Universidade Federal do Maranhão
Departamento de Informática
Disciplina: ***Estrutura de Dados II 2025.1***
Prof.: ***João Dallyson Sousa de Almeida***

Atividade Prática 2 (Individual)
20% da segunda avaliação

1. DESCRIÇÃO

Você deverá implementar uma Única Tabela Hash (obs: seguindo a implementação apresentada em aula e fornecida em anexo), que suporte dois Tipos de Entradas, ou seja uma tabela que implementa um indexador híbrido para uma aplicação de transações financeiras, com os seguintes campos:
{ id: string, valor: float, origem: string, destino: string, timestamp: string }

A tabela hash principal armazena registros indexados por dois campos:

- "id" → Tratamento de colisão por encadeamento (lista encadeada).
- "origem" → Tratamento de colisão por endereçamento aberto com sondagem quadrática.

Se ocorrerem mais que 3 colisões para a mesma "origem", você deve migrar automaticamente esses registros para uma árvore AVL.

Se a árvore AVL ultrapassar altura 10 para uma "origem", você deve transformar essa subárvore em Rubro-Negra.

Você deve implementar uma função para busca que permita buscar por "origem" e "intervalo".

OBS: Não são duas tabelas separadas, mas sim uma mesma estrutura com comportamentos diferentes para cada tipo de chave. Não há escolha entre "id" ou "origem" — todo registro é indexado pelos dois campos.

Exemplo de Entrada:

transacoes

id,origem,destino,valor,timestamp				
ID00000	ORIG001	DEST627	795,28	2024-05-11
ID00001	ORIG058	DEST782	18,01	2020-09-18
ID00002	ORIG047	DEST562	890,01	2024-01-12
ID00003	ORIG004	DEST919	960,90	2021-11-02
ID00004	ORIG011	DEST170	705,94	2023-02-16
ID00005	ORIG014	DEST312	850,85	2020-10-08
ID00006	ORIG015	DEST351	225,81	2020-10-21
ID00007	ORIG010	DEST488	805,69	2021-12-14
ID00001	ORIG009	DEST448	934,27	2024-09-27
ID00009	ORIG010	DEST950	789,68	2023-08-03

Sobre a implementação:

1. Você pode usar as classes LinkedList ou ArrayList, apenas para a lista encadeada.
2. Você deve criar a interface abaixo que será implementadas pelas Classes TreeAVL e TreeRB

```java
public interface BalancedTree <T extends Comparable<T>> {

    void insert(T value);

    boolean remove(T value);

    boolean find(T value);

    int getHeight();

    void printInOrder();


}
```

3. O nó da Árvore AVL deve ter a seguinte estrutura:

```java
public class NodeAVL <AnyType> {

     AnyType element;       // Dados do no
     NoAVL<AnyType> left;   // Filho a esquerda
     NoAVL<AnyType> right;    // Filho a direita
    int   height;              //Altura

    NodeAVL( AnyType e )
    {
        this (e, null, null );
    }

    NodeAVL(AnyType e, NodeAVL left, NodeAVL  right)
    {
        element  = e;
        left     = left;
        right    = right;
        height   = 0;
    }

}
```

4. O nó da Árvore Rubro-Negra deverá ter a seguinte estrutura:

```java
public class NodeRB <AnyType> {

    AnyType element;
    NodeRB<AnyType> parent, left, right;
    Color color;
    int N;              // conta subarvores

    enum Color {
        RED, BLACK
    }

    NodeRB( AnyType e ){
        this (e, null, null, null, false);
    }

    NodeRB(AnyType e, NodeRB l, NodeRB  r, NodeRB  p, boolean c){
        element  = e;
        left     = l;
        right    = r;
        parent    = p;
        color = Color.RED;
       // N;              // subtree count
    }
}
```

5. Você deverá usar a implementação HashTentativaLinear como base, disponível em: https://drive.google.com/file/d/1HD6IlwR6QcTw1W9ybyV984yRc3uoLAe9/view?usp=sharing

Exemplo completo de entrada:

```
transacoes = [
   {"id": "A1B2", "origem": "X9Y8", ...},  # Registro 1
   {"id": "A1B2", "origem": "X9Y8", ...},  # Registro 2 (ID repetido)
   {"id": "C3D4", "origem": "X9Y8", ...},  # Registro 3 (Origem repetida)
   {"id": "E5F6", "origem": "X9Y8", ...}   # Registro 4 (4ª colisão em "X9Y8")
]
```

Estado da Tabela Hash:

Após inserir Registro 1:

- id="A1B2" → Índice i (encadeamento): tabela[i] = [Registro1].
- origem="X9Y8" → Índice j (endereçamento aberto): tabela[j] = Registro1.

