Lista 2 - PMR2300

Fabio G. Cozman

24 de maio de 2013

Alguns exercícios a seguir lista tem implementações em software, com testes, nos arquivos em lista2.zip: Deque.java, ListaLigada.java, Verifique.java, ArvoreBinariaBusca.

1. Escreva uma classe chamada `Complexo` que opera sobre números complexos. O construtor básico deve ser:
   `Complexo( double a, double b ).`
   Um segundo construtor deve ser providenciado para criar o número real `a`:
   `Complexo( double a ).`
   Codifique os métodos `add(Complexo c)`, `subtract(Complexo c)`, `multiply(Complexo c)` e `divide(Complexo c)`.

2. Considere que você comprou uma biblioteca contendo a classe `Complexo`, cujo manual indica:

   ```
   public class Complexo:
           public Complexo(double a, double b);
           public Complexo add(Complexo c);
           public Complexo subtract(Complexo c);
           public Complexo multiply(Complexo c);
           public Complexo divide(Complexo c);
   ```

   Você deseja criar uma classe como `Complexo`, porém com métodos adicionais que permitem somar, subtrair, dividir e multiplicar por um número real:

   ```
           public Complexo add(double d);
           public Complexo subtract(double d);
           public Complexo multiply(double d);
           public Complexo divide(double d);
   ```

   Codifique uma classe `SuperComplexo` que contém toda a funcionalidade indicada anteriormente usando o esquema de hierarquias de Java.

3. Considere um programa que contenha um `package aeroporto`. Temos a classe `Pessoa`, com atributo `int RG`, e a classe `Passageiro`, que é subclasse de `Pessoa` e tem atributo `int ID` e método `public void access()`. Temos também a classe `Rota`, com atributos `private int ID` e `Aeroporto aeroporto`. A classe `Aeroporto` tem atributo `String nome`. Finalmente, a classe `Reserva` tem atributos `double valor` e `Passageiro pass`. Escreva código para todas as classes do pacote; não é necessário escrever código no corpo das funções (deixe em branco).

   DICA: O código para a classe Passageiro é

```
package aeroporto;
public class Passageiro extends Pessoa {
    int ID;
    public void access() {
    }
}
```

4. Considere um `package` denominado `escola`. Temos as seguintes classes. Classe `Pessoa` tem atributos `public String nome` e `private int ID`. Classe `Aluno` é subclasse de `Pessoa` e tem atributos `private int numero`. Classe `Area` tem atributo `protected int codigo`. Classe `Habilitacao` é subclasse de `Area` e tem atributo `Aluno aluno`.

   (a) Codifique uma implementação simples para cada classe. Se houver membros com acesso não indicado no diagrama de classes, assuma acesso público para esses membros.

   (b) É possível acessar:

   - `codigo` em `Area` a partir de `Pessoa`? De `Aluno`? De `Habilitacao`? De classes em outro pacote?
   - `nome` em `Pessoa` a partir de `Pessoa`? De `Aluno`? De `Habilitacao`? De classes em outro pacote?

5. Considere as seguintes classes:

   - `Tree` (contém uma variável `age` que é `protected`)
   - `Deciduous extends Tree`
   - `Evergreen extends Tree`
   - `Pine extends Evergreen`
   - `Forest`

   Suponha que todas as classes estão no mesmo `package`. Desenhe um diagrama de classes simplificado, sem indicar métodos nem variáveis. Quais classes podem acessar a variável `age` da classe `Tree`? Suponha agora que todas as classes estejam em `packages` diferentes; quais classes podem acessar a variável `age` da classe `Tree`?

6. Suponha que você está codificando um programa para controle fiscal de uma empresa. As seguintes classes estão sendo planejadas:

   - Classe Pessoa, contendo membros nome (String, public), rg (int, private) e id (int, protected).
   - Classe Cliente, sub-classe de Pessoa, contendo membro categoria (int).
   - Classe Funcionario, sub-classe de Pessoa, contendo membros posicao (int), avaliacao (double, private) e setor (int, protected).
   - Classe Mercadoria, contendo membros nome (String, public), valor (double) e codigoReceita (int, public).
   - Classe Parceiro, sub-classe de Pessoa, com membro codigo (int, protected).

