Universidade Federal do Ceará
Campus Crateús
Professor: Bruno C. H. Silva

Nota:
Disciplina: Estruturas de Dados
Turma: _____

Aluno: _____

Prova Final

Duração: 6 horas Data: / /

Esta prova contém 3 página(s), incluindo esta capa, e 3 questões, formando um total de 10 pontos.

Tabela (para uso EXCLUSIVO do professor)

Questão:	1	2	3	Total
Valor:	5	2.5	2.5	10
Pontuação:				

1. (5 pontos) Uma Árvore Binária de Busca (ABB) possuí as seguintes propriedades: seja $S = \{s_1, ..., s_n\}$ o conjunto de valores dos nós v_i de uma árvore T, esse conjunto satisfaz $\{s_1 < ... < s_n\}$, e, a cada nó $v_i \in T$ está associado um valor distinto $s_i \in S$. Seja v um nó de T, se $v_i \in T$ está a esquerda de v, então $v_i.s_i < v.s$. Se $v_i \in T$ está a direita de v, então $v_i.s_i > v.s$. Em outras palavras, os nós pertencentes à sub-árvore esquerda possuem valores inferiores do que o valor associado ao nó-raiz r. Já os nós nós pertencentes à sub-árvore direita possuem valores maiores do que o valor associado ao nó-raiz r. Você deve então implementar um TAD de Árvore Binária de Busca e testá-lo em um função *main*. A lógica das funções *create*, *clear*, *printAllPos*, *printAllPre*, *printAllIn* para uma Árvore Binária de Busca é a mesma da Árvore Binária implementada no último trabalho. Porém a lógica da inserção (*add*) e pesquisa (*find*) é diferente. O projeto de implementação destas duas funções é explicado como se segue:

   1.) Lógica da função de inserção:
   - Procure um local para inserir o novo nó, começando a procura a partir do nó-raiz;
   - Para cada nó-raiz de uma sub-árvore, compare: se o novo nó possui um valor menor do que o valor no nó-raiz (vai para sub-árvore esquerda), ou se o valor é maior que o valor no nó-raiz (vai para sub-árvore direita);
   - Se um ponteiro (filho esquerdo/direito de um nó-raiz) nulo é atingido, coloque o novo nó como sendo filho do nó-raiz.

   2.) Lógica da função de pesquisa:
   - Comece a busca a partir do nó-raiz;

- Para cada nó-raiz de uma sub-árvore compare: se o valor procurado é menor que o valor no nó-raiz (continua pela sub-árvore esquerda), ou se o valor é maior que o valor no nó-raiz (sub-árvore direita);
- Caso o nó contendo a chave pesquisada seja encontrado, retorne o nó pesquisado, caso contrário retorne *NULL*.

Para entender o algoritmo de inserção considere a seguinte sequência de inserções $\{17, 99, 13, 1, 3, 100, 400\}$. Teríamos a seguinte composição:

