

IMD0029 - Estrutura de Dados Básicas 1 – 2018.1 – Prova 04

Prof. Eiji Adachi M. Barbosa

Nome: _____

Matrícula: _____

ANTES DE COMEÇAR A PROVA, leia atentamente as seguintes instruções:

- Esta é uma prova escrita de caráter individual e sem consultas a pessoas ou material (impresso ou eletrônico).
- A prova vale 10, pontos e o valor de cada questão é informado no seu enunciado.
- **Preze por respostas legíveis, bem organizadas e simples.**
- As respostas devem ser fornecidas preferencialmente em caneta. Respostas fornecidas a lápis serão aceitas, mas eventuais questionamentos sobre a correção não serão aceitos.
- Celulares e outros dispositivos eletrônicos devem permanecer desligados durante toda a prova.
- Desvios éticos ou de honestidade levarão à anulação da prova do candidato (nota igual a zero).

Questão 1 (2,0 pontos): Sequências bitônicas inversas são aquelas que possuem duas sequências, sendo uma sequência inicial decrescente, seguida de uma sequência crescente. Ou seja, os elementos de uma sequência bitônica obedecem a seguinte relação:

$$A_0 > A_1 > \dots > A_{i-1} > A_i < A_{i+1} < \dots < A_n$$

Considere que um array bitônico-inverso é um array de inteiros sem repetições cujos elementos representam uma sequência bitônica inversa. Neste contexto, implemente uma função que recebe como entrada um array bitônico-inverso e retorna o índice do elemento da “base”. O elemento da “base” é o último elemento da sequência inicial decrescente e o primeiro elemento da sequência final crescente, ou seja, é o elemento A_i da relação acima. Sua função deverá obrigatoriamente ser **recursiva**, ter complexidade $\Theta(\lg(n))$ e seguir a assinatura:

```
int findBase(int a[], int arraySize)
```

Obs.: Nesta questão, não podem ser usadas instruções para realizar repetição, como for, while e do-while. Ou seja, você não poderá usar instruções de repetição; você deverá construir sua solução apenas com chamadas recursivas.

Questão 2 (2,0 pontos): Suponha que você tem à sua disposição uma Pilha genérica (ou parametrizada) já implementada corretamente, provendo as seguintes operações:

- void Stack<Element>::Pop()
- Element Stack<Element>::Top()
- Bool Stack<Element>::Push(Element)
- Int Stack<Element>::Size()

Suponha também que você possui uma lista simplesmente encadeada implementada seguindo a estrutura abaixo:

<pre>List{ Node* first; }</pre>	<pre>Node { Node* next; int value; }</pre>
-------------------------------------	--

Neste contexto, use a Pilha para inverter uma instância desta lista simplesmente encadeada. Sua solução deverá seguir a assinatura `void List::Revert()`.

Obs.: A solução da questão 2 deverá única e exclusivamente operar sobre a lista usando uma pilha auxiliar. A solução não poderá criar outras listas auxiliares, nem outras estrutura auxiliares que não seja a pilha (ex.: array, ou outras listas); ponteiros auxiliares para manipular os nós da lista podem ser usados.

Questão 3 (2,0 pontos): Considere a mesma estrutura de lista simplesmente encadeada apresentada na questão 2. Para essa estrutura de lista, implemente nesta questão a função `int List::sum()`, a qual itera recursivamente sobre a lista, calcula e retorna a soma total dos valores armazenados em cada nó da lista. Esta solução deverá ser obrigatoriamente recursiva.

Questão 4 (2,0 pontos): Considere uma tabela de dispersão que resolve colisões pelo método do endereçamento aberto com sondagem linear e segue a estrutura abaixo:

<pre>HashTable{ HashEntry** data; Int size; Int quantity; }</pre>	<pre>HashEntry { K key; V value; }</pre>
---	--

Nesta questão, implemente o método de inserção desta tabela. Sua implementação deve seguir a assinatura `void HashTable::Put(K key, V value)`.

Obs.: Nesta questão, você pode assumir que a função hash já está implementada e pode ser reusada. Não é necessário se preocupar com o redimensionamento dinâmico da tabela.

Questão 5 (2,0 pontos): Para cada uma das afirmações a seguir, marque V (verdadeiro) ou F (falso), **justificando sucintamente** sua resposta (mesmo as verdadeiras devem ser justificadas). Marcações de V ou F **sem justificativas não serão aceitas**.

1. () O algoritmo *Merge Sort* possui complexidade assintótica $\Theta(n \cdot \log_2(n))$ para o melhor caso e $\Theta(n^2)$ para o pior caso.
2. () Os algoritmos *Insertion Sort*, *Bubble Sort* e *Selection Sort* possuem a mesma complexidade assintótica para o melhor caso.
3. () O pior caso do *Quick Sort* é quando nas sucessivas chamadas recursivas se divide um problema de tamanho N em dois sub-problemas de tamanhos bastante similares.
4. () Os algoritmos de busca sequencial e de binária podem ser empregados nos mesmos tipos de array.
5. () As operações de inserir e remover elementos de uma Pilha só podem ser realizadas em tempo $\Theta(1)$ quando implementamos a Pilha usando listas encadeadas, não sendo possível realiza-las em tempo $\Theta(1)$ usando uma implementação baseada em arrays.
6. () As operações de inserir e remover elementos de uma Fila só podem ser realizadas em tempo $\Theta(1)$ quando implementamos a Fila usando listas encadeadas, não sendo possível realiza-las em tempo $\Theta(1)$ usando uma implementação baseada em arrays.
7. () Se tivermos um Deque à nossa disposição, é possível usá-lo como uma Pilha, mas não é possível usá-lo como uma Fila.
8. () De posse de uma lista simplesmente encadeada com ponteiros para início e fim, é possível criarmos uma estrutura que segue a estratégia F.I.L.O. (first-in last-out) da seguinte forma: ao realizarmos uma *inserção* nós inserimos no início da lista e ao realizarmos uma *remoção* nós removemos do fim da lista.
9. () De posse de uma lista simplesmente encadeada com ponteiros para início e fim, é possível criarmos uma estrutura que segue a estratégia F.I.F.O. (first-in first-out) da seguinte forma: ao realizarmos uma *inserção* nós inserimos no início da lista e ao realizarmos uma *remoção* nós removemos do fim da lista.
10. () As operações de inserir no início e no fim são realizadas em tempo constante apenas em listas duplamente encadeada com sentinelas cabeça e cauda, não sendo possível realiza-las em tempo constante em listas simplesmente encadeadas.