As três primeiras classes compõe o pacote pessoas; as demais compõe o pacote produtos.

(a) Apresente o código que define cada uma dessas classes com seus membros. [1.0]

(b) Quais classes podem acessar: [1.0]

- Membro id de Pessoa.
- Membro nome de Mercadoria.
- Membro categoria de cliente.
- Membro avaliacao de Funcionario.

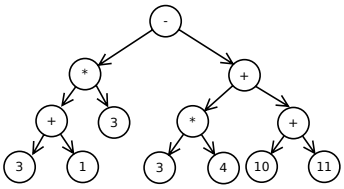
7. Descreva a saída da seguinte sequência de operações sobre uma pilha: push(5), push(3), pop(), push(2), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop().

8. Descreva a saída da seguinte sequência de operações sobre uma fila: enqueue(5), enqueue(3), dequeue(), enqueue(2), enqueue(8), dequeue(), dequeue(), enqueue(9), enqueue(1), dequeue(), enqueue(7), enqueue(6), dequeue(), dequeue().

9. Qual Ã© o custo em notação BigOh para inserção de um elemento em uma pilha com arranjo como implementado em aula, no pior caso? Justifique.

10. Considere a árvore de expressão abaixo. Nessa árvore, qual é o nó raiz? Quais são as folhas? Qual é a altura da árvore? Qual é a profundidade do nó com dado

10? Qual é a expressão representada pela árvore, em notação infixa (justifique)?



11. Considere a inserção dos seguintes números em uma árvore binária de busca: 30, 40, 24, 58, 48, 26, 11, 13. Desenhe a árvore após cada inserção.

12. Apresente um exemplo que contradiz a seguinte afirmação: a ordem de inserções em uma árvore binária de busca não importa; qualquer que seja a ordem de inserção dos elementos em uma árvore binária de busca, a árvore é a mesma.

13. Uma lista *duplamente ligada* é uma lista na qual cada nó tem uma ligação para o próximo nó e também para o nó anterior. Uma lista duplamente ligada é frequentemente referida como *deque*. Apresente o código para achar o nó do meio de um deque. Suponha as seguintes estruturas de classes:

```
class Deque {
    Node first, last;
    ... // codigo contendo os metodos da classe (a completar).
}
class Node {
    Object element;
    Node next, prev;
    ... // codigo contendo os metodos da classe (a completar).
}
```

Discussão e solução:

A idéia em um deque é que cada nó aponta para o próximo nó e para o nó anterior na lista. Percorremos um deque fazendo:

```
for (Node p = lista.first; p != null; p = p.next) {
  ...
}
```

Apenas para ilustrar o funcionamento de um deque, suponha que temos um objeto `e` e queremos achar o nó que contém esse objeto em um deque. Suponha que o deque é o objeto `lista`. Então podemos fazer:

```
for (Node p = lista.first; p != null; p = p.next) {
  if (p.element == e)
    break;
}
```

Ao fim desse processo, `p` é o nó que contém `e`.

Uma maneira bastante simple de implementar um deque é fazer o deque com "sentinelas"; ou seja, montar uma lista aonde o primeiro e o último nós não

contém nenhum dado e servem apenas para garantir que `first` e `last` nunca sejam `null`. Percorremos um deque com "sentinelas" fazendo:

```
for (Node p = lista.first.next; p != lista.last; p = p.next) {
   ...
}
```

Retornando ao exercício, considere a solução a seguir, que primeiro varre o deque contando o número de nós, e depois vai até a metade (note que estamos usando sentinelas):

```
// Encontra elemento do meio com contador.
public Object middleElement() {
  int i = 0;
  Node p, q;
  for (p = first.next; p != last; p = p.next) {
    i++;
  }
  int meio = i/2;
  i = 0;
  for (q = first.next; i < meio; q = q.next) {
     i++;
  }
  return(q.element);
}
```

É fácil resolver o exercício sem usar contadores para deques com número ímpar do nós. A solução é começar a varrer o deque do início e do final, a cada passo incrementando um lado e decrementando outro, até atingir o mesmo nó (que é o nó do meio).

```
// Encontra elemento do meio sem contador.
// Para deque com numero impar de elementos!
public Object middleElementOdd() {
     Node p = first.next;
     Node q = last.prev;
     while (p.element != q.element) {
          p = p.next;
          q = q.prev;
     }
     return(p.element);
}
```

