Algoritmos e Estruturas de Dados I Professora Isabel Harb Manssour Exercícios de Revisão

1. Considere as estruturas de dados que estudamos: ***Lista***, ***Fila***, ***Pilha***, ***Árvore genérica***, e ***ABP balanceada***. Para cada uma das seguintes situações, sugira qual delas é mais adequada.
   * A *Tabajara Soft* (TS) deseja manter os registros dos funcionários ordenados pelo nome e de forma que seja possível fazer uma busca eficiente pelo nome. ABP balanceada
   * A TS quer transformar os documentos XML que contêm informações sobre seus produtos em uma estrutura de dados que possa ser consultada. Árvore Genérica
   * A TS deseja armazenar apenas os CPFs dos seus empregados para que a cada mês um empregado seja escolhido aleatoriamente para recebe um brinde. Lista
   * A TS está implementando um sistema para consulta e alteração de dados dos seus funcionários e gostaria

de adicionar a funcionalidade de “desfazer/refazer” neste sistema. Pilha

* + A TS adquiriu ingressos para a temporada inteira de bocha, para serem compartilhados pelos empregados. Crie uma lista de espera para esse esporte popular. Fila
* A Árvore Genérica permite representar a hierarquia, como em pastas
* A Pilha permitiria armazenar as ações realizadas pelo usuário, como consultas e alterações, e desfazer ou refazer essas ações conforme necessário
* A Fila permite adicionar elementos no final e remover elementos do início,
* A ABP balanceada mantém os elementos ordenados e permite pesquisas eficientes em tempo logarítmico.
* Se for necessário um controle mais flexível sobre os elementos na lista de espera, como alterar a ordem de prioridade, remover um nome específico ou obter informações sobre um determinado empregado na lista, a estrutura de dados Lista pode ser mais útil nesse contexto.

1. Considerando a árvore binária ao lado, responda:

* Qual é o grau (Quantas subárvores ele tem) do nó “/” ? 2
* Qual é o nível(Num. De linhas que liga a raíz) do nó “F” ? 3
* Qual é a altura(nível mais alto) da árvore ? 4
* Quais são os nodos folha(sem subárvores) ? A, B, C, G, F, D, E

+

+ -

A B / \*

C G F -

* Realize o caminhamento pós-fixado(esquerda, direita, raiz): A, B, +, C, G, /, F, D, E, -, \*, -, +

D E

* Realize o caminhamento pré-fixado(raiz, esquerda, direita): +, +, A, B, -, /, C, G, \*, F, -, D, E
* Realize o caminhamento central(esquerda, raiz, direita): A, +, B, +, C, /, G, -, F, \*, D, -, E

Uma aresta de uma árvore T é um par de nodos (u,v) tal que u é pai de v, ou vice-versa

Um caminho de T é uma sequência de nodos tais que quaisquer dois nodos consecutivos da sequência formam uma aresta

1. Suponha que um conjunto de *strings* seja inserido em uma árvore binária de pesquisa que armazena *strings* na seguinte ordem: Eva, Adão, Romeu, Julieta, Tomas, Diego, João. Crie um desenho que mostre a árvore resultante depois de cada inserção. DONE
2. Suponha que os seguintes números sejam inseridos em uma árvore binária de pesquisa (ABP), nesta ordem: 50, 20, 40, 70, 60, 80, 90, 10, 30, 15, 45, 55, 65, 25, 5, 35, 85, 75, 95. Apresente um desenho que mostre como fica a árvore resultante. Supondo que se quer saber se o número 92 está ou não armazenado nesta árvore, apresente também a sequência de números que são comparados até descobrir que o número não está armazenado na árvore. DONE
3. Desenhe uma árvore binária T que satisfaça simultaneamente o seguinte:

* Cada nodo de T armazena um único caractere;
* Um caminhamento pré-ordem resulta em: EXAMFUN;
* Um caminhamento central resulta em: MAFXUEN.

Diagrama

Descrição gerada automaticamente

1. Apresente o algoritmo de um método para a Árvore Binária estudada em aula que conta o número de folhas da árvore.

Int countLeaves {

return countLeaves(root);

}

Int countLeaves(Node n) {

If (n==null){

Return 0;  
}

If(n.left == null && n.right == null) {

Return 1;

}

Return countLeaves(n.left) + countLeaves(n.right);

}

1. Apresente o algoritmo de um método para a Árvore Binária estudada em aula que conta o total de números pares armazenados na árvore. Não podem ser chamados os métodos de caminhamento já implementados, tal como o método *positionsPos*. Texto

   Descrição gerada automaticamente
2. Apresente o algoritmo de um método para a Árvore Genérica estudada em aula que conta o número de folhas da árvore.

Public int countLeaves() {  
 return countLeaves(root);

}

Private int countLeaves(Node n) {

If (n == null) {

Return 0;

}

If(n.right == null && n.left == null) {

Return 1;

}

return countLeaves(n.right) + countLeaves(n.left);

}

1. Desenhe a representação de uma árvore binária para a seguinte expressão aritmética: (((5+2) ∗ (2−1))/((2+9)+((7−2)−1)) ∗8). DONE
2. Apresente o algoritmo de um método para a árvore binária estudada em aula (*BinaryTreeOfInteger*) que calcula e retorna a média dos valores armazenados na árvore, sabendo que não podem ser chamados os métodos de caminhamento já implementados, tal como o método *positionsPos*.

Texto

Descrição gerada automaticamente

1. Apresente o algoritmo de um método (privado) para a árvore genérica estudada em aula que retorna o número total de nodos de uma sub-árvore cuja raiz é passada por parâmetro.

Texto

Descrição gerada automaticamente

1. Explique por que o algoritmo *insertion-sort* é considerável estável.

Porque na ordenação ele não altera a posição relativa dos elementos que têm mesmo valor.

Um algoritmo de ordenação é estável se não altera a posição relativa dos elementos que têm o mesmo valor

Algoritmos estáveis: Bubble-sort, Insertion-sort, Merge-sort

Algoritmos instáveis Quick-sort

1. Explique em qual situação o algoritmo *insertion-sort* possui um melhor desempenho?

Em uma situação em que os elementos já estão ordenados.

1. Qual é a notação O do algoritmo *quick-sort*? Justifique e diga se este algoritmo é considerável estável.

O(n log n). Não é estável, pois seleciona qualquer elemento para ser pivô e isso pode acarretar em trocar de ordem elementos que têm mesmo valor, nesse caso inclusive, podendo se tornar O(n)2

Comparação quanto ao comportamento

◼ Bubble-sort: O(n2 )

◼ Insertion-sort: O(n2 )

◼ Merge-sort: O(n log n)

◼ Quick-sort: O(n log n)

1. Dada a sequência de números: [32, 18, 9, 26, 17, 5, 2, 14], ordene-a em ordem crescente segundo os seguintes algoritmos, apresentando a sequência obtida após cada passo do algoritmo: *merge-sort* e *quick-sort* (considere que o pivô é o elemento do meio do vetor).

Merge-sort divide ao meio sempre

Quick-sorte sempre tem pivô, quando não está no enunciado, pegar um aleatório

1. Escreva um método boolean check(int []vet) que verifica se o vetor passado por parâmetro está ordenado ou não.

Private boolean check(int[] vet) {

int aux = 0;  
 for(int i = 0; i<vet.length - 1; i++) {

if(vet[i] > vet[i + 1]) {

return false;

}

}

Returne true;