Centro Federal de Educação Tecnológica de Minas Gerais

Disciplina: Laboratório de Algoritmos e Estruturas de Dados 2

Trabalho Prático 2

Nomes: Lucas Rafael Alves de Souza;
Sérgio Henrique Mendes de Assis;
Yasmim Augusta Gomes.

1. Classe De instanciação de uma linha de montagem:

```
public class LinhaDeMontagem {
	private int[] linhadeMontagem;
	private int entrada;
	private int saida;

	public LinhaDeMontagem(int tamanho, int entrada, int saida) { // é inicializada com o tamanho da linha e os valores de entrada e saida
		this.linhadeMontagem = new int[tamanho];
		this.entrada = entrada;
		this.saida = saida;
	}

	public void valorestacao(int indice, int valor) { // deve-se adcionar o valor de tempo de cada estacao
		this.linhadeMontagem[indice -1] = valor;
	}
	public int[] getLinhadeMontagem() {
		return this.linhadeMontagem;
	}
	public int getEntrada() {
		return this.entrada;
	}
	public int getSaida() {
		return this.saida;
	}
}
```

2. Classe de instanciação das duas linhas de montagem com as transições:

```
public class LinhaDuplaComTransicao {
	private LinhaDeMontagem linha_A;
	private LinhaDeMontagem linha_B;

	private int [] transicao_AB;
	private int[] transicao_BA;

	public LinhaDuplaComTransicao(LinhaDeMontagem linha_A, LinhaDeMontagem linha_B) {// é inicializadado com as duas linhas de montagem
		this.linha_A = linha_A;
		this.linha_B = linha_B;

		int [] aux = linha_A.getLinhadeMontagem();
		int tamanho = aux.length;

		transicao_AB = new int[tamanho -1];
		transicao_BA = new int[tamanho-1];

	}
	public void t_AB(int indice, int valor) { // deve-se adcionar o valor das transicoes
		this.transicao_AB[indice -1] = valor;
	}
	public void t_BA(int indice, int valor) {
		this.transicao_BA[indice -1] = valor;
	}
```

```
	public LinhaDeMontagem getLinha_A() {
		return this.linha_A;
	}
	public LinhaDeMontagem getLinha_B() {
		return this.linha_B;
	}
	public int[][] transicoes(){
		int tamanho = transicao_BA.length;
		int[][] matriz = new int[2][tamanho];
		for(int i = 0; i<tamanho; i++) {
			matriz[0][i] = this.transicao_AB[i];
			matriz[1][i] = this.transicao_BA[i];
		}
		return matriz;
	}

}
```

3. Classe que realiza a solução por programação dinâmica:

```
public class ProgramacaoDinamica {
	private LinhaDuplaComTransicao linhas;

	private int [][] memorizacao;// matriz que memoriza
	private int [][] valores;// matriz que memoriza o caminho;
	private int valor_final;

	int tamanho;


	public ProgramacaoDinamica (LinhaDuplaComTransicao linhas) {
		this.linhas = linhas;
		this.tamanho = linhas.getLinha_A().getLinhadeMontagem().length;

		this.memorizacao = new int[2][tamanho];
		this.valores = new int[2][tamanho-1];
		this.valor_final = 0;

	}
	private void preencheMemoria(int estacao, int entradaA, int entradaB,
int[] linhaA, int[] linhaB, int[][]transicoes) {
		if(estacao <= tamanho) {
			if(estacao == 1 ) {// faz a primeira chamada de estcao
				memorizacao[0][0] = entradaA + linhaA[0];
				memorizacao[1][0] = entradaB + linhaB[0];


				preencheMemoria(estacao+1, entradaA, entradaB,
linhaA, linhaB, transicoes);
			}else {
				int tempoA1 = memorizacao[0][estacao-2] +
linhaA[estacao-1]; // soma o tempo indo pela mesma linha A
				int tempoA2 = memorizacao[1][estacao-2] +
transicoes[1][estacao-2] + linhaA[estacao-1];// soma o tempo indo pela linha
B
				if(tempoA1 <= tempoA2) {// verifica a menor, se for
menor ou igual preenche a tabela de memoria A;
					memorizacao[0][estacao-1] = tempoA1;
					valores[0][estacao-2] = 1;
```

```
                        } else {
                                memorizacao[0][estacao-1] = tempoA2;
                                valores[0][estacao-2] = 2;
                        }
                        int tempoB1 = memorizacao[1][estacao-2] +
linhaB[estacao-1];
                        int tempoB2 = memorizacao[0][estacao-2] +
transicoes[0][estacao-2] + linhaB[estacao-1];
                        if(tempoB1 <= tempoB2) {// verifica a menor, se for
menor ou igual preenche a tabela de memoria B;
                                memorizacao[1][estacao-1] = tempoB1;
                                valores[1][estacao-2] = 2;
                        } else {
                                memorizacao[1][estacao-1] = tempoB2;
                                valores[1][estacao-2] = 1;
                        }
                        preencheMemoria(estacao+1, entradaA, entradaB,
linhaA, linhaB, transicoes);

                }
            }

      }


      public void obterResposta() {
            int[] linhaA =  linhas.getLinha_A().getLinhadeMontagem();
            int[] linhaB =  linhas.getLinha_B().getLinhadeMontagem();

            int entradaA = linhas.getLinha_A().getEntrada();
            int entradaB = linhas.getLinha_B().getEntrada();

            int[][] transicoes = linhas.transicoes();

            int saida_A = linhas.getLinha_A().getSaida();
            int saida_B = linhas.getLinha_B().getSaida();


            preencheMemoria(1, entradaA, entradaB, linhaA, linhaB,
transicoes);

            int [] resposta = new int[tamanho];
            int fim;

            if((saida_A + memorizacao[0][tamanho-1]) <= (saida_B +
memorizacao[1][tamanho-1])) {// verifica qual sera o menor caminho
                  resposta[tamanho-1] = 1;
                  valor_final = saida_A + memorizacao[0][tamanho-1];
                  fim = 1;
            }else {
                  resposta[tamanho-1] = 2;
                  fim = 2;
                  valor_final = saida_B + memorizacao[1][tamanho-1];
            }
            for(int i = tamanho-2; i >= 0; i--) {
                  resposta[i] = valores[fim-1][i];
                  fim = valores[fim-1][i];
```

```
            }
            for(int j = 0; j < tamanho; j++) {
                  System.out.println("Linha de montagem " + resposta[j] + ",
Estacao " + (j+1) );

            }
            System.out.println("O caminho tem o valor de: " + valor_final);

      }
```