---

14. Crie um método para concatenar duas lista ligadas e formar uma nova lista ligada.

    Discussão:

    Vamos resolver esse problema assumindo duas listas (listas normais sem "sentinelas"), tomando as estruturas:

```
class ListaLigada {
    Node first;
    ...
```

```
}
class Node {
    Object element;
    Node next;
}
```

Então:

```
    // Concatena
    static ListaLigada concatena(ListaLigada lista1,
ListaLigada lista2) {
        Node q = null;
        // Achamos o pai do ultimo no' de lista1:
        for (Node p = lista1.first; p != null; p = p.next) {
           q = p;
        }
        // Agora ligamos o ultimo no' de lista 1 com lista2:
        q.next = lista2.first;
        // Montamos a nova lista:
        ListaLigada l = new ListaLigada();
        l.first = lista1.first;
        return(l);
    }
```

O custo desse procedimento é $\mathcal{O}(n)$, onde n é o tamanho de `lista1`.

---

15. Apresente um fragmento de código para concatenar dois deques com as estruturas esquematizadas no Exercício 1.

    Discussão e solução:

    Supondo deques sem sentinelas, basta fazer:

    ```
    lista2.first.prev = lista1.last;
    lista1.last.next = lista2.first;
    Deque l = new Deque();
    l.first = lista1.first;
    l.last = lista2.last;
    ```

    Note que esse processo destrói os deques iniciais. Para fazer a mesma coisa sem destruir os deques iniciais, é preciso efetivamente criar novos nós para cada um dos nós em `lista1` e `lista2`. Ou seja, é preciso criar um deque nova e varrer os outros dois deques (primeiro `lista1` e depois `lista2`), criando um nó para cada elemento nesses deques. Faça isso como um exercício.

    Considere agora a concatenação de deques com "sentinelas". Isso complica um pouco o código; como um exercício adicional, imagine o que deve ser feito.

---

16. Codifique um método que troca a posição de dois nós `x` e `y` em uma lista ligada. O método deve receber referências aos dois nós, com a seguinte chamada:

    ```
    void troque(Node x, Node y) {...}
    ```

Discussão e solução:

Se tivéssemos que trocar apenas os conteúdos de `x` e `y`, seria fácil:

```
Object temp = x.element;
x.element = y.element;
y.element = temp;
```

No entanto temos que trocar efetivamente os nós. Para isso, temos que encontrar o pai e o filho de `x` e `y`. Vamos assumir que `x` e `y` não são `null`. Fazemos então:

```
// Troca dois nos:
void troque(Node x, Node y) {
    Node paiX = null;
    Node paiY = null;
    for (Node p = first; p != null; p = p.next) {
      if (p.next == x)
        paiX = p;
      if (p.next == y)
        paiY = p;
    }
    Node filhoX = x.next;
    Node filhoY = y.next;
    // Agora fazemos a troca:
    paiX.next = y;
    y.next = filhoX;
    paiY.next = x;
    x.next = filhoY;
}
```

---

17. Codifique um algoritmo para garantir que sequências de caracteres estejam balanceadas em relação a parênteses, colchetes e chaves. O método deve receber uma String contendo uma sequência de caracteres, e verificar o balanceamento dos parênteses, colchetes e chaves.

    Discussão:

    O problema aqui é garantir que cada parênteses (ou colchete ou chave) que seja aberto seja também fechado, e que não haja problema na ordem de abertura e fechamento. A solução mais simples é através de uma pilha. Suponha que a entrada do algoritmo seja um arranjo contendo caracteres. Assuma que uma classe `Pilha` como vista em aula esteja disponível, porém assuma que esta classe manipule diretamente caracteres (ou seja, dê `push` e `pop` em caracteres). Assuma que `pop` retorna um espaço quando a pilha está vazia.

    Então, colocamos cada "abertura" na pilha, e retiramos uma "abertura" quando encontramos um "fechamento". Os erros possíveis são: uma "abertura" nunca é fechada (pilha não está vazia no final), uma "abertura" não casa com um "fechamento", ou um "fechamento" não tem abertura.

