CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

BRUNO NASCIMENTO DAMACENA
GIULIO LELIS SOUZA CASTRO
RAMON GRIFFO COSTA

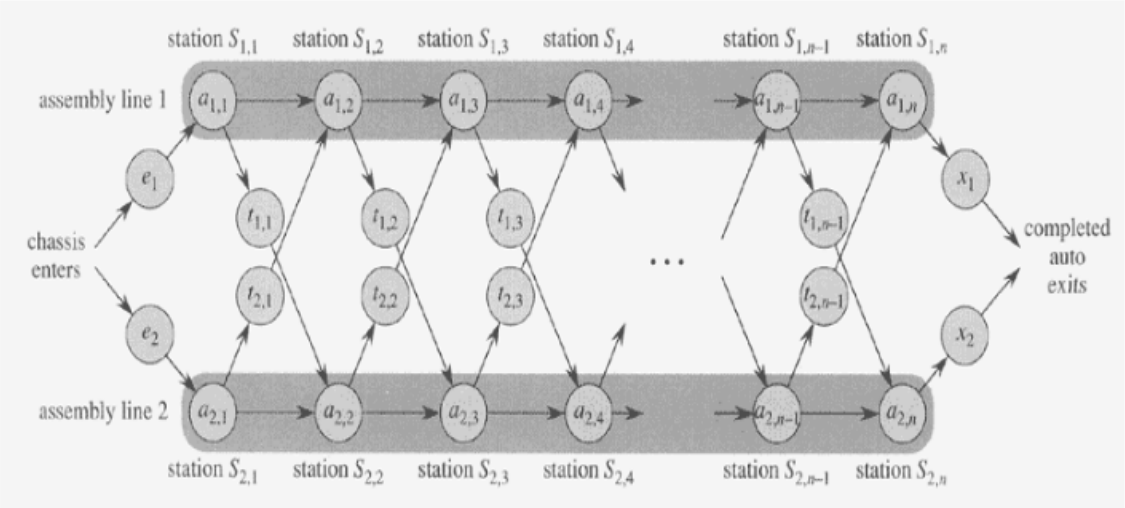
Trabalho Prático 2

Trabalho apresentado à disciplina de Laboratório de Algoritmos e Estrutura de Dados II do curso de Engenharia de Computação do Departamento de Computação do CEFET-MG

Professor Thiago de Souza Rodrigues.

Belo Horizonte
Junho de 2018

Nesse trabalho prático, o objetivo foi resolver o problema das linhas de montagem apresentados em sala de aula onde se quer verificar qual é o caminho mais eficiente da entrada nas linhas de montagem até a saída levando em consideração o tempo de processamento em cada estação e o tempo de transporte entre uma estação e outra, assim como o tempo de saída das linhas de montagem.



Códigos:

