Nome: Leonardo Botrel Tobias **Matrícula:** 201622040333
Data: 27/10/2017 **Número:** 07
Título da prática: Implementação em JAVA do TAD JAEDsMaps

Código que implementa a classe JAEDsMaps:

```
public class JAEDsMaps {

   private int antecessor[];
   private double p[];
   private int tempo[];
   private int distancia[];
   private JGrafo grafo;

   public JAEDsMaps (JGrafo grafo) {
       this.grafo = grafo;
   }

   public void obterArvoreCMC ( int raiz ) throws Exception {
       int n = this.grafo.getNumVertices();
       this.p = new double[n];// peso dos vértices
       this.tempo = new int[n];//Tempo dos vertices
       this.distancia = new int[n];//Distancia dos vertices
       int vs[] = new int[n+1]; // vértices
       this.antecessor = new int[n];
       //Inicializa o antecessor, peso, e heap
       for (int u = 0; u < n; u ++) {
           this.antecessor[u] = -1;
           p[u] = Double.MAX_VALUE; // ∞
           tempo[u] = 0;
           distancia[u] = 0;
           vs[u+1] = u; // Heap indireto a ser construído
       }

       p[raiz] = 0;

       FPHeapMinIndireto heap = new FPHeapMinIndireto (p, vs);
           heap.constroi ();
           while (!heap.vazio ()) {
               int u = heap.retiraMin ();

               if (!this.grafo.listaAdjVazia (u)) {//Se a lista de
adjacencias não estiver vazia
                   JGrafo.Aresta adj = grafo.primeiroListaAdj (u);

                   while (adj != null) {
```

```
                        int v = adj.getV2 ();

                        if (this.p[v] > (this.p[u] +
adj.getDistancia () + adj.getTempo())) {
                            antecessor[v] = u;
                            this.tempo[v] = adj.getTempo();
                            this.distancia[v] = adj.getDistancia();
                            heap.diminuiChave (v, this.p[u] +
adj.getDistancia () + adj.getTempo());//o heap ganha uma nova chave
menor
                        }

                        adj = grafo.proxAdj (u);//adj recebe a
proxima adjacencia
                    }
                }
            }
    }

    //Retorna o antecessor
    public int antecessor ( int u) {
        return this.antecessor[u];
    }

    //Retorna o vetor de pesos
    public double peso (int u) {
        return this.p[u];
    }

    public void imprimeCaminho ( int origem, int v) {
        if (origem == v)
            System.out. println (origem);
        else if (this.antecessor[v] == -1)
            System.out. println ( "Nao existe caminho de " +origem+
" ate " +v);
        else {
            imprimeCaminho (origem, this.antecessor[v]);
            System.out. println ("Distancia: " + distancia[v] + "
Tempo: " + tempo[v] + "\n" + v);
        }
    }

}
```

Resultado obtido nos grafos do item 4:

No primeiro grafo o caminho mais curto é: vértice 1 para o vértice 4 (Distância: 5, Tempo: 2)
vértice 4 para o vértice 5 (Distância: 2, Tempo: 7)

No segundo grafo o caminho mais curto é: vértice d para o vértice c (Distância: 5, Tempo: 4)
vértice c para o vértice e (Distância: 2, Tempo: 2)
vértice e para o vértice a (Distância: 6, Tempo: 3)