   (h) Defina uma função `ABin equilibraEspinha (ABin *a, int n)` que recebe uma espinha `*a` e um número `n` e produz uma árvore equilibrada com esses nodos. Em `*a` fica a árvore construída e é retornado o endereço dos elementos da espinha que não foram utilizados.

   (i) Usando as funções anteriores, defina uma função `void equilibra (ABin *a)` que equilibra uma árvore.