Pacote util:

```
package util;

/**
 *
 * @author BrunoDamacena
 * @author GiulioCastro
 * @author RamonGriffo
 */
public class Grafo {

    public class Peso {

        private int distancia, tempo;

        public Peso(int distancia, int tempo) {
            this.distancia = distancia;
            this.tempo = tempo;
        }

        public Peso(int distancia) {
            this.distancia = distancia;
            this.tempo = 0;
        }

        public int getDistancia() {
            return this.distancia;
        }
```

```
        public int getTempo() {
            return this.tempo;
        }

        public void setDistancia(int distancia) {
            this.distancia = distancia;
        }

        public void setTempo(int tempo) {
            this.tempo = tempo;
        }

        public int getPesoTotal() {
            return this.distancia + this.tempo;
        }

        @Override
        public String toString() {
            return distancia + "";
        }

    }

    public static class Aresta {

        private int v1, v2;
        private Peso peso;

        public Aresta(int v1, int v2, Peso peso) {
            this.v1 = v1;
            this.v2 = v2;
            this.peso = peso;
        }

        public Peso peso() {
            return this.peso;
        }

        public int v1() {
            return this.v1;
        }

        public int v2() {
            return this.v2;
        }

        public void setV1(int v1) {
            this.v1 = v1;
        }

        public void setV2(int v2) {
            this.v2 = v2;
        }
```

```
        public void setPeso(Peso peso) {
            this.peso = peso;
        }

    }
    public Peso mat[][]; // pesos do tipo inteiro
    private int numVertices;
    private int pos[]; //posicao atual ao se percorrer os adjs de um vertice v

    public Grafo(int numVertices) {
        this.mat = new Peso[numVertices][numVertices];
        this.pos = new int[numVertices];
        this.numVertices = numVertices;
        for (int i = 0; i < this.numVertices; i++) {
            for (int j = 0; j < this.numVertices; j++) {
                this.mat[i][j] = null;
            }
            this.pos[i] = -1;
        }
    }

    public void insereAresta(int v1, int v2, int distancia, int tempo) {
        Peso p = new Peso(distancia, tempo);
        this.mat[v1][v2] = p;
    }

    public void insereAresta(int v1, int v2, int distancia) {
        Peso p = new Peso(distancia);
        this.mat[v1][v2] = p;
    }

    public boolean existeAresta(int v1, int v2) {
        return (this.mat[v1][v2] != null);
    }

    public boolean listaAdjVazia(int v) {
        for (int i = 0; i < this.numVertices; i++) {
            if (this.mat[v][i] != null) {
                return false;
            }
        }
        return true;
    }

    /* retorna a primeira aresta que o vertice v participa ou
      se a lista de adjacencia de v for vazia */
    public Aresta primeiroListaAdj(int v) {
        this.pos[v] = -1;
        return this.proxAdj(v);
    }

    /* retorna a proxima aresta que o vertice v participa ou
       se a lista de adjacencia de v estiver no fim */
    public Aresta proxAdj(int v) {
```

```
        this.pos[v]++;
        while ((this.pos[v] < this.numVertices)
                && (this.mat[v][this.pos[v]] == null)) {
            this.pos[v]++;
        }
        if (this.pos[v] == this.numVertices) {
            return null;
        } else {
            return new Aresta(v, this.pos[v], this.mat[v][this.pos[v]]);
        }
    }

    public Aresta retiraAresta(int v1, int v2) {
        if (this.mat[v1][v2] == null) {
            return null; // @{\it Aresta n\~ao existe}@
        } else {
            Aresta aresta = new Aresta(v1, v2, this.mat[v1][v2]);
            this.mat[v1][v2] = null;
            return aresta;
        }
    }

    public int numVertices() {
        return this.numVertices;
    }

    public Grafo grafoTransposto() {
        Grafo grafoT = new Grafo(this.numVertices);
        for (int v = 0; v < this.numVertices; v++) {
            if (!this.listaAdjVazia(v)) {
                Aresta adj = this.primeiroListaAdj(v);
                while (adj != null) {
                    grafoT.insereAresta(adj.v2(), adj.v1(),
                    adj.peso().distancia, adj.peso().tempo);
                    adj = this.proxAdj(v);
                }
            }
        }
        return grafoT;
    }

    public void imprime() {
        System.out.println();
        for (int i = 0; i < this.numVertices; i++) {
            System.out.print("[" + i + "]-> ");
            for (int j = 0; j < this.numVertices; j++) {
                if (this.mat[i][j] != null) {
                    System.out.print(j + " > ");
                }
            }
            System.out.println(".");
        }
        System.out.println();
    }
```

```
}
```