    ```
    // Retorne false se ha' erro, true se nao ha' erro:
    static boolean verifique(char arranjo[]) {
      Pilha pilha = new Pilha();
    ```

```
        char teste;
        for (int i=0; i<arranjo.length; i++) {
          if ((arranjo[i] == '(') || (arranjo[i] == '[')
|| (arranjo[i] == '{'))
            pilha.push(arranjo[i]); // Insere caracter na pilha.
          // Se fechamento, verifique;
         //lembre que espaco significa pilha vazia!
          if (arranjo[i] == ')') {
            teste = pilha.pop();
            if ((teste == '␣') || (teste != '(')) {
              return(false); // Detectou erro!
            }
          }
          if (arranjo[i] == ']') {
            teste = pilha.pop();
            if ((teste == '␣') || (teste != '['))
              return(false); // Detectou erro!
          }
          if (arranjo[i] == '}') {
            teste = pilha.pop();
            if ((teste == '␣') || (teste != '{'))
              return(false); // Detectou erro!
          }
        }
        // Pilha tem que estar vazia!
        if (pilha.pop() != '␣')
          return(false);
        // Se chegou ate' aqui, nao ha' erro:
        return(true);
      }
}
```

18. Codifique um método que inverte uma fila. O método recebe um objeto do tipo `FilaAr` e deve retornar uma fila invertida.

    Discussão:

    Uma solução simples: crie uma pilha; coloque todos os elementos da fila na pilha, em ordem; depois retire todos os elementos da pilha e coloque na fila. Esse procedimento inverte a fila. Escreva o código para praticar.

19. Considere a seguinte expressão em notação pós-fixa (notação polonesa reversa):

    $$ABC*+C*D+$$

    Se $A = 2$, $B = 3$, $C = 4$, $D = 5$, qual o valor da expressão? Suponha que a expressão é avaliada através de uma pilha; desenhe o conteúdo da pilha ao encontrar cada operando.

    Solução:

A sequência abaixo mostra a pilha em vários momentos do algoritmo, indicando as operações que levam a mudanças na pilha:

$$\left|\begin{array}{c} C \\ B \\ A \end{array}\right| * \left|\begin{array}{c} B*C \\ A \end{array}\right| + \left|\begin{array}{c} C \\ A+B*C \end{array}\right| * \left|\begin{array}{c} D \\ C*(A+B*C) \end{array}\right| + \left|\begin{array}{c} \\ D+C*(A+B*C) \\ \end{array}\right|$$

Para os valores indicados, temos $5 + 4(2 + 3 * 4) = 61$.

20. Escreva um método que recebe uma lista ligada `lista`, um objeto `e` e um nó `x` da lista, e insere um novo nó contendo `e` *antes* de `x`.

    Solução:

    A solução é o seguinte método que pode ser inserido na classe ListaLigada:

```
void insere(Node x, Object e) {
  // Primeiro encontra pai de x.
  for (Node p = first; p != null; p = p.next)
    if (p.next == x)
      break;
  if (p == null)
    return;
  // Cria novo no'.
  Node novo = new Node();
  novo.element = e;
  // Insere novo no'.
  p.next = novo;
  novo.next = x;
}
```

21. Modifique a implementação de fila com arranjo para que o método `dequeue` reduza o comprimento do arranjo (pela metade) sempre que apenas um quarto do arranjo esteja ocupado.

22. Escreva uma classe que implementa uma fila com arranjo circular, com os métodos `enqueue`, `dequeue` (como visto em aula) e com o novo método `enqueueFirst`, que coloca um objeto como primeiro da fila. Pede-se que o método `enqueueFirst` tenha custo $\mathcal{O}(1)$, exceto nos casos em que o arranjo está cheio (resolva essa situação por amortização).

23. Esquematize um procedimento *não* recursivo para imprimir uma árvore binária em ordem anterior.

    Discussão e solução:

    Use uma pilha:

    (a) Inicialize uma pilha e `push` a raiz na pilha.

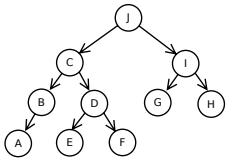
(b) Repita até que a pilha esteja vazia e não haja mais nenhum nó na árvore:

i. Retire o nó que está no topo da pilha, com `pop`, e imprima esse nó.

ii. Tome os filhos do nó que foi retirado e coloque-os na pilha; coloque primeiro o nó direito e depois o nó esquerdo.