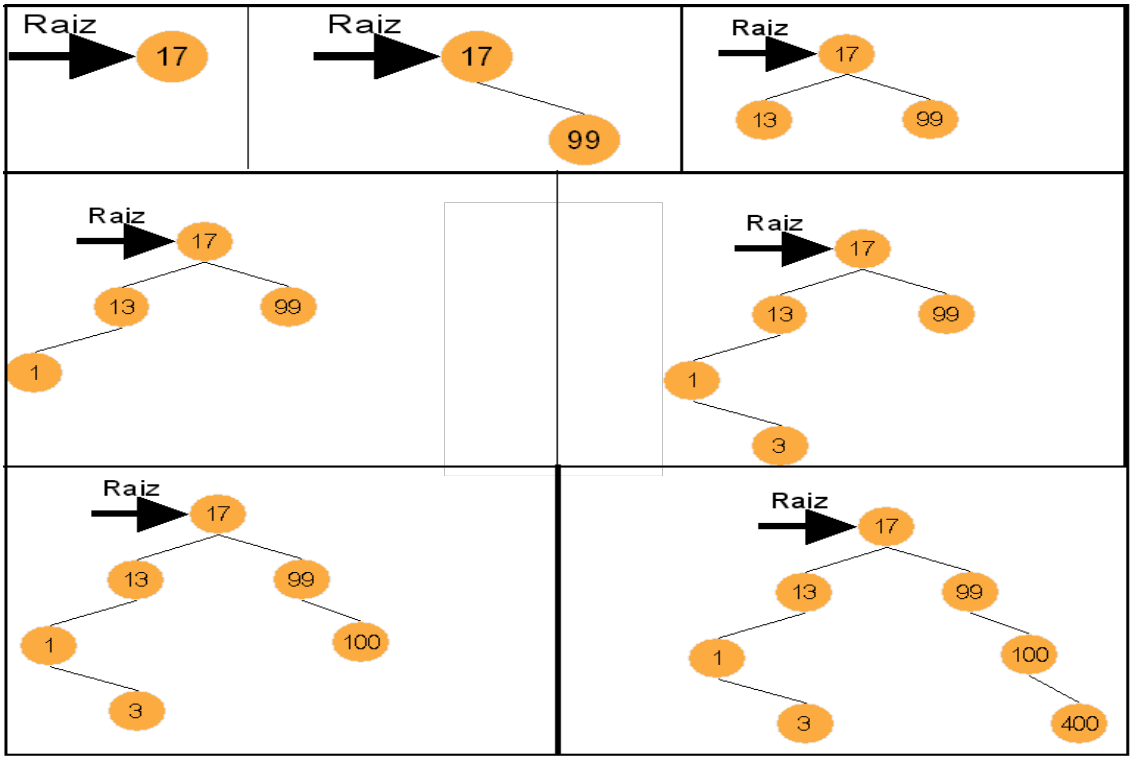

Figura 1: Ilustração da operação de inserção em uma ABB.

Você deve explicar a complexidade de cada função que implementar. A descrição da complexidade de cada função deve ter pelo menos 3 linhas. Todas as respostas devem ser fornecidas em um arquivo PDF. Inclusive o código fonte.

2. (2.5 pontos) Considere um sistema de validação de compras de uma rede de lojas. Cada compra tem: um código (número inteiro); e, um valor (float). O usuário deve cadastrar cada compra efetuada da rede de lojas, para, posteriormente, validá-las. Cada compra é cadastrada no repositório de compras $Y = \{y_0, ..., y_k, ..., y_n\}$. As compras são ordenadas no repositório conforme o seu valor, isto é, temos que $Y = \{valor(y_0) < ... < y_k < ... < y_n\}$. Y pode abstraído como uma estrutura de dados que utiliza um vetor para armazenar as compras y de limite n. k funciona como um contador de inserções e específica em que posição do vetor dados a próxima compra será cadastrada. Quando um repositório é criado, $k = 0$ e o arranjo de dados que registra as compras tem dimensão n. Cada nova compra cadastrada em Y deve ir para a posição no vetor de dados de tal forma que o mesmo se mantenha ordenado. Quando o usuário decide validar uma compra, a função *tratar* deve retornar a compra mais cara, i. e., aquela que está na

última posição do vetor de dados, tendo em vista a ordenação deste vetor. A compra retornada pela função *tratar* deve ser removida do vetor de dados. Dada esta descrição, identifique o projeto de implementação de uma das estruturas de dados estudadas que melhor se adequá a este contexto e implemente o repositório de compras conforme a estrutura de dados que você identificou. As operações esperadas são:

- *criarRepositorio*: recebe n e cria um o repositório de compras de dimensão n;
- *adicionar*: adiciona um elemento no arranjo de dados do repositório de compras de tal forma que o mesmo se mantenha ordenado. A ordem é crescente. Esta operação retorna verdadeiro se foi possível adicionar a compra, isto é, se o repositório já não estava cheio;
- *tratar*: retorna a compra mais cara, i. e., aquela com o maior valor. Retira a compra do repositório;
- *tamanho*: retorna a quantidade de elementos inseridos no repositório;
- *pesquisarValor*: recebe um valor de compra v e retorna a posição da compra com o valor igual a v no repositório. Esta função deve ter complexidade $O(\log n)$. Retorna -1 se o elemento não foi encontrado;
- *pesquisarCodigo*: recebe um código de compra i e retorna a posição da compra com código i. Caso não exista uma compra com código igual a i no repositório, retorna -1;
- *estaCheio*: retorna verdadeiro se o repositório está cheio e falso para o caso oposto;
- *estaVazio*: retorna verdadeiro se o repositório está vazio e falso para o caso oposto;
- *esvaziar*: remove todas as compras do repositório.

Descreva com a suas palavras a complexidade das suas funções.

3. (2.5 pontos) Na questão dois dessa prova, temos um exemplo de aplicação que necessita do emprego de uma estrutura de dados. Agora, você deve identificar e descrever um enunciado de aplicação que necessite o desenvolvimento de um programa para operacionalizá-lo. Este programa deve fazer o uso de alguma das estruturas de dados estudadas até então. Justifique o uso dessa estrutura de dados e implemente o programa. O seu exemplo deve ser original e descrito com as suas palavras. Comente a complexidade das funções que você implementar para a sua estrutura de dados. Seu exemplo deve ser original. Portanto o professor irá checar se o exemplo já não foi exposto na internet, material didático ou fornecido por um colega que cursou a disciplina.