DCC – UFMG – Especialização em Engenharia de Software

Trabalho Prático 3 : Busca em Texto – Árvore Patrícia
Estrutura de Dados Fundamentais – Prof. Roberto S. Bigonha

Charles Wellington de Oliveira Fortes

Belo Horizonte
Maio de 2011

Descrição

Enunciado do Trabalho

Neste trabalho deve ser utilizada uma árvore patrícia para automação de busca textual desenvolvida em Java, porém sem a utilização de quaisquer recursos nativos como ArrayList, Vector, etc., sendo permitido somente a utilização de tipos básicos, arranjos e classes.

Para resolver o problema, você deve usar árvore Patrícia, conforme interfaces a seguir, sendo estas implementadas por meio das classes ArvorePatriciaPalavra e NodoPatriciaPalavra:

Interfaces:

```
public enum NodoTipo { INTERNO, EXTERNO }

public interface IItemPalavra {
   public String getPalavra();
   public int numDeOcorrencias();
   public void addOcorrencia(int posicao);
   public int[] getOcorrencia();
}

public interface INodoPatriciaPalavra {
   public void setTipo(NodoTipo t);
   public NodoTipo getTipo();
   /*para nodo INTERNO*/
   public void setNodoPatEsq(INodoPatriciaPalavra n);
   public INodoPatriciaPalavra getNodoPatEsq();
   public void setNodoPatDir (INodoPatriciaPalavra n);
   public INodoPatriciaPalavra getNodoPatDir();
   public void setIndex(int i);
   public int getIndex();
   public int calcBit(String p);
   /*para nodo externo*/
   public void setItemPalavra(IItemPalavra item);
   public IItemPalavra getItemPalavra();
}

public interface IArvorePatriciaPalavra  {
   public void inicializa();
   public boolean vazia();
   public boolean insere(IItemPalavra item) throws InvalidKeyException;
   public boolean remove(String p)  throws InvalidKeyException ;
   public IItemPalavra pesquisa(String p)  throws InvalidKeyException;
}
```

Requisitos

- As palavras são representadas por um item que armazena suas ocorrências no texto
- Um nó da árvore patrícia pode ser interna ou externa
- A função calcBit(String) é utilizada para recuperar o i-ézimo bit da palavra passada como parâmetro, onde i é definido pelo índice armazenado no nodo;
- Os testes devem ser feitos utilizando um arquivo texto com pelo menos 1000 palavras;
- Apresente o teste e analise a complexidade do código

Considerações

O fonte dos métodos serão dispostos junto às suas descrições para melhor entendimento, sendo que o fonte completo do sistema está disposto no Anexo 2 deste trabalho.

Para o armazenamento das posições das ocorrências das palavras foi utilizado um vetor de inteiros que armazena apenas a posição da palavra no texto quanto a ordem de seu aparecimento.

Por serem parte do enunciado e estarem descritas no item descrição deste trabalho, as interfaces solicitadas não serão descritas em detalhes, seu entendimento poderá ser tido através da descrição de sua implementação.

Detalhes da implementação

Arquitetura

Pacotes

TP3 → Pacote que contém o programa de testes “Main.java” e as classes que implementam o solicitado neste trabalho.

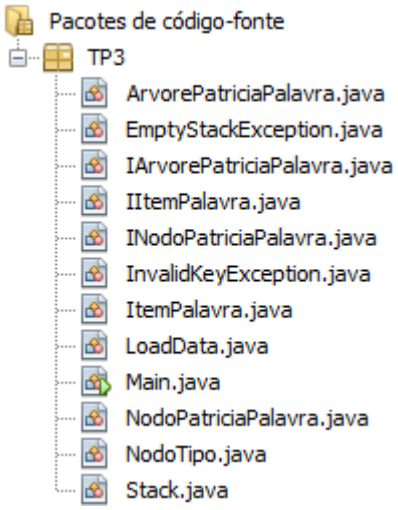

Imagem 1 – Pacotes do Sistema

Classes

TP3 → LoadData.Java:

Classe abstrata utilizada para facilitar a carga no sistema por arquivos texto. Quando necessário a importação de um arquivo de texto, basta herdar desta classe e implementar os métodos abstratos:

- getCaminhoArquivo() → Implementar um retorno de uma string que represente o caminho do arquivo a ser importado;

- getData() → Recupera a estrutura de dados importada, sendo que esta é uma matriz de strings;
- adicionar(String linha) → recebe como parâmetro uma linha do arquivo lido e a decompõe nos campos do vetor que será adicionado a matriz.

Os três métodos acima serão do tipo “protected” para que a classe herdeira seja obrigada a implementá-los, mas quem a utilizar não tenha acesso a estes métodos, se abstraindo de como são feitas suas operações de interpretação dos dados, visualizando apenas o método público “load()” que retorna a estrutura de dados que é o que interessa a ele.

- load() → implementa as operações de leitura dos dados do arquivo e chama os métodos necessários a sua interpretação e formatação, retornando ao utilizador a estrutura de uma matriz contendo os dados.