Código-Fonte 1: Grafo.java

```
package util;

public class FPHeapMinIndireto {

    private double p[];
    private int n, pos[], fp[];

    /**
     *
     * @author BrunoDamacena
     * @author GiulioCastro
     * @author RamonGriffo
     */
    public FPHeapMinIndireto(double p[], int v[]) {
        this.p = p;
        this.fp = v;
        this.n = this.fp.length - 1;
        this.pos = new int[this.n];
        for (int u = 0; u < this.n; u++) {
            this.pos[u] = u + 1;
        }
    }

    public void refaz(int esq, int dir) {
        int j = esq * 2;
        int x = this.fp[esq];
        while (j <= dir) {
            if ((j < dir) && (this.p[fp[j]] > this.p[fp[j + 1]])) {
                j++;
            }
            if (this.p[x] <= this.p[fp[j]]) {
                break;
            }
            this.fp[esq] = this.fp[j];
            this.pos[fp[j]] = esq;
            esq = j;
            j = esq * 2;
        }
        this.fp[esq] = x;
        this.pos[x] = esq;
    }

    public void constroi() {
        int esq = n / 2 + 1;
        while (esq > 1) {
            esq--;
            this.refaz(esq, this.n);
        }
    }
```

```
    public int retiraMin() throws Exception {
        int minimo;
        if (this.n < 1) {
            throw new Exception("Erro: heap vazio");
        } else {
            minimo = this.fp[1];
            this.fp[1] = this.fp[this.n];
            this.pos[fp[this.n--]] = 1;
            this.refaz(1, this.n);
        }
        return minimo;
    }

    public void diminuiChave(int i, double chaveNova) throws Exception {
        i = this.pos[i];
        int x = fp[i];
        if (chaveNova < 0) {
            throw new Exception("Erro: chaveNova com valor incorreto");
        }
        this.p[x] = chaveNova;
        while ((i > 1) && (this.p[x] <= this.p[fp[i / 2]])) {
            this.fp[i] = this.fp[i / 2];
            this.pos[fp[i / 2]] = i;
            i /= 2;
        }
        this.fp[i] = x;
        this.pos[x] = i;
    }

    public boolean vazio() {
        return this.n == 0;
    }

    public void imprime() {
        for (int i = 1; i <= this.n; i++) {
            System.out.print(this.p[fp[i]] + " ");
        }
        System.out.println();
    }
}
```

Código-Fonte 2: FPHeapMinIndireto.java

```
package util;

/**
 *
 * @author BrunoDamacena
 * @author GiulioCastro
 * @author RamonGriffo
 */
public class LinhadeMontagem {

    private int[] tempoEstacao;
```

```
    private int[] tempoTransporte;

    public LinhadeMontagem(int[] tempoEstacao, int[] tempoTransporte) {
        this.tempoEstacao = tempoEstacao;
        this.tempoTransporte = tempoTransporte;
    }

    public int[] getTempoEst() {
        return tempoEstacao;
    }

    public int[] getTempoTrans() {
        return tempoTransporte;
    }
}
```

Código-Fonte 3: LinhadeMontagem.java

Pacote guloso:

```
package guloso;

import util.FPHeapMinIndireto;
import util.Grafo;

/**
 *
 * @author BrunoDamacena
 * @author GiulioCastro
 * @author RamonGriffo
 */
public class Dijkstra {

    private int antecessor[];
    private double p[];
    private Grafo grafo;
    private int tempoTotal;

    public void getTempoTotal() {
        System.out.println("\nTempo total: " + tempoTotal);
    }

    public Dijkstra(Grafo grafo) {
        this.grafo = grafo;
        this.tempoTotal = 0;
    }

    public void obterArvoreCMC(int raiz) throws Exception {
        int n = this.grafo.numVertices();
        this.p = new double[n]; // @{\it peso dos v\'ertices}@
        int vs[] = new int[n + 1]; // @{\it v\'ertices}@
        this.antecessor = new int[n];
        for (int u = 0; u < n; u++) {
            this.antecessor[u] = -1;
```

```
            p[u] = Double.MAX_VALUE; // @$\infty$@
            vs[u + 1] = u; // @{\it Heap indireto a ser construi\'{\i}do}@
        }
        p[raiz] = 0;
        FPHeapMinIndireto heap = new FPHeapMinIndireto(p, vs);
        heap.constroi();
        while (!heap.vazio()) {
            int u = heap.retiraMin();
            if (!this.grafo.listaAdjVazia(u)) {
                Grafo.Aresta adj = grafo.primeiroListaAdj(u);
                while (adj != null) {
                    int v = adj.v2();
                    if (this.p[v] > (this.p[u] + adj.peso().getPesoTotal())) {
                        antecessor[v] = u;
                        heap.diminuiChave(v, this.p[u] +
                        adj.peso().getPesoTotal());
                    }
                    adj = grafo.proxAdj(u);
                }
            }
        }
    }

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

    public double peso(int u) {
        return this.p[u];
    }

    public void imprimeCaminho(int origem, int v) {
        if (origem == v) {
            return;
        } else if (this.antecessor[v] == -1) {
            System.out.println("Nao existe caminho de " + origem + " ate " + v);
        } else {
            //chama a funcao recursivamente
            imprimeCaminho(origem, this.antecessor[v]);
            //imprime as arestas
            System.out.println("Aresta " + antecessor[v] + " a " + v);
            //imprime a distancia e o tempo
            System.out.println("Distancia: " +
            grafo.mat[antecessor[v]][v].getDistancia() + "
Tempo: " + grafo.mat[antecessor[v]][v].getDistancia());
            tempoTotal += grafo.mat[antecessor[v]][v].getDistancia();
        }
    }
}
```

