Unidade VI:
Árvores Binárias

Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação

Exercício (1)

- Faça um método que retorna a altura da árvore. Em seguida, insira vários elementos de forma aleatória. Para cada inserção, mostre na tela o número de elementos da árvore, o logaritmo (base 2) desse número e a altura

Exercício (2)

- Faça um método que retorne a soma dos elementos existentes na árvore

Exercício (2)

- Faça um método que retorne a soma dos elementos existentes na árvore

```
public int soma(){
    return soma(raiz);
}

public int soma(No i){
    int resp = 0;
    if(i != null){
        resp = i.elemento + soma(i.esq) + soma(i.dir);
    }
    return resp;
}
```

Exercício (3)

- Faça um método que retorne o número de elementos pares existentes na árvore.

Exercício (3)

- Faça um método que retorne o número de elementos pares existentes na árvore.

```
public int numPares(){
   return numPares(raiz);
}

public int numPares(No i){
   int resp = 0;
   if(i != null){
      resp = ((i.elemento % 2 == 0) ? 1 : 0) + numPares(i.esq) + numPares(i.dir);
   }
   return resp;
}
```

Exercício (4)

- Faça um método estático que recebe dois objetos do tipo árvore binária e retorne um booleano indicando se as duas árvores são iguais.

Exercício (4)

- Faça um método estático que recebe dois objetos do tipo árvore binária e retorne um booleano indicando se as duas árvores são iguais.

```
public static boolean igual (ArvoreBinaria a1, ArvoreBinaria a2){
   return igual(a1.raiz, a2.raiz);
}

private static boolean igual (No i1, No i2){
   boolean resp;
   if(i1 != null && i2 != null){
      resp = (i1.elemento == i2.elemento) && igual(i1.esq, i2.esq) && igual(i1.dir, i2.dir);
   } else if(i1 == null && i2 == null){
      resp = true;
   } else {
      resp = false;
   }
   return resp;
}
```

Exercício (5)

- Faça um método que retorna ***true*** se a árvore contém algum número divisível por onze.

Exercício (5)

- Faça um método que retorna ***true*** se a árvore contém algum número divisível por onze.

```
public boolean hasDiv11(){
   return hasDiv11(raiz);
}

public boolean hasDiv11(No i){
   boolean resp = false;
   if(i != null){
      resp = (i.elemento % 11 == 0) || hasDiv11(i.esq) || hasDiv11(i.dir);
   }
   return resp;
}
```

Exercício (6)

- Um algoritmo de ordenação é o *TreeSort* que insere os elementos do *array* em uma árvore binária e utiliza um "mostrar" para ordenar os elementos do *array*. Implemente o *TreeSort* e faça a análise de complexidade do mesmo.

Exercício (7)

- Faça o método ***No toArvoreBinaria(Celula primeiro, CelulaDupla primeiro)*** que recebe o nó cabeça de uma lista simples e o de outra dupla. Em seguida, crie uma árvore binária contendo os elementos intercalados das duas listas e retorne o endereço do nó raiz da árvore criada.

Exercício (7)

- Faça o método ***lulaDupla primeiro)*** que r ...o de outra dupla. Em seguida, cr ...os intercalados das duas listas ...criada.

```
No toAB(Celula p1, CelulaDupla p2){
   No resp = null;
   p1 = p1.prox;
   p2 = p2.prox;
   while(p1 != null && p2 != null){
      resp = inserir(resp, p1.elemento);
      resp = inserir(resp, p2.elemento);
      p1 = p1.prox;
      p2 = p2.prox;
   }
   while(p1 != null){
      resp = inserir(resp, p1.elemento);
      p1 = p1.prox;
   }
   while(p2 != null){
      resp = inserir(resp, p2.elemento);
      p2 = p2.prox;
   }
   return resp;
}
```

Exercício (8)

- O método ***remover*** privado e recursivo apresentado em nossa árvore recebe e um valor e retorna um No. Altere tal método para que o mesmo retorne ***void***.

Exercício (8)

- O método ***remover*** privado e recursivo apresentado em nossa árvore recebe e um valor e retorna um No. Altere tal método para que o mesmo retorne ***void***.

```
public void remover2(int x) throws Exception {
   if (raiz == null) {
      throw new Exception("Erro ao remover2!");
   } else if(x < raiz.elemento){
      remover2(x, raiz.esq, raiz);
   } else if (x > raiz.elemento){
      remover2(x, raiz.dir, raiz);
   } else if (raiz.dir == null) {
      raiz = raiz.esq;
   } else if (raiz.esq == null) {
      raiz = raiz.dir;
   } else {
      raiz.esq = antecessor(raiz, raiz.esq);
   }
}
```

```
private void remover2(int x, No i, No pai) throws Except
   if (i == null) {
      throw new Exception("Erro ao remover2!");
   } else if (x < i.elemento) {
      remover2(x, i.esq, i);
   } else if (x > i.elemento) {
      remover2(x, i.dir, i);
   } else if (i.dir == null) {
      if(x < pai.elemento){
         pai.esq = i.esq;
      } else {
         pai.dir = i.esq;
      }
   } else if (i.esq == null) {
      if(x < pai.elemento){
         pai.esq = i.dir;
      } else {
         pai.dir = i.dir;
      }
   } else {
      i.esq = antecessor(i, i.esq);
   }
}
```