Implementação

```
package TP3.UTIL;

import java.io.*;

public abstract class LoadData {
   protected abstract String getCaminhoArquivo();
   protected abstract void adicionar(String linha);
   protected abstract String[][] getData();

   public String[][] load() throws Exception
   {
      InputStream in = new FileInputStream(getCaminhoArquivo());
      InputStreamReader reader = new InputStreamReader(in);
      BufferedReader data = new BufferedReader(reader);

      while (data.ready())
      {
         adicionar(data.readLine());
      }

      return getData();
   }
}
```

Complexidade do Algoritmo: O(n), sendo n o número de linhas do arquivo;

TP3 → ArvorePatriciaPalavra.java

A Classe ArvorePatriciaPalavra.java implementa a interface IArvorePatriciaPalavra, sendo ela a representação da implementação da árvore patrícia que será usada no sistema, seus métodos serão descritos abaixo juntamente com o detalhamento de sua implementação:

Métodos Auxiliares (Privados)

A classe ArvorePatriciaPalavra possui os métodos privados:

- “bit(String p, int idx)“ → recebe string qualquer e calcula seu valor binário, retornando o bit de uma i-ézima posição informada no parâmetro idx. É utilizada para calcular os bits para que seja decidido onde na árvore será inserido o nodo.

Implementação

```
private int bit(String p, int idx)
   {
      if (idx <=0) return 0;
      int n = (idx-1)/8;
      if (n < p.length()){
         int j = 7 - (idx-1) %8;
         int a = p.charAt(n);
         return ((a >>>j) & 1);
      } else return 1;
   }
```

Complexidade do Algoritmo: O(1).

- “CheckWord(String p)“ → Método utilizado para validar uma palavra, verificando se ela pode ser parte da árvore patrícia (se não há caracteres inválidos por exemplo);

Implementação

```
   private void CheckWord(String p) throws InvalidKeyException {
      if (p == null) throw new InvalidKeyException();
      if (p.length() == 0)throw new InvalidKeyException();
      if (p.indexOf(InvalidChar) > 0)throw new InvalidKeyException();
      if (p.equals(reg0.getPalavra()))throw new InvalidKeyException();
   }
```

Complexidade do Algoritmo: O(1).

- “createNewNodo(int bitIndex, IItemPalavra item, String word) “ →Método que cria um novo nó para ser inserido na árvore patrícia;

Implementação

```
private INodoPatriciaPalavra createNewNodo(int bitIndex, IItemPalavra item, String word) {
   INodoPatriciaPalavra innerNode = new NodoPatriciaPalavra();
   innerNode.setTipo(NodoTipo.INTERNO);
   innerNode.setIndex(bitIndex);

   INodoPatriciaPalavra outerNode = new NodoPatriciaPalavra();
   outerNode.setItemPalavra(item);
   outerNode.setTipo(NodoTipo.EXTERNO);

   if (innerNode.calcBit(word) == 0) {
      innerNode.setNodoPatEsq(outerNode);
   } else {
      innerNode.setNodoPatDir(outerNode);
   }

   return innerNode;
}
```

Complexidade do Algoritmo: O(1).

Interface Pública (Métodos Públicos)

A interface pública da classe é composta pelas funcionalidades solicitadas no enunciado do trabalho.

- pesquisa(String p) → Método que recebe como parâmetro uma palavra para ser localizada no banco de dados, retornando o item ao qual pertence a chave, caso o item não seja localizado, a rotina retorna NULL.

Implementação

```
public IItemPalavra pesquisa(String p) throws InvalidKeyException {
   INodoPatriciaPalavra currentNode = _rootNode;
   INodoPatriciaPalavra InnerNode, OuterNode;
   CheckWord(p);
   while (currentNode.getTipo() == NodoTipo.INTERNO)
   {
      InnerNode = currentNode;
      if (bit(p, InnerNode.getIndex()) == 0)
         currentNode = InnerNode.getNodoPatEsq();
      else
         currentNode = InnerNode.getNodoPatDir();
   }
   OuterNode = currentNode;
   if (p.equals(OuterNode.getItemPalavra().getPalavra()))
      return OuterNode.getItemPalavra();
   else
      return null;
}
```

Complexidade do Algoritmo: O (log N).

- remove(String p) → Método localiza uma palavra na árvore para ser removida. O algoritmo primeiro localiza a chave indicada e por fim, caso a localize, remove o item e retorna se a remoção foi realizada com sucesso ou não.

Implementação

```
public boolean remove(String p)  throws InvalidKeyException {
      INodoPatriciaPalavra currentNode = _rootNode;
      INodoPatriciaPalavra InnerNode, OuterNode, parentNode;
      CheckWord(p);
      InnerNode = currentNode;
      parentNode = InnerNode;
      while (currentNode.getTipo() == NodoTipo.INTERNO)
      {
         parentNode = InnerNode;
         InnerNode = currentNode;
         if (bit(p, InnerNode.getIndex()) == 0)
            currentNode = InnerNode.getNodoPatEsq();
         else
            currentNode = InnerNode.getNodoPatDir();
      }
      OuterNode = currentNode;
      if (p.equals(OuterNode.getItemPalavra().getPalavra())){
         if (OuterNode == InnerNode.getNodoPatEsq()) {
            OuterNode = InnerNode.getNodoPatDir();
         } else {
            OuterNode = InnerNode.getNodoPatEsq();
         }

         if (InnerNode == parentNode.getNodoPatEsq()) {
            parentNode.setNodoPatEsq(OuterNode);
         } else {
            parentNode.setNodoPatDir(OuterNode);
         }
         return true;
      }
      else
         return false;
   }
```

Complexidade do Algoritmo: O (log N).