Código-Fonte 4: Dijkstra.java

```
package guloso;
```

```
import util.Grafo;

/**
 *
 * @author BrunoDamacena
 * @author GiulioCastro
 * @author RamonGriffo
 */
public class JAEDsMaps {

    public JAEDsMaps() {
    }

    public void caminhoMinimoDijkstra(Grafo grafo, int v1, int v2)
    throws Exception {
        //inicia o algoritmo de Dijkstra com o grafo
        Dijkstra d = new Dijkstra(grafo);
        //gera os caminhos minimos de todos os vertices
        d.obterArvoreCMC(v1);
        //imprime o caminho minimo dos dois vertices passados como parametro
        d.imprimeCaminho(v1, v2);
        //imprime o tempo total
        d.getTempoTotal();
    }
}
```

Código-Fonte 5: JAEDsMaps.java

```
package guloso;

import util.Grafo;
import util.LinhadeMontagem;

/**
 *
 * @author BrunoDamacena
 * @author GiulioCastro
 * @author RamonGriffo
 */
public class Guloso {

    private Grafo grafo;
    private LinhadeMontagem L1, L2;

    public Guloso(LinhadeMontagem L1, LinhadeMontagem L2) {
        this.L1 = L1;
        this.L2 = L1;
    }

    public void gulosoMontagem() {
        //inicializa o grafo com num de vertices igual a soma do tamanho de
        custos de A1 e A2
        this.grafo = new Grafo(this.L1.getTempoEst().length +
        this.L2.getTempoEst().length - 2);
```

```
        //linha 1
        //sai do estado inicial para a primeira estacao
        grafo.insereAresta(0, 1, this.L1.getTempoEst()[0] +
        this.L1.getTempoEst()[1]);

        //sai da estacao para estacao da linha
        for (int i = 1; i < this.L1.getTempoEst().length - 2; i++) {
            grafo.insereAresta(i, i + 1, this.L1.getTempoEst()[i + 1]);
        }

        //transporta ao ponto final
        grafo.insereAresta(this.L1.getTempoEst().length - 2,
        grafo.numVertices() - 1, this.L1.getTempoEst()
        [this.L1.getTempoEst().length - 1]);

        //linha 2
        //sai do vertive inicial
        grafo.insereAresta(0, this.L1.getTempoEst().length - 1,
        this.L2.getTempoEst()[0] + this.L2.getTempoEst()[1]);

        //sai da estacao para estacao da linha
        for (int i = 0; i < this.L2.getTempoEst().length - 2; i++) {
            grafo.insereAresta(i + this.L1.getTempoEst().length - 1,
            i + this.L1.getTempoEst().length, this.L2.getTempoEst()[i + 2]);
        }

        //transporta entre linhas
        for (int i = 0; i < this.L1.getTempoTrans().length; i++) {
            grafo.insereAresta(i + 1, i + this.L1.getTempoEst().length,
            this.L1.getTempoTrans()[i] + this.L2.getTempoEst()[i + 2]);
            grafo.insereAresta(i + this.L1.getTempoEst().length - 1,
            i + 2, this.L2.getTempoTrans()[i] + this.L1.getTempoEst()[i + 2]);
        }
    }

    public void caminhoMinimoGuloso() throws Exception {
        JAEDsMaps j = new JAEDsMaps();
        j.caminhoMinimoDijkstra(grafo, 0, grafo.numVertices() - 1);
    }
}
```