Esse procedimento garante que todo nó será impresso antes de seus filhos, e todo filho esquerdo de um nó será processado antes do filho direito do nó.

Considere como exemplo a seguinte árvore:



A pilha será inicializada e o nó A será colocado na pilha. A partir daí:

(a) A será retirado e impresso; C e B serão colocados.

(b) B será retirado e impresso; E e D serão colocados.

(c) D será retirado e impresso; nenhum será colocado.

(d) E será retirado e impresso; H será colocado.

(e) H será retirado e impresso; nenhum será colocado.

(f) C será retirado e impresso; G e F serão colocados.

(g) F será retirado e impresso; nenhum será colocado.

(h) G será retirado e impresso; nenhum será colocado e o processo termina.

A ordem de impressão é ABDEHCFG, justamente a ordem anterior.

24. Demonstre que o número máximo de elementos em uma árvore binária de altura h é $2^h - 1$.

Solução:

A raiz está na profundidade 0, seus filhos na profundidade 1, e assim por diante. A altura da árvore é igual à maior profundidade de um nó na árvore mais 1.

O número máximo de nós ocorre quando todos os nós tem 2 filhos; nesse caso, o número de nós é:

$$\sum_{i=0}^{h-1} 2^i = \frac{2^h - 1}{2 - 1} = 2^h - 1.$$

25. Em uma árvore binária, um nó com 2 filhos é dito *completo*. Demonstre que o número de nós completos mais um é igual ao número de folhas.

Solução:

Denote por C o número de nós completos e por F o número de folhas. A maneira mais simples de demonstrar isso é por indução finita. Considere uma árvore com um único nó; nesse caso $C = 0$ e $F = 1$ (portanto a afirmação $C + 1 = F$ é verdadeira). Considere agora uma árvore qualquer com n nós e suponha que $C + 1 = F$ vale para essa árvore. Suponha agora que um nó seja adicionado a essa árvore. O pai desse nó tem que ter zero ou um filho (pois a árvore é binária). Se o pai tem zero filhos, o pai é uma folha; a adição do novo nó não muda nem C nem F e portanto $C + 1 = F$ por hipótese. Se o pai tem um filho, agora o pai ficou completo (portanto C foi incrementado) e o novo nó é uma folha nova (portanto F foi incrementado) e o resultado é que $C + 1 = F$.

26. Considere uma árvore binária de busca (ou seja, todo nó tem uma chave maior que a chave do filho esquerdo e menor que a chave do filho direito). Sabemos que em tal estrutura, uma ordem interior produz uma sequência de nós com chaves em ordem crescente. Suponha que uma árvore binária de busca seja implementada de forma que todo nó mantenha um valor inteiro `sizeOfSubtree`, que armazena o número de nós na sub-árvore com raiz no próprio nó. Por exemplo, se um nó tem dois filhos e esses filhos não tem filhos, então `sizeOfSubtree` é igual a 3. A estrutura básica seria:

```
public class SpecialNode {
        int key;
        SpecialNode left, right;
        int sizeOfSubtree;
}
```

Codifique um método `kth` na classe `SpecialNode` que encontra o k-ésimo elemento na sub-árvore com raiz no nó de chamada – o k-ésimo elemento alcançado em ordem interior. Ou seja, se `n` é um `SpecialNode` e chamarmos `n.kth(3)`, então o método deve retornar o conteúdo de `key` no terceiro nó (em ordem crescente) na sub-árvore cuja raiz é `n`. Assuma que o usuário sempre chama `n.kth(j)` com um valor de j que é maior ou igual a 1 e menor ou igual ao número de nós na sub-árvore com raiz em n.

Discussão e solução:

Para encontrar a solução, imagine um nó A com dois filhos B e C. Suponha que B tenha `sizeOfSubtree` igual a n e C tenha `size OfSubtree` igual a m. Ou seja, existem n nós *antes* de A na árvore que começa em A, e m nós *depois* de A na árvore que começa em A. Ou seja, A é o $(n + 1)$-ésimo nó. Portanto se tivermos `A.kth(j)`, temos três opções:

(a) Se $j = n + 1$, então A é o nó procurado.

(b) Se $j < n + 1$, então o j-ésimo nó está na sub-árvore com raiz em B.

(c) Se $j > n + 1$, então o j-ésimo nó está na sub-árvore com raiz em C.

Temos:

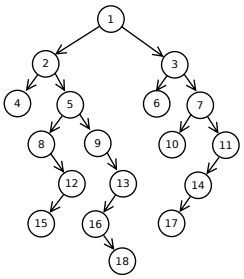
```
public class SpecialNode {
    int key;
    SpecialNode left, right;
    int sizeOfSubtree;
    ...

    // Encontre k-esimo elemento na ordem interior:
    int kth(int j) {
        int n = 0;
        if (left != null)
            n = left.sizeOfSubtree;
        if (j == (n+1))
            return(key);
        if (j < (n+1))
            return( left.kth(j) );
        if (j > (n+1))
            return( right.kth( j - (n+1) ) );
        return(-1); // Essa ultima linha nunca
                    //e' atingida(apenas para compilar).
    }

    ...
}
```

Note que o problema assume que `kth` nunca é chamado com um valor menor que um ou maior que o número total de nós, portanto não é preciso verificar essas condições. Tente imaginar o que mudaria se essas condições tivessem que ser verificadas; seria preciso verificar o número de nós no filho direito para evitar "estouro".

27. Dada a árvore abaixo, imprima os nós em ordem interior, ordem anterior, e ordem posterior.



Solução:

Ordem posterior: 4, 15, 12, 8, 18, 16, 13, 9, 5, 2, 6, 10, 17, 14, 11, 7, 3, 1.

Ordem interior: 4, 2, 8, 15, 12, 5, 9, 16, 18, 13, 1, 6, 3, 10, 7, 17, 14, 11.

Ordem anterior: 1, 2, 4, 5, 8, 12, 15, 9, 13, 16, 18, 3, 6, 7, 10, 11, 14, 17.

28. Considere uma estrutura de pilha como visto em aula na classe `PilhaAr` (ou seja, uma pilha implementada com arranjo). Temos um arranjo básico de armazenamento chamado `arranjo` e um inteiro `topo` que indica o topo da pilha; se `topo` é igual a -1, a pilha está vazia. Codifique um método `insira(Object e, int i)` que insere o objeto `e` na pilha na posição `i`. O objeto `e` deve ser inserido apenas se `i` for maior ou igual a zero ou menor ou igual à posição do topo da pilha.

    Solução:

    Um possível método a ser inserido na classe `PilhaAr` é:

```
public void insira(Object e, int i) {
    if ((i<0) || (i>top)) // Condicoes em que nada acontece.
        return;
    // Insere:
    topo++;
    if (topo == arranjo.length)
        doubleArray(); // Metodo que duplica arranjo,
                       //visto em aula.
    for (int j=topo; j>i; j--)
        arranjo[j] = arranjo[j-1]
    arranjo[i] = e;
}
```

29. Considere uma linguagem onde os caracteres tem a seguinte distribuição (indicamos espaço por "[ ]"):

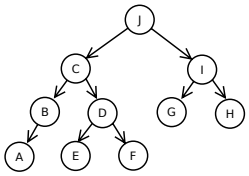
D	E	M	O	R	A	[ ]
10	18	9	20	12	23	8

Para essa linguagem, monte o código de Huffman. Codifique a palavra DEMORA no código obtido. Usando o código obtido, qual seria o tamanho médio de um texto com 100 caracteres?

30. Suponha que uma árvore binária contém um caracter em cada nó. A ordem interior de visita aos nós produz a sequência ABCEDFJGIH. A ordem anterior de visita aos nós produz a sequência JCBADEFIGH. Desenhe uma árvore binária que produza essas ordens.

    Discussão e solução:

    Para resolver, note que a ordem anterior de uma árvore sempre começa pela raiz, enquanto que a ordem interior tem a sub-árvore "esquerda" antes da raiz, e a sub-árvore "direita" depois da raiz. Assim:



Tente esquematizar um algoritmo (e codificar um método) que receba uma ordem anterior e uma ordem interior, e gere uma árvore que produza essas ordens.