- insere(IItemPalavra item) →Este método localiza a posição na árvore aonde o novo item deve ser inserido, cria um novo nó para o elemento e reorganiza a arvore para recebê-lo.

Implementação

```
public boolean insere(IItemPalavra item) throws InvalidKeyException {
      INodoPatriciaPalavra currentNode = _rootNode;
      INodoPatriciaPalavra InnerNode, OuterNode;
      CheckWord(item.getPalavra());
      int bitIndex = 0;

      Stack<INodoPatriciaPalavra> stack = new Stack<INodoPatriciaPalavra>();
      while (currentNode.getTipo() == NodoTipo.INTERNO)
      {
         InnerNode = currentNode;
         if (InnerNode.getIndex() == bitIndex + 1) {
            bitIndex++;
         }

         stack.empilha(currentNode);
         if (bit(item.getPalavra(), InnerNode.getIndex()) == 0)
            currentNode = InnerNode.getNodoPatEsq();
         else
            currentNode = InnerNode.getNodoPatDir();
      }
      OuterNode = currentNode;
      if (item.getPalavra().equalsIgnoreCase(OuterNode.getItemPalavra().getPalavra())){
         OuterNode.getItemPalavra().addOcorrencia(bitIndex);
         return true;
      }
      else
      {
         while (bit(item.getPalavra(), bitIndex) == bit(OuterNode.getItemPalavra()
               .getPalavra(), bitIndex)) {
            bitIndex++;
         }
         INodoPatriciaPalavra novoNodo = createNewNodo(bitIndex, item, item.getPalavra());
         item.addOcorrencia(bitIndex);
         try {
            currentNode = stack.desempilha();
            while (currentNode.getIndex() > bitIndex) {
               currentNode = stack.desempilha();
            }

         } catch (EmptyStackException e) {
            currentNode = _rootNode;
         }

         if (currentNode.calcBit(OuterNode.getItemPalavra().getPalavra()) == 0) {
            InnerNode = currentNode.getNodoPatEsq();
            currentNode.setNodoPatEsq(novoNodo);
         } else {
            InnerNode = currentNode.getNodoPatDir();
            currentNode.setNodoPatDir(novoNodo);
         }

         if (novoNodo.calcBit(item.getPalavra()) == 0) {
            novoNodo.setNodoPatDir(InnerNode);
         } else {
            novoNodo.setNodoPatEsq(InnerNode);
         }
      }
      return true;
   }
```

Complexidade do Algoritmo: O (log N).

- vazia() → O método verifica se a tabela está vazia, para tal, é verificado se o nodo raiz é do tipo externo.

Implementação

```
public boolean vazia() {
   return _rootNode.getNodoPatEsq().getTipo() == NodoTipo.EXTERNO;
}
```

Complexidade do Algoritmo: O (1).

- inicializa() → inicializa os objetos da árvore patrícia (nodo raiz).

Implementação

```
public void inicializa() {
      _rootNode = new NodoPatriciaPalavra();
      _rootNode.setTipo(NodoTipo.INTERNO);

      INodoPatriciaPalavra esq = new NodoPatriciaPalavra();
      esq.setTipo(NodoTipo.EXTERNO);
      _rootNode.setNodoPatEsq(esq);

      esq.setItemPalavra(reg0);
   }
```

Complexidade do Algoritmo: O (1).

TP3 → LoadData.Java:

Classe abstrata utilizada para facilitar a carga no sistema por arquivos texto. Quando necessário a importação de um arquivo de texto, basta herdar desta classe e implementar os métodos abstratos:

- getCaminhoArquivo() → Implementar um retorno de uma string que represente o caminho do arquivo a ser importado;
- getData() → Recupera a estrutura de dados importada, sendo que esta é uma matriz de strings;
- adicionar(String linha) → recebe como parâmetro uma linha do arquivo lido e a decompõe nos campos do vetor que será adicionado a matriz.

Os três métodos acima serão do tipo “protected” para que a classe herdeira seja obrigada a implementá-los, mas quem a utilizar não tenha acesso a estes métodos, se abstraindo de como são feitas suas operações de interpretação dos dados, visualizando apenas o método público “load()” que retorna a estrutura de dados que é o que interessa a ele.

- load() → implementa as operações de leitura dos dados do arquivo e chama os métodos necessários a sua interpretação e formatação, retornando ao utilizador a estrutura de uma matriz contendo os dados.

Implementação

```
package TP3.UTIL;

import java.io.*;

public abstract class LoadData {
   protected abstract String getCaminhoArquivo();
   protected abstract void adicionar(String linha);
   protected abstract String[][] getData();

   public String[][] load() throws Exception
   {
      InputStream in = new FileInputStream(getCaminhoArquivo());
      InputStreamReader reader = new InputStreamReader(in);
      BufferedReader data = new BufferedReader(reader);

      while (data.ready())
      {
         adicionar(data.readLine());
      }

      return getData();
   }
}
```

Complexidade do Algoritmo: O(n), sendo n o número de linhas do arquivo;

TP3 → ItemPalavra.java

A Classe ItemPalavra.java representa o item que estará dentro do nodo da árvore patrícia, sendo que o próprio item representa a palavra e suas ocorrências no meio do texto.