4. Classe com solução por Algoritmo Guloso:

```
public class Guloso {
      private LinhaDuplaComTransicao linhas;
      private int[] resposta;
      private int tamanho;
      private int valor_final;

      public Guloso (LinhaDuplaComTransicao linhas) {
            this.linhas = linhas;

            this.tamanho = linhas.getLinha_A().getLinhadeMontagem().length;

            this.resposta = new int[tamanho];
            this.valor_final = 0;
      }
```

```
      private void execucao(int estacao, int entradaA, int entradaB, int[]
linhaA, int[] linhaB, int[][]transicoes, int saida_A, int saida_B) {
            if(estacao < tamanho) {
                  if(estacao == 1 ) {// faz a primeira chamada de estcao

                        int tempoA = entradaA + linhaA[0]; // soma o tempo
indo pela mesma linha A
                        int tempoB = entradaB + linhaB[0];// soma o tempo
indo pela linha B
                        if(tempoA <= tempoB) {// verifica a menor, se for
menor ou igual preenche a tabela de memoria A;
                              resposta[estacao-1] = 1;
                              valor_final = entradaA + linhaA[0];
                        } else {
                              resposta[estacao-1] = 2;
                              valor_final = entradaB + linhaB[0];
                        }

                        execucao(estacao+1, entradaA, entradaB, linhaA,
linhaB, transicoes, saida_A, saida_B);
                  }else {
                        if(resposta[estacao-2] == 1) {
                              int tempoA = linhaA[estacao-1]; // soma o
tempo indo pela mesma linha A
                              int tempoB = transicoes[0][estacao-2] +
linhaB[estacao-1];// soma o tempo indo pela linha B
                              if(tempoA <= tempoB) {// preenche a resposta
verificando o menor
                                    resposta[estacao-1] = 1;
                                    valor_final = valor_final + tempoA;
```

```
                                    } else {
                                          resposta[estacao-1] = 2;
                                          valor_final = valor_final + tempoB;
                                    }
                                    execucao(estacao+1, entradaA, entradaB,
linhaA, linhaB, transicoes, saida_A, saida_B);
                              }else {
                                    if(resposta[estacao-2] == 2) {
                                          int tempoA = transicoes[1][estacao-2]
+ linhaA[estacao-1]; // verifica a proxima estacao na linha A
                                          int tempoB = linhaB[estacao-1]; //
verifica a proxima
                                          if(tempoA <= tempoB) {// verifica a
menor, se for menor ou igual preenche a resposta
                                                resposta[estacao-1] = 1;
                                                valor_final = valor_final +
tempoA;
                                          } else {
                                                resposta[estacao-1] = 2;
                                                valor_final = valor_final +
tempoB;
                                          }
                                          execucao(estacao+1, entradaA,
entradaB, linhaA, linhaB, transicoes, saida_A, saida_B);
                                    }

                        }

                  }
                  }else if(estacao == tamanho) {
                  if(resposta[estacao-2] == 1) {
                        int tempoA = linhaA[estacao-1] + saida_A; // soma o tempo
indo pela mesma linha A
                        int tempoB = transicoes[0][estacao-2] + linhaB[estacao-1]
+ saida_B;// soma o tempo indo pela linha B
                        if(tempoA <= tempoB) {// preenche a resposta verificando o
menor
                              resposta[estacao-1] = 1;
                              valor_final = valor_final + tempoA;
                        } else {
                              resposta[estacao-1] = 2;
                              valor_final = valor_final + tempoB;
                        }
                  }else {
                        if(resposta[estacao-2] == 2) {
                              int tempoA2 = transicoes[1][estacao-2] +
linhaA[estacao-1] + saida_A; // verifica a proxima estacao na linha A
                              int tempoB2 = linhaB[estacao-1] +  saida_B; //
verifica a proxima
                              if(tempoA2 <= tempoB2) {// verifica a menor, se for
menor ou igual preenche a resposta
                                    resposta[estacao-1] = 1;
                                    valor_final = valor_final + tempoA2;
                              } else {
                                    resposta[estacao-1] = 2;
                                    valor_final = valor_final + tempoB2;
                              }
                  }
```

```
	}
	}

}
	public void obterResposta() {
		int[] linhaA =  linhas.getLinha_A().getLinhadeMontagem();
		int[] linhaB =  linhas.getLinha_B().getLinhadeMontagem();

		int entradaA = linhas.getLinha_A().getEntrada();
		int entradaB = linhas.getLinha_B().getEntrada();

		int[][] transicoes = linhas.transicoes();

		int saida_A = linhas.getLinha_A().getSaida();
		int saida_B = linhas.getLinha_B().getSaida();


		execucao(1, entradaA, entradaB, linhaA, linhaB, transicoes,
saida_A, saida_B);




		for(int j = 0; j < tamanho; j++) {
			System.out.println("Linha de montagem " + resposta[j] + ",
Estacao " + (j+1) );
		}

		System.out.println("O caminho tem o valor de: " + valor_final);
	}
```

