Exercícios de Revisão AEDS II

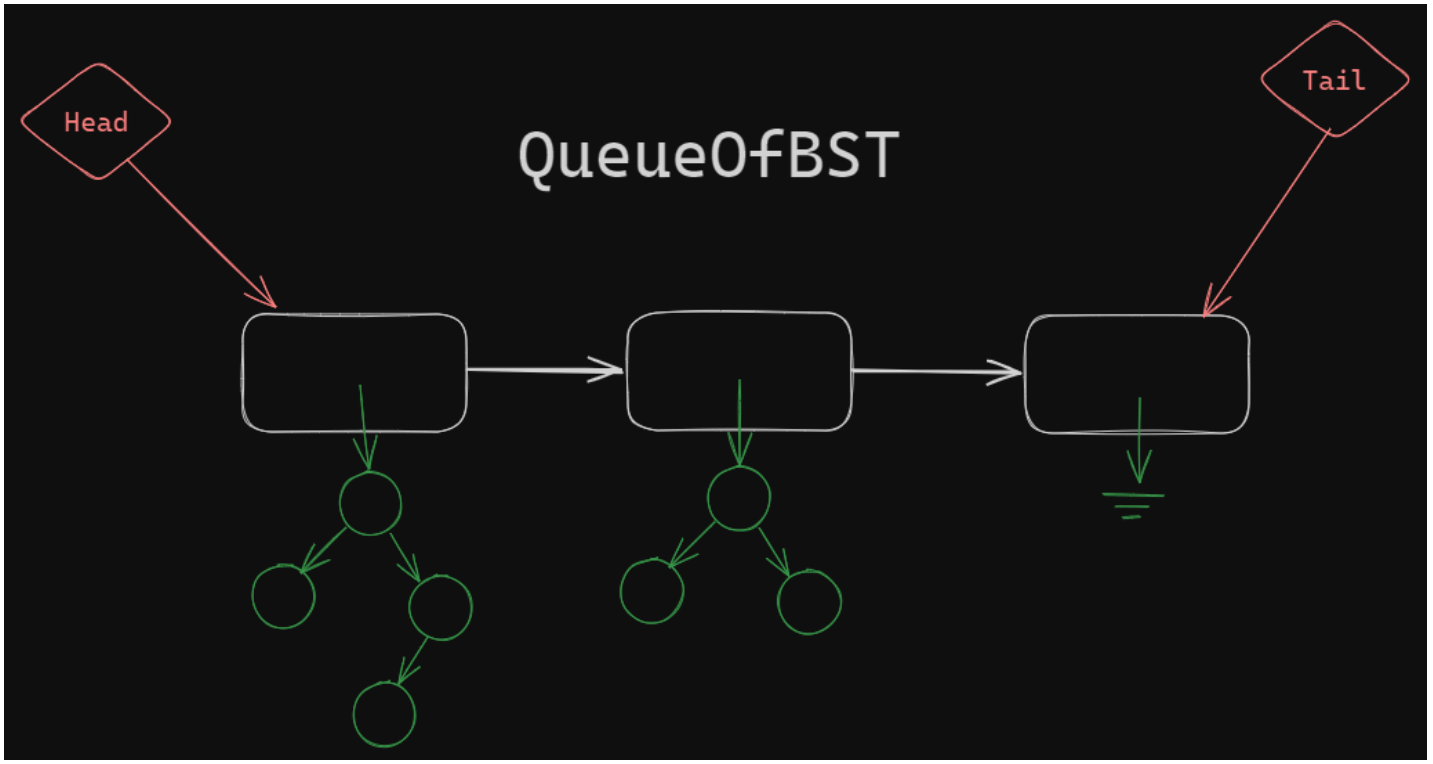
2024/02 - Prova 2

1. **[Linked List]**. Implemente um algoritmo para encontrar o nó do meio de uma lista simplesmente encadeada de tamanho desconhecido sem contar o número total de nós da lista. A complexidade de espaço deve ser $O(1)$ e a assinatura da função: $Node\ \ findMidlle(Node\ head)$

2. **[Linked List]**. Dada uma lista simplesmente encadeada, implemente um algoritmo que a inverta completamente, retornando um ponteiro para o novo primeiro nó da lista. Almeja-se que o algoritmo seja executado em tempo $O(n)$ e com complexidade de espaço $O(1)$. O método deve seguir a seguinte assinatura: $Node\ \ reverse(Node\ head)$

3. **[Linked List + Merge Sort]**. Explique 2 motivos que tornam o Merge Sort em memória primária mais eficiente para listas ligadas do que arrays.