Por representar apenas uma estrutura com atributos e seus acessores, todas serão exibidas unicamente abaixo:

Implementação

```
package TP3;

public class ItemPalavra implements IItemPalavra{

    private String _word = "";
    private int[] _oc = new int[5000];
    private int _lastOc = 0;

    public ItemPalavra(String s) {
        _word = s;
    }

    public String getPalavra() {
        return _word;
    }

    public int numDeOcorrencias() {
        return _lastOc;
    }

    public void addOcorrencia(int posicao) {
        _oc[_lastOc++] = posicao;
    }

    public int[] getOcorrencia() {
        return _oc;
    }

    public int getPosicaoUltimaOcorrencia()
    {
        return _lastOc;
    }

}
```

Complexidade de todos os algoritmos: O(1).

TP3 → NodoPatriciaPalavra.java

A Classe NodoPatriciaPalavra.java representa um nodo da arvore patrícia.

Por representar apenas uma estrutura com atributos e seus acessores,com a exceção do método calcBit que será descrito abaixo, todas serão exibidas unicamente:

- calcBit(String P) → método que calcula o valor binário do i-ézimo caractere de uma palavra, sendo que a i-ezima posição é obtida pelo índice da posição do nodo na árvore (propriedade _index da implementação).

Implementação

```
package TP3;

public class NodoPatriciaPalavra implements INodoPatriciaPalavra
{
    private NodoTipo _nodeType;
    private int _index = 0;
    private IItemPalavra _reg;
    private INodoPatriciaPalavra _leftNode;
    private INodoPatriciaPalavra _rightNode;


    public void setTipo(NodoTipo t) {        _nodeType = t;    }

    public NodoTipo getTipo() {        return _nodeType;    }

   public void setNodoPatEsq(INodoPatriciaPalavra n)    {        _leftNode = n;    }

    public INodoPatriciaPalavra getNodoPatEsq()    {        return _leftNode;    }

    public void setNodoPatDir(INodoPatriciaPalavra n)    {        _rightNode = n;    }

    public INodoPatriciaPalavra getNodoPatDir()    {        return _rightNode;    }

    public void setIndex(int i)    {        _index = i;    }

    public int getIndex()    {        return _index;    }

    public int calcBit(String p)    {
        if (_index <= 0) {
            return 0;
        }
        int n = (_index-1) / 8;
        if (n < p.length()) {
            int j = 7 - (_index-1) % 8;
            int a = p.charAt(n);
            return ((a >>> j) & 1);
        } else {
            return 1;
        }
    }

    public void setItemPalavra(IItemPalavra item)    {        _reg = item;    }

    public IItemPalavra getItemPalavra()    {        return _reg;    }
}
```

Complexidade de todos os algoritmos: O(1).

TP3 → Stack.java

A Classe Stack.java é uma classe que representas as funcionalidades de pilhas utilizadas pelas função da árvore patrícia..

Devido ao enfoque da implementação do trabalho, a apresentação da implementação será exibida de forma única abaixo:

Implementação

```
package TP3;
public class Stack<T>
{
    private static class Elemento<T>
    {
        Elemento<T> link;
        T info;
        Elemento (T info) {
            this.info = info;
        }
    }

    private Elemento<T> topo;
    public Stack() {}
    public void empilha(T item)
    {
        Elemento<T> novo;
        novo = new Elemento<T>(item);
        novo.link = topo;
        topo = novo;
    }

    public T desempilha() throws EmptyStackException
    {
        T info;
        if (vazia()) {
            throw new EmptyStackException("a pilha está vazia");
        }
        info = topo.info;
        topo = topo.link;

        return info;
    }

    public boolean vazia()
    {
        return topo == null;
    }
}
```

Complexidade de todos os algoritmos: O(1).

Classes de exceções

Foram criadas duas classes para levantamento de exceções específicas do sistema sendo elas:

TP3 → EmptyStackException.java → Disparado sempre que se tenta desempilhar uma pilha de dados vazia.

TP3 → InvalidKeyException.java → Disparado sempre que uma chave inválida para a árvore patrícia á passada como parâmetro para os métodos de inserção, exclusão e pesquisa.

Implementação

```
package TP3;

public class EmptyStackException extends Exception{

    EmptyStackException(String string)
    {
        super(string);
    }
}

package TP3;

class InvalidKeyException extends Exception {

    public InvalidKeyException() {
    }

}
```

Saída (formatada com o número das linhas da saída)

1. run:

2. ----- Inserindo texto na árvore ------
3. ----- Pesquisando pela palavra 'nome' ------
4. Palavra: nome
5. Numero de Ocorrencias (esperado 3): 3
6. ----- Removendo a palavra 'nome' ------
7. ----- Pesquisando pela palavra 'nome' ------
8. Nenhum item encontrado!

9. CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)

Anexo1: Arquivo de Entrada (Texto sobre deuses gregos – 1089 palavras)

Geia - Mãe de todos os seres, personificaãão da terra. Surgiu do Caos e gerou Urano, os Montes, o Mar, os Titãs, os Centãmanos (Hecatonquiros), os Gigantes, as Erãnies, etc. O mito de Gãia provãvelmente comeãou como uma veneraãão neolãtica da terra-mãe antes da invasão Indo-Europãia que posteriormente se tornou a civilizaãão Helenãstica.
Urano - O primeiro rei do Universo, segundo Hesãodo (cãu estrelado). Casou-se com Gãia, da qual teve os Titãs, as Titãnidas, os Ciclopes e os Hecatonquiros. Urano, por ãdio, lanãou no Tãrtaro os Ciclopes e os Hecatonquiros, Gãia porãm deu uma foice aos Titãs para que se vingassem. Cronos, o mais audacioso deles, castrou Urano e tornou-se o senhor do universo!
Cronos - Filho de Urano e Gãia. O mais jovem dos Titãs. Se tornou senhor do cãu castrando o pai. Casou com Rãia, e teve Hãstia, Demãter, Hera, Ades e Poseidon. Como tinha medo de ser destronado, Cronos engolia os filhos ao nascerem. Comeu todos exceto Zeus, que Rãia conseguiu salvar enganando Cronos enrolando uma pedra em um pano, a qual ele engoliu sem perceber a troca. Mais tarde Zeus voltou, deu ao pai um remãdio que o fez vomitar os filhos, e logo depois o destronou e baniu-o no tãrtaro. Cronos escapou e fugiu para a Itãlia onde reinou sobre o nome de Saturno. Este perãodo no qual reinou foi chamado de "A era de ouro terrestre".
Ciclopes - Arges, Brontes e Estãropes. Pertenciam a raãa dos gigantes. Forjavam os raios e os trovães para Zeus. Teriam sido mortos por Apolo para vingar a morte de Asclãpio. Segundo Homero, porãm, teria sido um povo de gigantes rudes, fortes, indiferentes ãs divindades, dedicados ao pastoreio.
Hecantoquiros (ou Centimanos) - Briareu, Coto e Giges. Gigantes de cem braãos e cinqãenta cabeãas. Tendo hostilizado o pai, este os mandou pra horrãveis cavernas nas vãsceras da terra. Participaram da rebellião contra Urano. Quando Cronos tomou o poder, os aprisionou no tãrtaro. Libertados por Zeus, lutaram contra as titãs. Com a habilidade de arremeãar cem pedras de uma vez venceram os titãs. Briareus era guarda-costas de Zeus.
Titãs - Oceano, Hipãrion, Japeto, Cãos, Crãos e Cronos.
Titanidas - Tãia, Rãia, Tãmis, Mnemãsine, Febe e Tãis.
Zeus - O deus supremo do mundo, o deus por excelãncia. Presidia aos fenãmenos atmosfãricos, recolhia e dispersava as nuvens, comandava as tempestades, criava os relãmpagos e o trovão e lanãava a chuva com sua poderosa mão direita, ã sua vontade, o raio destruidor; por outro lado mandava chuva benãfica para fecundar a terra e amadurecer os frutos. Chamado de o pai dos deuses, por que, apesar de ser o caãula de sua divina famãlia, tinha autoridade sobre todos os deuses, dos quais era o chefe reconhecido por todos. Tinha o supremo governo do mundo e zelava pela ordem e da harmonia que reinava nas coisas. Depois de ter destronado o sei pai, dividiu com seus irmãos o domãnio do mundo. Morava no Olimpo, quando sacudia a ãgide, o escudo formidãvel que lanãava relãmpagos explodia a procela. Casou-se com Hera, porãm teve muitos amores.
Hera - Irmã e esposa de Zeus, a mais excelsa das deusas. A Ilãada a representa como orgulhosa, obstinada, ciumenta e rixosa. Odiava sobretudo Hãracles, que procurou diversas vezes matar. Na guerra de Trãia por ãdio dos troianos, devido ao julgamento de Pãris, ajudou os gregos.
Hestia - Deusa do fogo e da lareira.
Demeter - ã a maior das divindades gregas ligadas ã terra produtora; seu nome significa Terra-mãe. De Zeus teve Persãfone, que foi raptada por Hades. Enraivecida, fez com que a terra se tornasse ãrida. Zeus, para aplacã-la, obteve de Hades que Persãfone permanecesse quatro meses nos Infernos, junto com o marido, e oito meses ao lado de sua mãe. O seu mito em relaãão a Persãfone teve lugar nos mistãrios eleusinos.
Apolo - Filho de Zeus e de Leto, tambãm chamado Febo, irmão gãmeo de ãrtemis, nasceu ãs fraldas do monte Cinto, na ilha de Delos. ã o deus radiante, o deus da luz benãfica. A lenda mostra-nos Apolo, ainda garoto, combatendo contra o gigante Tãtio e matando-o, e contra a serpente Pãton, monstro saãdo da terra, que assolava os campos, matando-a tambãm. Apolo ã porãm, tambãm concebido como divindade malãfica, executora de vinganãas. Em contraposiãão, como dã a morte, dã tambãm a vida: ã mãdico, deus da saãde, amigo da juventude bela e forte. ã o inventor da adivinhaãão, da mãsica e da poesia, condutor das Musas, afasta as desventuras e protege os rebanhos.
Artemis - Deusas da caãa, filha de Zeus e Leto, irmã gãmea de Apolo. Representava a mais luminosa encarnaãão da pureza feminina. Eram-lhe oferecidos sacrifãcios humanos em tempos antiquãssimos. Deusa da Lua, declinava-se, circundada por suas ninfas, vagar de dia pelos bosques ã caãa de feras, ã