31. Uma classe que implementa árvores binárias tem a seguinte definição:

```
public class NoArvoreBinaria {
   private int dado;
   private NoArvoreBinaria esquerdo, direito;

   public NoArvoreBinaria(int d) {
     dado = d;
     esquerdo = null; direito = null;
   }

   ... // Inserir, remover, etc.
}
```

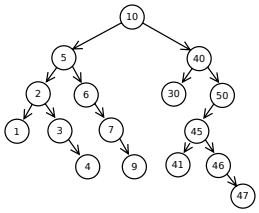
Deseja-se incluir nessa classe um método que faça busca por nível em árvore, a partir de um nó que chama o método:

```
public int buscaPorNivel(int d) {
  // Retorna 1 se encontrou d; retorna 0 caso contrario.
}
```

Descreva um algoritmo que realiza busca por nível e escreva a função `buscaPorNivel`. Qual é a complexidade da busca implementada no pior caso, para árvore contendo N nós, em notação BigOh?

Discussão: use uma estrutura de dados auxiliar; recursão direta na árvore não resolve essa questão.

32. Considere a árvore binária de busca abaixo, onde estão indicadas as chaves em cada nó.



O que acontece se a raiz é removida? *Discussão:*

Existem duas alternativas: ou substituímos 10 pelo maior elemento à sua esquerda (no caso, 9) e removemos esse elemento, ou substituímos 10 pelo menor elemento à sua direita (no caso, 30) e removemos esse elemento.

33. Considere que um texto foi codificado usando os caracteres na tabela abaixo. A tabela indica a frequência com que cada caracter apareceu no texto:

a	d	i	m	n	o	q	r	x	espaço
27	10	18	3	4	5	2	7	1	23

Obtenha o código de Huffman para esse texto. Codifique a mensagem "adiado o dia do xaxa" no código gerado. Se o texto original continha 10.000 caracteres, qual o tamanho do texto em bits após a compressão usando o código gerado?

34. Considere uma hashtable de encadeamento separado, na qual são inseridas árvores binárias de busca. Cada árvore contém N nós, e cada nó contém um dado. Suponha que N árvores foram inseridas na hashtable, e que todos os dados em uma determinada árvore sejam mapeados no mesmo elemento (*bucket*) da hashtable.

    - Suponha primeiro que houve espalhamento perfeito, ou seja, cada posição da hashtable contém apenas uma árvore.
        - Assuma que toda árvore inserida na hashtable está balanceada. Qual é a complexidade de encontrar um dado nessa estrutura, no pior caso, em notação BigOh?
        - Agora abandone a suposição que as árvores estão balanceadas. Qual é a complexidade de encontrar um dado nessa estrutura, no pior caso, em notação BigOh?
    - Agora suponha que não houve espalhamento perfeito.

- Assuma que toda árvore inserida na hashtable está balanceada. Qual é a complexidade de encontrar um dado nessa estrutura, no pior caso, em notação BigOh?
- Agora abandone a suposição que as árvores estão balanceadas. Qual é a complexidade de encontrar um dado nessa estrutura, no pior caso, em notação BigOh?

35. Uma interface gráfica deve abrir uma janela com um botão. O seguinte código foi feito:

```
import java.awt.*;
import java.awt.event.*;

public class InterfaceGrafica {
  public static void main(String args[]) {
    Frame f = new Frame();
    f.setSize(100,100);
    Button b = new Button("Aperte");
    f.add(b);
    f.show();
  }

  public InterfaceGrafica() {
  }
}
```

É necessário processar o `ActionEvent` gerado pelo `Button` através do método `actionPerformed`. A ação a ser feita é simplesmente escrever "Aperte" na tela. Lembre-se que esse método deve estar presente na interface `ActionListener` (a qual deve ser comunicada ao `Button` pelo método `addActionListener`). Com essa informações, modifique o código para processar o botão.

Considere que uma subclasse de `Button` deva ser criada, chamada `NewButton`. Essa subclasse deve ser codificada de forma que, se o `Button` no código acima for substituído por um `NewButton`, toda a funcionalidade do parágrafo anterior seja obtida sem nenhuma outra modificação. Explique como a classe `NewButton` deve funcionar e codifique a classe `NewButton`.