Relatório técnico da segunda prova de Estruturas de dados 2

1 - Introdução:

O presente relatório tem como objetivo mostrar a metodologia utilizada para a solução do segundo trabalho de estrutura de dados 2, apresentando discussões, resultados e descobertas.

2 - Metodologia:

Para a construção dos algoritmos foi utilizado o editor de texto VsCode com a versão 11 do Java. Para o primeiro programa (lista de prioridades) foram testado 20 condutores, sendo estes classes que continham sexo, idade, nome, endereco e uma lista de prioridades. Para o código de Mapa foram testadas listas de números inteiros (Integer) e Strings, de igual modo foram testados os mesmos valores para os Conjuntos.

3 - Processos de desenvolvimento:

3.1: Lista de prioridades:

Para o desenvolvimento do código de Lista de prioridade foi utilizado um max-heap formados por duas classes: o proprio MaxHeap.java e a classe auxiliar Bloco.java. A classe Bloco foi utilizada para a generalização do heap, sendo formada por uma key que contendo o valor do bloco, métodos gets e sets e um compareTo para comparar tipos. A classe MaxHeap de acordo como o disposto no material e pseudocódigos vistos em sala, contém a manipulação da lista do heap, ela é composta por uma lista de blocos do tipo <T>, o tamanho atual da lista e o tamanho do heap.

Para o problema apresentado foram criadas duas classes: Condutor.java e Condicao.java. A primeira como o nome surge representava todos os dados do condutor de acordo como descrito no problema, valendo salientar apenas na parte das condições que nesta classe é representada por uma lista de Condição. A classe Condição serve para gerar o score da classe Condutor, ela contém um nome, um peso e um valor booleano para indicar a ocorrência ou não da mesma.

A lógica utilizada para classificar os condutores no heap foi a criação de um score, basicamente o condutor com maior score é o condutor com maior prioridade. O score é definido como uma média ponderada da idade + condições do condutor, ela quase sempre representa um valor entre 0 e 1, sendo esses utilizados para a classificação do heap.

3.2 Conjuntos e mapas:

- Mapa:

Para o desenvolvimento do mapa requisitado no trabalho foi utilizado três classes, uma para ser a Tabela Hash, uma para conter a Key (chave), e outra para conter o Value (valor) da tabela. Como a tabela era genérica incluindo as chaves e valores, a construção da tabela requisitava dois tipos, o K para as chaves e o V para os valores de forma que internamente os tipos eram instanciados.

A classe TabelaHash.java era então composta por uma lista de Keys<K>, um inteiro m e a tabela do tipo Value<V> []. No construtor da classe Tabela Hash é requisitado um número m desejado onde o real m é setado de acordo como o primo menor antecessor a potência de 2 maior que o número passado para o construtor. A tabela é então criada dinamicamente para o tamanho m retornado.

A classe Value<V> é composta por um array do tipo V, está classe é responsável por armazenar os valores da hash, e funciona como uma lista encadeada. Já a classe

Key<K> contém a key K que armazena o valor da chave da tabela, um inteiro para o hashCode e um inteiro para a posição interna dentro array de value.

- Conjunto:

Para o desenvolvimento do Conjunto foi utilizado a mesma estrutura do mapa, no caso uma tabela hash. A diferença entre as duas aplicações é que além do nome ir para Set<V> as keys e values possuem o mesmo tipo V e armazenam basicamente o mesmo valor. No método de adicionar é modificado a parte que reconhece a duplicação de chave para remover e adicionar o valor.

4 - Resultados e discussões:

4.1 Lista de prioridades:

De longe a maior dificuldade em implementar a lista de prioridade com o max-heap foi a questão do tipo genérico. Como resultado foi utilizado a interface do comparable na classe condutor.

Outro ponto foi a questão da numeração dos pseudocódigos está começando a contar a partir de 1 sendo que na divisão os números usavam o piso. Para a subida e descida foi preciso realizar um reajuste para o devido funcionamento. Isso ocorre pois a metade do valor da posição do índice não representa o valor da posição no array; por exemplo, o index 1 do array deve ter como l/2 = 0 no caso ½ = 0.5 -> 0, só que o 2 deve apontar também apontar para o 0, pois no exemplo começando do 1, o 2 representa 3 e tem como metade 1.5 -> 1, porém no exemplo tem-se que 2/2 = 1 -> 1. A solução encontrada foi a seguinte.

```
//correção da contagem começando em 0
int aux = i;
if (i > 0 && i % 2 == 0) {
    aux--;
}
int j = Math.abs(aux / 2);
```

(Correção do index para subida)

```
int j = i * 2;
//correção da contagem a partir do 0
j -= 1;
```

(Correção do index para descida)