MC202 - Turmas EFGHI
Lista de Exercícios 3

1. Considere um heap binário armazenado nas posições $H[0..n-1]$ de um vetor H. Cada elemento $H[j]$ do vetor é um ponteiro para um registro que contém, entre outros, um campo *prioridade* $H[j]\uparrow.pr$ (um número real); sendo $H[0]$ o elemento de menor prioridade.

   Suponha que é às vezes necessário retirar ou alterar a prioridade de um elemento d que já está no meio do *heap*. Para evitar ter que procurá-lo no *heap* todo, acrescentamos a seu registro um campo adicional $d\uparrow.pos$ que dá sua posição corrente no *heap*. Ou seja, $H[i]\uparrow.pos = i$ para todo i em $\{0..n-1\}$. Note porém que as operações de inserção e remoção geralmente movem vários elementos de lugar no *heap*, e portanto precisam atualizar os campos *pos* dos mesmos.

   (a) Escreva um procedimento $Insere(d, \mathbf{var}H, \mathbf{var}n)$ que tem esse cuidado.

   (b) Escreva um procedimento eficiente $Remove(d, \mathbf{var}H, \mathbf{var}n)$ que retira o elemento d do *heap*, qualquer que seja sua posição neste.

   (c) Escreva um procedimento eficiente $Altera(d, p, \mathbf{var}H, \mathbf{var}n)$ que muda a prioridade do dado d para o valor p, e rearranja os elementos do *heap* H se necessário (e *apenas* se necessário).

2. Suponha que cada nó p de uma árvore de busca binária tem os apontadores $p\uparrow.chave$, $p\uparrow.esq$ e $p\uparrow.dir$ tradicionais, com nós em ordem crescente da esquerda para a direita. Suponha que cada nó possui também um campo $p\uparrow.prox$, inicialmente indefinido. Escreva um procedimento que, dado um ponteiro A para a raiz da árvore, atribui todos os apontadores *prox* de modo a formar uma lista linear com todos os nós da árvore, em ordem *crescente* de chave. O procedimento deve devolver o endereço L do primeiro nó da lista. Isto é, $L\uparrow$ deve ser elemento de menor chave na árvore, $L\uparrow.prox\uparrow$ deve ser o segundo menor, e assim por diante.

3. Suponha que um arquivo de dados contém um número conhecido N de linhas, cada qual consistindo de uma chave *ch* e um dado d associado à mesma; e que esse arquivo já está em ordem crescente de chave. Escreva um procedimento que lê sequencialmente esse arquivo e constrói uma árvore binária de busca plena para esses elementos (onde todo nível k, exceto possivelmente o último, tem 2^k elementos). O procedimento deve usar no máximo $O(N)$ operações, e no máximo $O(\log N)$ espaço adicional. *Dica*: use um vetor para guardar as partes da árvore que podem ser construídas com os primeiros k elementos do arquivo, para $0 \leq k \leq N$.

4. Se precisamos percorrer uma lista simplesmente ligada na ordem "errada" (do último elemento para o primeiro), podemos primeiro inverter a lista, imprimir, e inverter de novo. Procure adaptar esta idéia para percorrer todos os nós de uma árvore de busca binária, sem usar recursão, pilha explícita, ou vetores auxiliares.