noite, porãm, com o seu pãlido raio, mostrava o caminho aos viajores. Quando a Lua, escondida pelas nuvens, tornava-se ameaãadora e incutia medo nos homens, tomava o nome de Hãcate.
Atena - Surgiu toda armada do cãrebro de Zeus, depois de ter ele engolido seu primeira esposa Mãtis. Era o sãmbolo da inteligãncia, da guerra justa, da casta mocidade e das artes domãsticas e uma das divindades mais veneradas. Um esplãndido templo, o Partenon, surgia em sua honra na Acrãpole de Atenas, a cidade que lhe era particularmente consagrada. Obra maravilhosa de Ictino e de Calãcrates, o Partenon continha uma colossal estãtua de ouro dessa deusa, de autoria do famoso escultor Fãdias.
Hermes - Filho de Zeus e de Maia, o arauto dos deuses e fiel mensageiro de seu pai, nasceu numa gruta do monte Ciline, na Arcãdia. Lodo que nasceu, fugiu do berão e roubou cinqãenta novilhas do rebanho de Apolo, em seguida, com a casca de uma tartaruga, construiu a primeira lira e com o som deste instrumento aplacou Apolo, enfurecido pelo furto; esse deus acabou por deixar-lhe as novilhas e deu-lhe o caduceu, a vara de ouro, sãmbolo da paz, n troca da lira. Zeus deu-lhe o encargo de levar os mortos a Hades, daã o epãteto de Psicompompo. Inventou, alãm da lira, as letras e os algarismos, fundou os ritos religiosos e introduziu a cultura da oliveira. Deus dos Sonhos, eram lhe oferecidos sacrifãcios de porcos, cordeiros, cabritos... Seus atributos eram a prudãncia e a esperteza. Livrou Ares das correntes dos Aloãdas, levou Prãamo ã tenda de Aquiles e matou Argos, guarda de Io. Era representado com um jovem ãgil e vigoroso, com duas pequenas asas nos pãs, um chapãu de abas largas na cabeãa e o caduceu nas mãos.