Exercício (9)

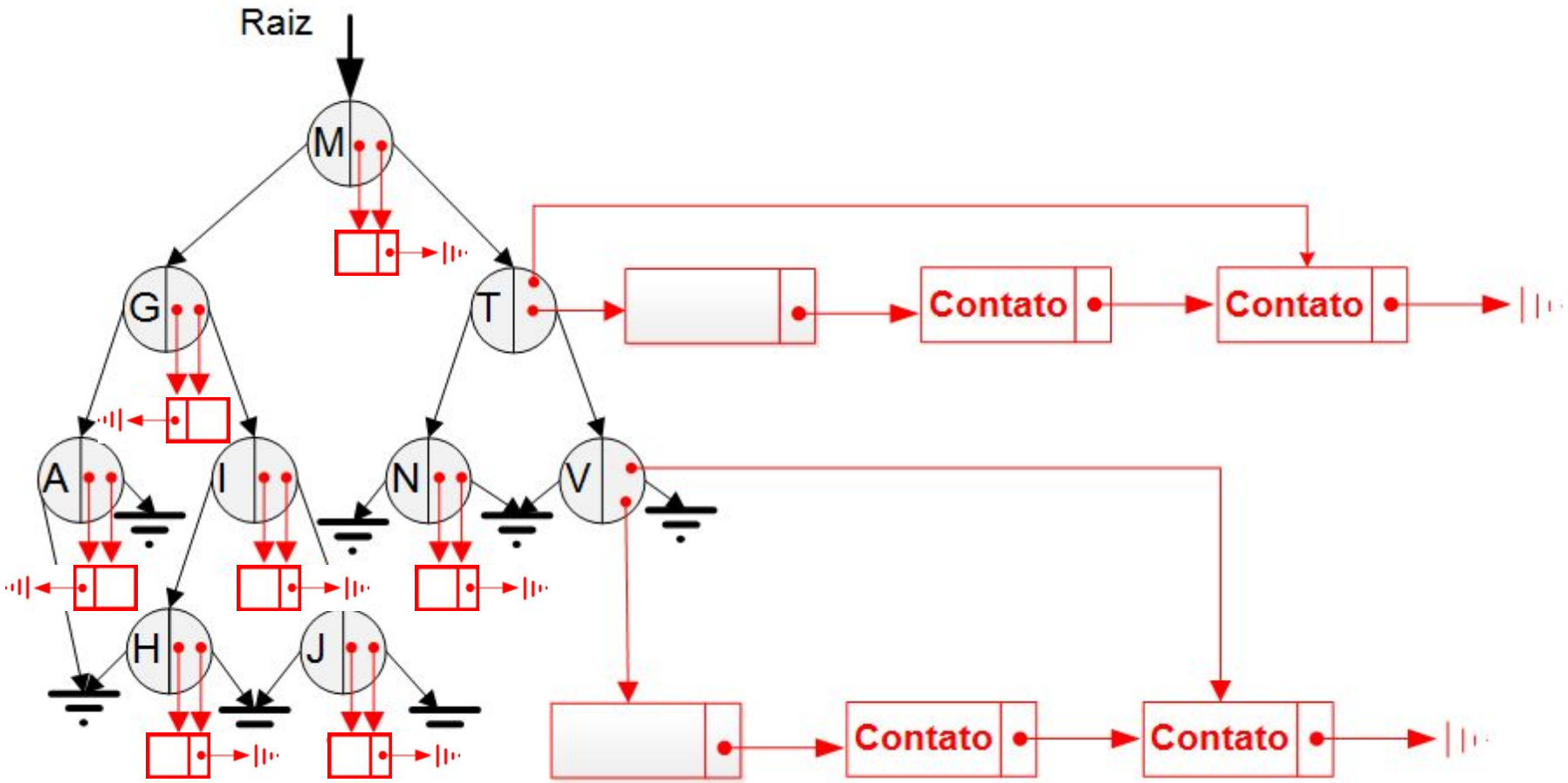
- O método ***remover*** privado e recursivo apresentado em nossa árvore recebe e um valor e retorna um No. No exercício anterior, o ***remover2*** retorna *void*. Implemente um método ***remover3*** que retorna o elemento removido.