Código-Fonte 6: Guloso.java

Pacote programacaoDinamica:

```
package programacaoDinamica;

import util.LinhadeMontagem;

/**
 *
 * @author BrunoDamacena
 * @author GiulioCastro
```

```
 * @author RamonGriffo
 */
public class ProgramacaoDinamica {

    private LinhadeMontagem L1, L2;  //linhas de montagem
    private int[] tempoMin1, tempoMin2; //vetores para guardar os custos
    minimos
    private int[] caminho1, caminho2; //vetores para guardar o caminho de
    quais linhas devem ser seguidas
    private int tempoFinal, linhaFinal; //valores que armazenarao a saida
    otima
    private int i; //variavel a ser iterada no metodo recursivo

    public ProgramacaoDinamica(LinhadeMontagem L1, LinhadeMontagem L2) {
        this.tempoMin1 = new int[L1.getTempoEst().length - 2];
        this.tempoMin2 = new int[L2.getTempoEst().length - 2];
        this.caminho1 = new int[L1.getTempoEst().length - 2];
        this.caminho2 = new int[L2.getTempoEst().length - 2];

        tempoMin1[0] = L1.getTempoEst()[0] + L1.getTempoEst()[1];
//inicializa a primeira posicao do primeiro vetor dos caminhos minimos
        tempoMin2[0] = L2.getTempoEst()[0] + L2.getTempoEst()[1];
//inicializa a primeira posicao do segundo vetor dos caminhos minimos

        caminho1[0] = 1;  //inicializa a primeira posicao do vetor da ordem
        das linhas
        caminho2[0] = 2;  //inicializa a primeira posicao do vetor da ordem
        das linhas

        this.L1 = L1;
        this.L2 = L2;
        this.i = 1;
    }

    //metodo dinamico para calcular o caminho minimo entre a entrada e saida
    public void caminhoMinimo() {
        //loop que comeca do 2 para que nao haja conflitos em arrays
        diferentes e de tamanhos diferentes
        for (i = 2; i <= L1.getTempoEst().length - 2; i++) {

            //linha 1
            //calcula o custo de se manter na estacao 1
            int tempo1 = tempoMin1[i - 2] + L1.getTempoEst()[i];
            //calcula o custo de se trocar para a estacao 2
            int tempo2 = tempoMin2[i - 2] +
            L2.getTempoTrans()[i - 2] + L1.getTempoEst()[i];

            //compara qual dos dois custos e melhor para ser armazenado
            no vetor
            if (tempo1 <= tempo2) {
                //armazena o custo 1 na posicao de memoria adequada do vetor
                custoMin1 (e a subsolucao otima)
                tempoMin1[i - 1] = tempo1;
                //armazena que a linha 1 e que contem a subsolucao otima
```

```
                caminho1[i - 1] = 1;
            } else {
                //armazena o custo 2 na posicao de memoria adequada do vetor
                custoMin1 (e a subsolucao otima)
                tempoMin1[i - 1] = tempo2;
                //armazena que a linha 2 e que contem a subsolucao otima
                caminho1[i - 1] = 2;
            }

            //linha 2
            //calcula o custo de se manter na estacao 2
            tempo1 = tempoMin2[i - 2] + L2.getTempoEst()[i];
            //calcula o custo de se trocar para a estacao 1
            tempo2 = tempoMin1[i - 2] + L1.getTempoTrans()[i - 2] +
            L2.getTempoEst()[i];

            //compara qual dos dois custos e melhor para ser armazenado no
            vetor
            if (tempo1 <= tempo2) {
                //armazena o custo 1 na posicao de memoria adequada do vetor
                custoMin2 (e a subsolucao otima)
                tempoMin2[i - 1] = tempo1;
                //armazena que a linha 2 e que contem a subsolucao otima
                caminho2[i - 1] = 2;
            } else {
                //armazena o custo 2 na posicao de memoria adequada do vetor
                custoMin2 (e a subsolucao otima)
                tempoMin2[i - 1] = tempo2;
                //armazena que a linha 1 e que contem a subsolucao a
                caminho2[i - 1] = 1;
            }
        }

        //ao final calcula qual saida e otima
        if (tempoMin1[i - 2] + L1.getTempoEst()[L1.getTempoEst().length - 1]
        <= tempoMin2[i - 2] +
        L2.getTempoEst()[L2.getTempoEst().length - 1]) {
            //armazena a saida otima caso seja a primeira
            tempoFinal = tempoMin1[i - 2] + L1.getTempoEst()
            [L1.getTempoEst().length - 1];
            //registra que e na linha 1
            linhaFinal = 1;
        } else {
            //armazena a saida otima caso seja a primeira
            tempoFinal = tempoMin2[i - 2] + L2.getTempoEst()
            [L2.getTempoEst().length - 1];
            //registra que e na linha 2
            linhaFinal = 2;
        }

    }

    public void imprimeCaminhoMinimo() {
        int j = linhaFinal;
```

```
        //printa a primeira linha antes do loop
        System.out.println("Linha: " + j + "    Estacao: " +
        (L1.getTempoEst().length - 2));
        //loop decrescente, da ultima estacao ate a primeira
        for (i = L1.getTempoEst().length - 2; i > 1; i--) {
            //condicao para qual linha deve ser printada, de acordo com os
            valores salvos nos vetores l1 e l2, na funcao caminhoMinimo()
            if (j == 1) {
                j = caminho1[i - 1];
            } else {
                j = caminho2[i - 1];
            }
            //printa as estacoes e linhas de acordo com os valores previamente armazenad
            System.out.println("Linha: " + j + "    Estacao: " + (i - 1));
        }
        //ao final imprime o tempo total
        System.out.println("\nTempo gasto: " + tempoFinal + "\n\n");
    }
}
```