Anexo 2: Fonte do Sistema Completo

```
package TP.UTIL;

import java.io.*;

/**
 * Classe que fornece método de importar dados de um arquivo texto
 * @author Charles.Fortes
 */
public abstract class LoadData {
    protected abstract String getCaminhoArquivo();
    protected abstract void adicionar(String linha);
    protected abstract String[][] getData();

    public String[][] load() throws Exception
    {
        InputStream in = new FileInputStream(getCaminhoArquivo());
        InputStreamReader reader = new InputStreamReader(in);
        BufferedReader data = new BufferedReader(reader);

        while (data.ready())
        {
            adicionar(data.readLine());
        }

        return getData();
    }
}
```

```
public class ArvorePatriciaPalavra implements IArvorePatriciaPalavra{
    private INodoPatriciaPalavra _rootNode;
    private final char InvalidChar = 0x00FF;
    IItemPalavra reg0 = new ItemPalavra(" ");

    public void inicializa() {
        _rootNode = new NodoPatriciaPalavra();
        _rootNode.setTipo(NodoTipo.INTERNO);

        INodoPatriciaPalavra esq = new NodoPatriciaPalavra();
        esq.setTipo(NodoTipo.EXTERNO);
        _rootNode.setNodoPatEsq(esq);

        esq.setItemPalavra(reg0);
    }

    public boolean vazia() {
        return _rootNode.getNodoPatEsq().getTipo() == NodoTipo.EXTERNO;
    }

    public boolean insere(IItemPalavra item) throws InvalidKeyException {
        INodoPatriciaPalavra currentNode = _rootNode;
        INodoPatriciaPalavra InnerNode, OuterNode;
        CheckWord(item.getPalavra());
        int bitIndex = 0;

        Stack<INodoPatriciaPalavra> stack = new Stack<INodoPatriciaPalavra>();
        while (currentNode.getTipo() == NodoTipo.INTERNO)
        {
            InnerNode = currentNode;
            if (InnerNode.getIndex() == bitIndex + 1) {
                bitIndex++;
            }

            stack.empilha(currentNode);
            if (bit(item.getPalavra(), InnerNode.getIndex()) == 0)
                currentNode = InnerNode.getNodoPatEsq();
            else
                currentNode = InnerNode.getNodoPatDir();
```

```
    }
    OuterNode = currentNode;
    if (item.getPalavra().equalsIgnoreCase(OuterNode.getItemPalavra().getPalavra())){
        OuterNode.getItemPalavra().addOcorrencia(bitIndex);
        return true;
    }
    else
    {
        while (bit(item.getPalavra(), bitIndex) == bit(OuterNode.getItemPalavra()
                .getPalavra(), bitIndex)) {
            bitIndex++;
        }
        INodoPatriciaPalavra novoNodo = createNewNodo(bitIndex, item, item.getPalavra());
        item.addOcorrencia(bitIndex);
        try {
            currentNode = stack.desempilha();
            while (currentNode.getIndex() > bitIndex) {
                currentNode = stack.desempilha();
            }

        } catch (EmptyStackException e) {
            currentNode = _rootNode;
        }

        if (currentNode.calcBit(OuterNode.getItemPalavra().getPalavra()) == 0) {
            InnerNode = currentNode.getNodoPatEsq();
            currentNode.setNodoPatEsq(novoNodo);
        } else {
            InnerNode = currentNode.getNodoPatDir();
            currentNode.setNodoPatDir(novoNodo);
        }

        if (novoNodo.calcBit(item.getPalavra()) == 0) {
            novoNodo.setNodoPatDir(InnerNode);
        } else {
            novoNodo.setNodoPatEsq(InnerNode);
        }
    }
    return true;
}

public boolean remove(String p)  throws InvalidKeyException {
    INodoPatriciaPalavra currentNode = _rootNode;
    INodoPatriciaPalavra InnerNode, OuterNode, parentNode;
    CheckWord(p);
    InnerNode = currentNode;
    parentNode = InnerNode;
    while (currentNode.getTipo() == NodoTipo.INTERNO)
    {
        parentNode = InnerNode;
        InnerNode = currentNode;
        if (bit(p, InnerNode.getIndex()) == 0)
            currentNode = InnerNode.getNodoPatEsq();
        else
            currentNode = InnerNode.getNodoPatDir();
    }
    OuterNode = currentNode;
    if (p.equals(OuterNode.getItemPalavra().getPalavra())){
        if (OuterNode == InnerNode.getNodoPatEsq()) {
            OuterNode = InnerNode.getNodoPatDir();
        } else {
            OuterNode = InnerNode.getNodoPatEsq();
        }

        if (InnerNode == parentNode.getNodoPatEsq()) {
            parentNode.setNodoPatEsq(OuterNode);
        } else {
            parentNode.setNodoPatDir(OuterNode);
        }
        return true;
```

```
        }
        else
            return false;
    }

    public IItemPalavra pesquisa(String p) throws InvalidKeyException {
        INodoPatriciaPalavra currentNode = _rootNode;
        INodoPatriciaPalavra InnerNode, OuterNode;
        CheckWord(p);
        while (currentNode.getTipo() == NodoTipo.INTERNO)
        {
            InnerNode = currentNode;
            if (bit(p, InnerNode.getIndex()) == 0)
                currentNode = InnerNode.getNodoPatEsq();
            else
                currentNode = InnerNode.getNodoPatDir();
        }
        OuterNode = currentNode;
        if (p.equals(OuterNode.getItemPalavra().getPalavra()))
            return OuterNode.getItemPalavra();
        else
            return null;
    }

    private int bit(String p, int idx)
    {
        if (idx <=0) return 0;
        int n = (idx-1)/8;
        if (n < p.length()){
            int j = 7 - (idx-1) %8;
            int a = p.charAt(n);
            return ((a >>>j) & 1);
        } else return 1;
    }
    private void CheckWord(String p) throws InvalidKeyException {
        if (p == null) throw new InvalidKeyException();
        if (p.length() == 0)throw new InvalidKeyException();
        if (p.indexOf(InvalidChar) > 0)throw new InvalidKeyException();
        if (p.equals(reg0.getPalavra()))throw new InvalidKeyException();
    }

    private INodoPatriciaPalavra createNewNodo(int bitIndex, IItemPalavra item, String word) {
        INodoPatriciaPalavra innerNode = new NodoPatriciaPalavra();
        innerNode.setTipo(NodoTipo.INTERNO);
        innerNode.setIndex(bitIndex);

        INodoPatriciaPalavra outerNode = new NodoPatriciaPalavra();
        outerNode.setItemPalavra(item);
        outerNode.setTipo(NodoTipo.EXTERNO);

        if (innerNode.calcBit(word) == 0) {
            innerNode.setNodoPatEsq(outerNode);