Após inserir Registro 2:

- id="A1B2" → Índice i: tabela[i] = [Registro1, Registro2].
- origem="X9Y8" → Colisão em j → usa sondagem quadrática (ex.: j+1): tabela[j+1] = Registro2.

Após inserir Registro 3:

- id="C3D4" → Novo índice k: tabela[k] = [Registro3].
- origem="X9Y8" → Colisão em j e j+1 → usa j+4 (quadrático): tabela[j+4] = Registro3.

Após inserir Registro 4:

- origem="X9Y8" atinge 4 colisões → migra TODOS registros com origem="X9Y8" para uma árvore AVL.
- A tabela hash agora armazena no índice j um ponteiro para a árvore AVL.

2. Descrição sobre os arquivos de dados

Você deve utilizar arquivos de tamanho pequeno, médio e grande.

3. Análise dos resultados

A análise deve ser feita sobre o número de comparações, atribuições e tempo de execução dos algoritmos. Procure organizar os dados coletados em tabelas/gráficos a partir dos dados.

4. Entrega

- Código fonte do programa em Java (bem identado e comentado).
- Relatório dos resultados do trabalho
- Upload no SIGAA.

O Relatório deve apresentar:

1. Testes: apresentação dos testes realizados.
2. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação

```java
import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
```

```java
import java.util.*;

public class DatasetGenerator {
    public static void main(String[] args) {
        List<Map<String, String>> transacoes = new ArrayList<>();
        Random random = new Random();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

        // 1. Gerar 20% de origens que aparecerão em 80% das
transações (Padrão 80/20)
        List<String> origensFrequentes = new ArrayList<>();
        for (int i = 0; i < 20; i++) {
            origensFrequentes.add("ORIG" + String.format("%03d",
i));
        }

        // 2. Gerar 80% de origens menos frequentes
        List<String> origensRaras = new ArrayList<>();
        for (int i = 20; i < 100; i++) {
            origensRaras.add("ORIG" + String.format("%03d", i));
        }

        // 3. Gerar transações (10.000 registros)
        for (int i = 0; i < 10000; i++) {
            Map<String, String> transacao = new HashMap<>();

            // ID (10% de chance de repetir IDs anteriores)
            String id;
            if (i > 0 && random.nextDouble() < 0.1) {
                id = transacoes.get(random.nextInt(i)).get("id"); //
Reusa ID existente
            } else {
                id = "ID" + String.format("%05d", i);
            }

            // Origem (80% chance de usar uma origem frequente)
            String origem;
            if (random.nextDouble() < 0.8) {
                origem =
origensFrequentes.get(random.nextInt(origensFrequentes.size()));
            } else {
                origem =
origensRaras.get(random.nextInt(origensRaras.size()));
            }

            // Timestamp (aleatório entre 2020-2024, não ordenado)
            Calendar cal = Calendar.getInstance();
            cal.set(2020 + random.nextInt(5), random.nextInt(12),
random.nextInt(28) + 1);
            String timestamp = sdf.format(cal.getTime());

            // Destino (aleatório)
```

```
            String destino = "DEST" + String.format("%03d",
random.nextInt(1000));

            // Valor (aleatório entre 10.00 e 1000.00)
            String valor = String.format("%.2f", 10 +
random.nextDouble() * 990);

            // Monta a transação
            transacao.put("id", id);
            transacao.put("origem", origem);
            transacao.put("destino", destino);
            transacao.put("timestamp", timestamp);
            transacao.put("valor", valor);
            System.out.println(valor);

            transacoes.add(transacao);
        }

        // 4. Salva em CSV (opcional: JSON ou outro formato)
        try (FileWriter writer = new FileWriter("transacoes.csv")) {
            writer.write("id,origem,destino,valor,timestamp\n");
            for (Map<String, String> t : transacoes) {
                writer.write(String.join(";",
                    t.get("id"),
                    t.get("origem"),
                    t.get("destino"),
                    t.get("valor"),
                    t.get("timestamp")
                ) + "\n");
            }
            System.out.println("Dataset gerado em transacoes.csv");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```