Código-Fonte 7: ProgramacaoDinamica.java

Pacote tp2:

```
package tp2;

import programacaoDinamica.ProgramacaoDinamica;
import util.LinhadeMontagem;
import guloso.*;

/**
 *
 * @author BrunoDamacena
 * @author GiulioCastro
 * @author RamonGriffo
 */
public class TP2 {

    public static void main(String[] args) throws Exception {
        Guloso guloso;
        ProgramacaoDinamica programacaoDinamica;
        LinhadeMontagem Linha1;
        LinhadeMontagem Linha2;

        System.out.println("Instancia 1");
        /* tempo de processamento de cada estacao na Linha 1 com o primeiro e
        ultimo elementos sendo o tempo de entrada e saida dessa linha,
        respectivamente */
        int[] tempoEstacao1 = new int[]{3, 5, 7, 10, 5, 9, 11, 9, 5, 2, 6};
        /* tempo de processamento de cada estacao na Linha 2 com o primeiro e
        ultimo elementos sendo o tempo de entrada e saida dessa linha,
        respectivamentes */
        int[] tempoEstacao2 = new int[]{2, 6, 3, 9, 11, 4, 9, 3, 12, 4, 5};
```

```
        // tempo de transporte de uma Estacao na Linha 1 ate a Estacao
        seguinte na Linha 2
        int[] tempoTransporte1 = new int[]{3, 5, 4, 2, 7, 5, 8, 1};
        // tempo de transporte de uma Estacao na Linha 2 ate a Estacao
        seguinte na Linha 1
        int[] tempoTransporte2 = new int[]{5, 3, 7, 5, 6, 2, 5, 2};

        Linha1 = new LinhadeMontagem(tempoEstacao1, tempoTransporte1);
//cria a linha 1 com seus respectivos custos de estacoes e transportes
        Linha2 = new LinhadeMontagem(tempoEstacao2, tempoTransporte2);
//cria a linha 2 com seus respectivos custos de estacoes e transportes

        System.out.println("\nGuloso: ");
        //Roda o algoritmo guloso para a instancia 1
        guloso = new Guloso(Linha1, Linha2);
        guloso.gulosoMontagem();
        guloso.caminhoMinimoGuloso();

        System.out.println("\nProgramacao Dinamica: \n");
        //roda o algoritmo exponencial para a nstancia 1
        programacaoDinamica = new ProgramacaoDinamica(Linha1, Linha2);
        programacaoDinamica.caminhoMinimo();
        programacaoDinamica.imprimeCaminhoMinimo();

        System.out.println("Instancia 2");

        tempoEstacao1 = new int[]{5, 10, 6, 3, 8, 5, 3, 7, 12, 8};
        tempoEstacao2 = new int[]{7, 3, 5, 3, 7, 6, 4, 9, 10, 9};

        tempoTransporte1 = new int[]{4, 2, 7, 2, 5, 8, 2};
        tempoTransporte2 = new int[]{6, 1, 7, 3, 6, 4, 5};

        Linha1 = new LinhadeMontagem(tempoEstacao1, tempoTransporte1);
//cria a linha A1 com seus respectivos custos de estacoes e transportes
        Linha2 = new LinhadeMontagem(tempoEstacao2, tempoTransporte2);
//cria a linha A2 com seus respectivos custos de estacoes e transportes

        System.out.println("\nGuloso: ");
        //Roda o algoritmo guloso para a instancia 2
        guloso = new Guloso(Linha1, Linha2);
        guloso.gulosoMontagem();
        guloso.caminhoMinimoGuloso();

        System.out.println("\nProgramacao Dinamica: \n");
        //roda o algoritmo exponencial para a instancia 2
        programacaoDinamica = new ProgramacaoDinamica(Linha1, Linha2);
        programacaoDinamica.caminhoMinimo();
        programacaoDinamica.imprimeCaminhoMinimo();
    }
}
```