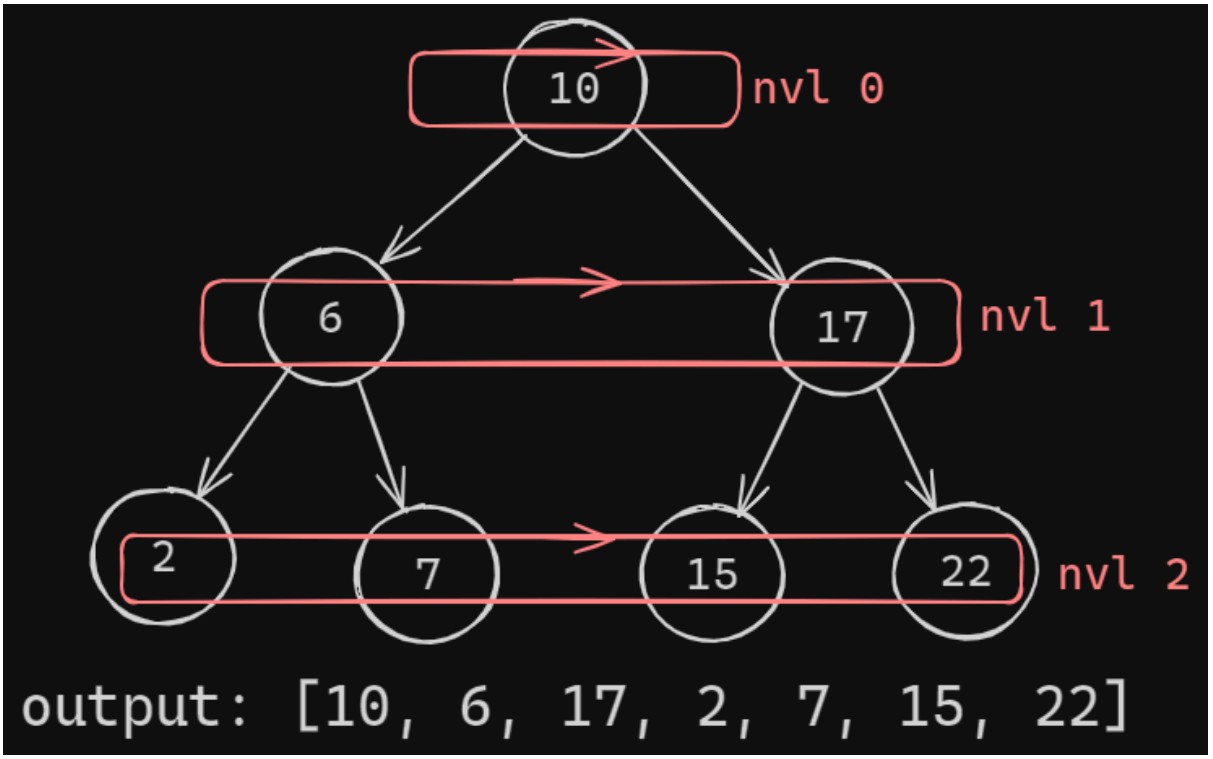
4. **[BST + Queue]**. Conforme mostra a Figura, temos a estrutura de dados formada pela junção de Fila Flexível e uma Árvore Binária de Pesquisa. Nessa estrutura chamada $QueueOfBST$, um ponteiro deve referenciar o primeiro nó da fila e outro o último, sendo respectivamente, $QueueNode\ head$ e $QueueNode\ tail$. Além disso, a classe $QueueNode$ é formado por 2 ponteiros: $QueueNode\ next$ e $BSTNode\ root$. A classe $BSTNode$ possui um elemento do tipo inteiro e dois outras referências: $BSTNode\ left$ e $BSTNode$ right.



a) Implemente um algoritmo para adicionar elementos na $QueueOfBST$ e apresente sua complexidade de tempo, levando em consideração que as células da estrutura podem não terem sido instanciadas. O método deve seguir a seguinte assinatura: $boolean\ add(int\ element)$.

b) Implemente um algoritmo que retorne o maior elemento presente na $QueueOfBST$ e apresente sua complexidade de tempo. O método deve seguir a seguinte assinatura: $int\ getMaxValue()$.

5. **[BST]**. Faça um algoritmo que percorre uma Árvore Binária de Busca por nível.



6. **[QuickSort]**. Explique um motivo que faz com que o QuickSort seja ineficiente para arrays pequenos (ex: 10 elementos ou menos). Em seguida, implemente uma versão híbrida que use Quicksort até certo ponto (threshold) e depois finalize com outro algoritmo de ordenação eficiente para arrays pequenos.

7. **[Stack]**. Desenvolva uma pilha especial $MinStack$ que, além de operações padrão ($IsEmpty$, $peek$, $push$, pop), possui uma função $getMin$ que retorna o valor mínimo da pilha em tempo constante $O(1)$. Implemente essa classe $MinStack$ como flexível e elabore uma solução para o método $getMin()$.