        } else {
            innerNode.setNodoPatDir(outerNode);
        }

        return innerNode;
    }

}
```

```
package TP3;

public class ItemPalavra implements IItemPalavra{

    private String _word = "";
    private int[] _oc = new int[5000];
    private int _lastOc = 0;
```

```
    public ItemPalavra(String s) {
        _word = s;
    }

    public String getPalavra() {
        return _word;
    }

    public int numDeOcorrencias() {
        return _lastOc;
    }

    public void addOcorrencia(int posicao) {
        _oc[_lastOc++] = posicao;
    }

    public int[] getOcorrencia() {
        return _oc;
    }

    public int getPosicaoUltimaOcorrencia()
    {
        return _lastOc;
    }

}
```

```
package TP3;

public class EmptyStackException extends Exception{

    EmptyStackException(String string)
    {
        super(string);
    }
}
```

```
package TP3;
public interface IArvorePatriciaPalavra  {
    public void inicializa();
    public boolean vazia();
    public boolean insere(IItemPalavra item) throws InvalidKeyException;
    public boolean remove(String p)  throws InvalidKeyException ;
    public IItemPalavra pesquisa(String p)  throws InvalidKeyException;
}
```

```
package TP3;

public interface IItemPalavra {
    public String getPalavra();
    public int numDeOcorrencias();
    public void addOcorrencia(int posicao);
    public int[] getOcorrencia();
}
```

```
package TP3;

public interface INodoPatriciaPalavra {
    public void setTipo(NodoTipo t);
    public NodoTipo getTipo();
    /*para nodo INTERNO*/
    public void setNodoPatEsq(INodoPatriciaPalavra n);
    public INodoPatriciaPalavra getNodoPatEsq();
    public void setNodoPatDir (INodoPatriciaPalavra n);
    public INodoPatriciaPalavra getNodoPatDir();
    public void setIndex(int i);
    public int getIndex();
    public int calcBit(String p);
    /*para nodo externo*/
```

```
    public void setItemPalavra(IItemPalavra item);
    public IItemPalavra getItemPalavra();
}
```

```
package TP3;

class InvalidKeyException extends Exception {

    public InvalidKeyException() {
    }

}
```

```
package TP3;

public class Main {

    public static void main(String[] args) throws Exception {
        PatriciaLoader loader = new PatriciaLoader();
        ArvorePatriciaPalavra arvore = new ArvorePatriciaPalavra();
        arvore.inicializa();

        System.out.println("----- Inserindo texto na árvore ------ ");

        for (String[] ln : loader.load())
        {
            if (ln == null)
                break;

            for (String s : ln)
                try {
                arvore.insere(new ItemPalavra(s));
            } catch (InvalidKeyException ex) {
                System.out.println("Uma ou mais chaves contem erro!\n" + ex.getMessage());
            }
        }

        System.out.println("----- Pesquisando pela palavra 'nome' ------ ");

        IItemPalavra p = arvore.pesquisa("nome");

        System.out.println("Palavra: " + p.getPalavra()
                + "\nNumero de Ocorrencias (esperado 3): " + p.numDeOcorrencias() );

        System.out.println("----- Removendo a palavra 'nome' ------ ");
        arvore.remove("nome");

        System.out.println("----- Pesquisando pela palavra 'nome' ------ ");
        p = arvore.pesquisa("nome");

        if (p == null)
            System.out.println("Nenhum item encontrado!");
        else
            System.out.println("Palavra: " + p.getPalavra()
                + "\nNumero de Ocorrencias (esperado 0): " + p.numDeOcorrencias() );
    }

    private static class PatriciaLoader extends LoadData
    {
        private String[][] data = new String[100][];
        private int lastPosition = 0;
        @Override
        protected String getCaminhoArquivo() {
            return "DeusesGregos.txt";
        }

        @Override
        protected void adicionar(String linha) {
            data[lastPosition++] = linha.split(" ");
        }

        @Override
```

```
        protected String[][] getData() {
            return data;
        }

    }

}
```

```
package TP3;

public class NodoPatriciaPalavra implements INodoPatriciaPalavra
{
    private NodoTipo _nodeType;
    private int _index = 0;
    private IItemPalavra _reg;
    private INodoPatriciaPalavra _leftNode;
    private INodoPatriciaPalavra _rightNode;


    public void setTipo(NodoTipo t) {
        _nodeType = t;
    }

    public NodoTipo getTipo() {
        return _nodeType;
    }

   public void setNodoPatEsq(INodoPatriciaPalavra n)
    {
        _leftNode = n;
    }

    public INodoPatriciaPalavra getNodoPatEsq()
    {
        return _leftNode;
    }

    public void setNodoPatDir(INodoPatriciaPalavra n)
    {
        _rightNode = n;
    }

    public INodoPatriciaPalavra getNodoPatDir()
    {
        return _rightNode;
    }

    public void setIndex(int i)
    {
        _index = i;
    }

    public int getIndex()
    {
        return _index;
    }

    public int calcBit(String p)
    {
        if (_index <= 0) {
            return 0;
        }
        int n = (_index-1) / 8;
        if (n < p.length()) {
            int j = 7 - (_index-1) % 8;
            int a = p.charAt(n);
            return ((a >>> j) & 1);
        } else {
            return 1;
        }
    }
```

```
    public void setItemPalavra(IItemPalavra item)
    {
        _reg = item;
    }

    public IItemPalavra getItemPalavra()
    {
        return _reg;
    }
}
```

```
package TP3;

public enum NodoTipo {
    INTERNO, EXTERNO
}
```

```
package TP3;

public class Stack<T>
{
    private static class Elemento<T>
    {
        Elemento<T> link;

        T info;

        Elemento (T info) {
            this.info = info;
        }
    }

    private Elemento<T> topo;

    public Stack() {}

    public void empilha(T item)
    {
        Elemento<T> novo;

        novo = new Elemento<T>(item);
        novo.link = topo;
        topo = novo;
    }

    public T desempilha() throws EmptyStackException
    {
        T info;

        if (vazia()) {
            throw new EmptyStackException("a pilha está vazia");
        }

        info = topo.info;
        topo = topo.link;

        return info;
    }

    public boolean vazia()
    {
        return topo == null;
    }
}
```