Código-Fonte 8: TP2.java

Entradas: Instância 1:
A1: [3, 5, 7, 10, 5, 9, 11, 9, 5, 2, 6]
A2: [2, 6, 3, 9, 11, 4, 9, 3, 12, 4, 5]

T1: [3, 5, 4, 2, 7, 5, 8, 1]
T2: [5, 3, 7, 5, 6, 2, 5, 2]

Instância 2:
A1: [5, 10, 6, 3, 8, 5, 3, 7, 12, 8]
A2: [7, 3, 5, 3, 7, 6, 4, 9, 10, 9]

T1: [4, 2, 7, 2, 5, 8, 2]
T2: [6, 1, 7, 3, 6, 4, 5]

Saída:
Instancia 1

Guloso:
Aresta 0 a 10
Distancia: 8 Tempo: 8
Aresta 10 a 11
Distancia: 7 Tempo: 7
Aresta 11 a 12
Distancia: 10 Tempo: 10
Aresta 12 a 13
Distancia: 5 Tempo: 5
Aresta 13 a 14
Distancia: 9 Tempo: 9
Aresta 14 a 15
Distancia: 11 Tempo: 11
Aresta 15 a 16
Distancia: 9 Tempo: 9
Aresta 16 a 17
Distancia: 5 Tempo: 5
Aresta 17 a 18
Distancia: 2 Tempo: 2
Aresta 18 a 19
Distancia: 6 Tempo: 6

Tempo total: 72

Programacao Dinamica:

Linha: 1 Estação: 9
Linha: 1 Estação: 8
Linha: 2 Estação: 7
Linha: 2 Estação: 6

Linha: 2 Estação: 5
Linha: 2 Estação: 4
Linha: 2 Estação: 3
Linha: 2 Estação: 2
Linha: 2 Estação: 1

Tempo gasto: 65

Instancia 2

Guloso:
Aresta 0 a 1
Distancia: 15 Tempo: 15
Aresta 1 a 2
Distancia: 6 Tempo: 6
Aresta 2 a 3
Distancia: 3 Tempo: 3
Aresta 3 a 4
Distancia: 8 Tempo: 8
Aresta 4 a 5
Distancia: 5 Tempo: 5
Aresta 5 a 6
Distancia: 3 Tempo: 3
Aresta 6 a 7
Distancia: 7 Tempo: 7
Aresta 7 a 8
Distancia: 12 Tempo: 12
Aresta 8 a 17
Distancia: 8 Tempo: 8

Tempo total: 67

Programacao Dinamica:

Linha: 1 Estação: 8
Linha: 1 Estação: 7
Linha: 1 Estação: 6
Linha: 1 Estação: 5
Linha: 1 Estação: 4
Linha: 1 Estação: 3
Linha: 2 Estação: 2
Linha: 2 Estação: 1

Tempo gasto: 62