Exercício (10)

- Você foi contratado para desenvolver uma agenda de contatos (atributos nome, telefone, email e CPF) para um escritório de advocacia

Exercício (10)

- Um colega sugeriu implementar uma árvore de binária de listas em que a pesquisa na árvore acontece pela primeira letra do nome e, quando encontramos a letra, temos uma pesquisa em uma lista de contatos



Exercício (10)

- Crie uma classe Contato contendo os atributos String nome, telefone e email e int CPF

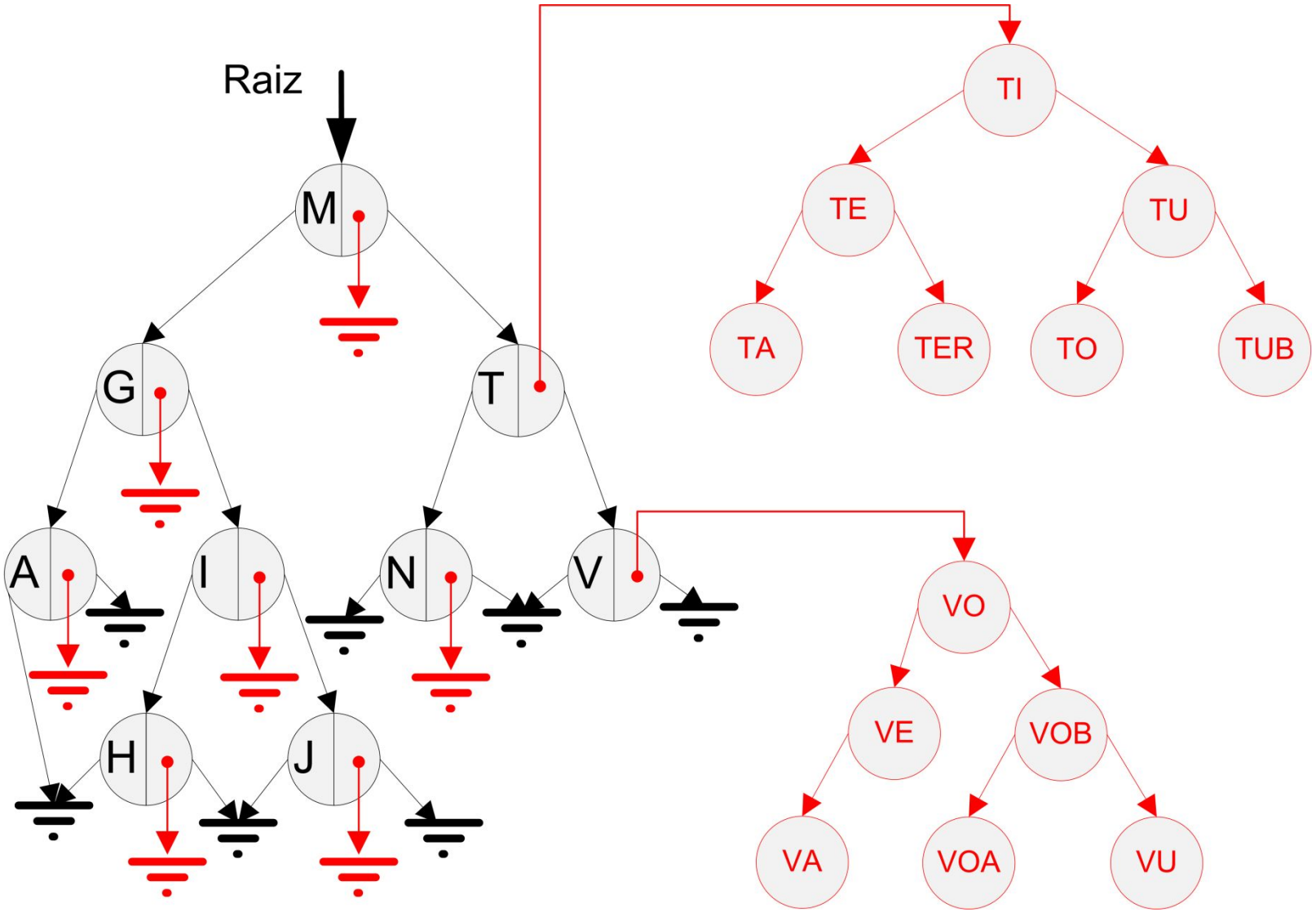
- Crie uma classe Celula contendo os atributos Contato contato e Celula prox

- Crie uma classe No contendo os atributos Celula primeiro e ultimo, No esq e dir, e char letra

- Crie uma classe Agenda contendo o atributo No raiz, os métodos inserir(Contato contato), remover(String nome), pesquisar(String nome) e pesquisar(int cpf). Para cada método, mostre o melhor e pior caso

Exercício (11)

- Implemente os métodos pesquisar, inserir, remover para a estrutura abaixo:



Exercício (12)

- Implemente os métodos pesquisar, inserir, remover para a estrutura abaixo: