Centro Federal de Educação Tecnológica de Minas Gerais - Departamento de Computação

Laboratório de Algoritmos e Estruturas de Dados II (2ECOM.025)

Lista de Exercícios 5

Professores: Amadeu Almeida e Thiago Rodrigues **Data de entrega:** 23 de Setembro de 2019

Instruções: leia com atenção **todas** as orientações abaixo antes de começar a lista.

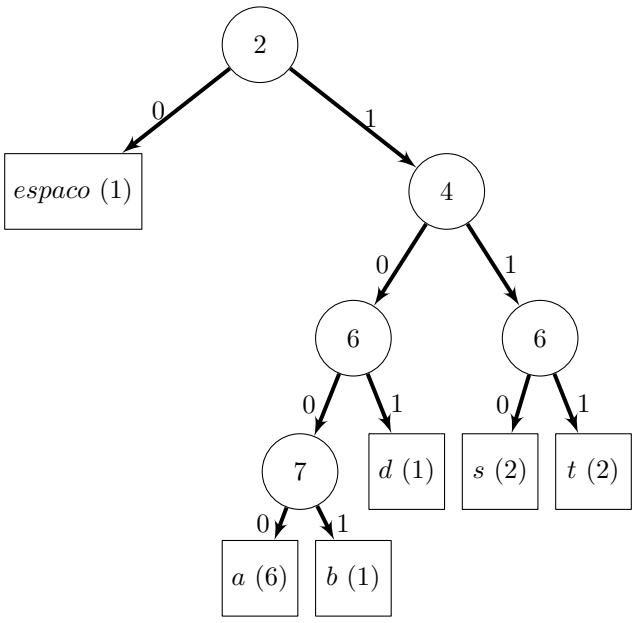
- Os exercícios deverão ser feitos individualmente ou em dupla e valem 6 pontos.
- Submeta no SIGAA um arquivo compactado chamado **Lista5.zip**. Ele deverá conter:
  - As classes Java que implementam as soluções das tarefas.
  - Um relatório em **PDF** com os gráficos dos experimentos e suas respectivas explicações.
- Caso a tarefa seja feita em dupla, apenas um aluno precisa submete-la.
- O prazo de entrega é até às 23:59 do dia 23 de setembro.
- Códigos fonte em PDF e submissões em atraso serão desconsiderados.
- O enunciado desta lista possui 2 exercícios e 4 páginas.

Exercício 1: classe ArvorePatricia

Os objetivos desta tarefa são:

- Implementar uma classe que cria uma estrutura de dados do tipo **Árvore Patrícia**;
- Utilizar esta estrutura para implementar um sistema que indexa todos os caracteres da tabela ASCII presentes em um texto;
- Computar quantidade vezes que cada um desses caracteres aparece.

Por exemplo, o texto **batata assada** possui a seguinte indexação em uma **Árvore Patricia** com chaves de 8 bits.



Legenda da figura acima:

- Os nós circulares apresentam índice do bit onde dois ou mais caracteres se diferem.
- Os nós retangulares mostram uma chave da árvore e, entre parenteses, a quantidade de vezes que ela aparece no texto.

Tabela referente a árvore:

Código ASCII	Binário	Caractere	Quantidade
32	00100000	espaço	1
97	01100001	a	6
98	01100010	b	1
100	01100100	d	1
115	01110011	s	2
116	01110100	t	2

1. Abra um projeto chamado **Lista5** e crie um arquivo nomeado ArvorePatricia.java.
2. Transcreva as classes **NoArvorePatricia**, **NoInternoArvorePatricia** e **NoExternoArvorePatricia** conforme especificado abaixo. Estas três classes devem ser incluídas na **ArvorePatricia**, de modo que cada nó interno da árvore seja um objeto de **NoInternoArvorePatricia** e cada nó externo contenha um objeto de **NoExternoArvorePatricia**.

```
// patNo nos slides
private static abstract class NoArvorePatricia {
}

//PatNoInt nos slides
private static class NoInternoArvorePatricia extends NoArvorePatricia {

    // indice que remete a posicao do bit relativo a um conjunto de letras
       em um no interno
    // index nos slides
    int indice;

    //Variaveis que indicam quem sao os nos filhos de um no interno.
    NoArvorePatricia filhoEsquerda, filhoDireita;
}

//PatNoExt nos slides
private static class NoExternoArvorePatricia extends NoArvorePatricia {

    //Atributos de um no externo da arvore:
    //um caractere que aparece no minimo uma vez
    char chave;
    // quantidade de vezes que a letra aparece
    int quantidade;
}
```

3. Implemente os atributos da classe **ArvorePatricia**:
   - private NoArvorePatricia raiz;
   - private final int numeroDeBitsNaChave;
   - private int nosVisitadosPesquisa;
   - private int caracteresDistintos;
4. Elabore os métodos a seguir:
   - **public ArvorePatricia(int nbitsChave)** - método construtor. Ele inicializa o número de bits contidos em cada chave da árvore.

- **private int pesquisa(char chavePesquisa, NoArvorePatricia raizAtual)** - pesquisa por um caractere na Árvore Patricia e retorna quantas vezes ele apareceu no texto. Se a chave não estiver na árvore, este método retorna 0.
- **public int pesquisa(char chavePesquisa)** - inicia a busca por um caractere a partir do nó raiz da árvore.
- **private int testaBit(int posicaoAtual, char chaveInsercao)** - retorna o i-ésimo bit, a partir da esquerda, da chave passada nos parâmetros.
- **private boolean verificaNoExterno(NoArvorePatricia noAtual)** - retorna **true** se um nó passado por parâmetro é externo e **false** caso contrário.
- **private NoArvorePatricia criaNoInterno(int posicaoAtual, NoArvorePatricia filhoEsq, NoArvorePatricia filhoDir)** - cria um nó interno.
- **private NoArvorePatricia criaNoExterno(char chaveInsercao)** - constrói um nó externo.
- **private NoArvorePatricia insereEntre(char chaveInsercao, NoArvorePatricia raizAtual, int posicaoAtual)** - este método objetiva:
  - Determinar a subárvore onde um nó externo deve ser criado.
  - Gerar o nó interno que será o pai do externo recém-concebido
  - Transformar a raiz corrente da subárvore no outro filho do nó interno.
- **private NoArvorePatricia insere(char chaveInsercao, NoArvorePatricia raizAtual)** - insere um novo caractere do texto na Árvore Patricia ou atualiza a quantidade de vezes que a chave apareceu na redação, caso ela já esteja na árvore.
- **public void insere(char chaveInsercao)** - inicializa o processo de inserção de um caractere à partir do nó raiz da árvore.
- **public int getNosVisitadosPesquisa()** - retorna a quantidade de nós visitados durante uma pesquisa
- **public int getCaracteresDistintos()** - retorna o número de chaves distintas encontrados em um texto

5. Explique o objetivo das classes e dos métodos implementados nos itens 2 e 4 por meio de comentários no próprio código.

Exercício 2: Experimentos computacionais

Este exercício visa testar as limitações de uma **Árvore Patricia** quando a quantidade de bits de cada chave é menor que 8 e a eficácia do método de pesquisa da classe implementada no exercício 1.

1. Antes de começar os experimentos, utilize um Gerador de Lero Lero (clique aqui) para criar um texto contendo 200 frases.
2. Experimento 1:
   (a) Crie 8 **Árvores Patricia** diferentes onde cada chave inserida em um nó externo da árvore possui N bits, no qual N varia em intervalos de 1, entre os números 1 e 8. Em seguida, insira o texto gerado no item anterior e calcule o número de chaves distintas em cada árvore.
   (b) Para cada uma das árvores geradas no exercício (a), pesquise por um caractere pertencente ao texto e retorne a quantidade de nós visitados.
3. Experimento 2:
   (a) Gere 2 **Árvores Patricia** distintas nas quais cada chave dos nós externos da árvore contém 9 e 10 bits, respectivamente. Em seguida, insira o texto criado no item 1 e calcule a quantidade de caracteres distintos em cada árvore.
   (b) Em cada uma das árvores concebidas no exercício (a), busque por um caractere que está no texto e retorne o número de nós visitados.
4. Faça três gráficos:

(a) Um que apresente, para cada valor de N, o número de caracteres distintos nas árvores do experimento 1. O eixo X deste gráfico contém a quantidade de bits em cada chave da **Árvore Patricia** (N) e o eixo Y é composto pelo número de caracteres distintos na árvore. Depois, explique os motivos que impedem alguns caracteres do texto de aparecerem nas árvores com menos de 8 bits.

(b) Um que compare, para cada valor de N, o número de nós visitados durante as buscas do experimento 1. Neste gráfico, o eixo X refere-se o número de bits em cada chave da árvore (N) enquanto o numero de nós investigados é retratado pelo eixo Y. Em seguida, explique relação entre o número de nós visitados, a quantidade de bits em cada árvore e a complexidade de um algoritmo de pesquisa em árvores.

(c) Um que mostre, para cada valor de N, o número de nós visitados e de chaves distintas encontradas durante as buscas das árvores de 8, 9 e 10 bits. O eixo X deste gráfico retrata o número de bits em cada chave da **Árvore Patricia** (N) e o eixo Y é representado pela quantidade de nós investigados e de caracteres distintos na árvore. Por fim, clarifique as razões pelas quais o número de nós visitados e de chaves diferentes são iguais nestes três casos.