5. Classe main para a execução dos testes

```
}**
 *
 * @Nomes:Lucas Rafael Alves de Souza.  Sergio Henrique Mendes de Assis.
Yasmim Augusta Gomes.
 *
 */
public class Main {
	public static void main ( String [] args) {
		LinhaDeMontagem linhaA1 = new LinhaDeMontagem(6, 2 ,3);
		LinhaDeMontagem linhaA2 = new LinhaDeMontagem(6, 4, 2);

		linhaA1.valorestacao(1,7);
		linhaA1.valorestacao(2,9);
		linhaA1.valorestacao(3,3);
		linhaA1.valorestacao(4,4);
		linhaA1.valorestacao(5,8);
		linhaA1.valorestacao(6,4);

		linhaA2.valorestacao(1,8);
		linhaA2.valorestacao(2,5);
		linhaA2.valorestacao(3,6);
		linhaA2.valorestacao(4,4);
		linhaA2.valorestacao(5,5);
		linhaA2.valorestacao(6,7);
```

```
            LinhaDuplaComTransicao caso1 = new
LinhaDuplaComTransicao(linhaA1, linhaA2);

            caso1.t_AB(1,2);
            caso1.t_AB(2,3);
            caso1.t_AB(3,1);
            caso1.t_AB(4,3);
            caso1.t_AB(5,4);

            caso1.t_BA(1,2);
            caso1.t_BA(2,1);
            caso1.t_BA(3,2);
            caso1.t_BA(4,2);
            caso1.t_BA(5,1);

            ProgramacaoDinamica teste1 = new ProgramacaoDinamica(caso1);
            Guloso teste1_Guloso = new Guloso(caso1);

            System.out.println("Resposta Caso Exemplo Programacao
Dinamica");
            teste1.obterResposta();

            System.out.println("Resposta Caso Exemplo Guloso");
            teste1_Guloso.obterResposta();

            /*
             * Execucao para as duas instancias
             */
            /// Primeira instancia, criar as linhas de montagem
            LinhaDeMontagem linhaA3 = new LinhaDeMontagem(9, 3 ,6);
            int vec_A3[] = {5,7,10,5,9,11,9,5,2};
            for(int i = 1; i <= 9; i++) {
                  linhaA3.valorestacao(i,vec_A3[i-1]);
            }
            LinhaDeMontagem linhaA4 = new LinhaDeMontagem(9, 2 ,5);
            int vec_A4[] = {6,3,9,11,4,9,3,12,4};
            for(int i = 1; i <= 9; i++) {
                  linhaA4.valorestacao(i,vec_A4[i-1]);
            }
            LinhaDuplaComTransicao caso2 = new
LinhaDuplaComTransicao(linhaA3, linhaA4);

            int vec_TA2[] = {3,5,4,2,7,5,8,1};
            for(int i = 1; i <= 8; i++) {
                  caso2.t_AB(i, vec_TA2[i-1]);
            }
            int vec_TB2[] = {5,3,7,5,6,2,5,2};
            for(int i = 1; i <= 8; i++) {
                  caso2.t_BA(i, vec_TB2[i-1]);
            }
            // Segunda instancia //////
            LinhaDeMontagem linhaA5 = new LinhaDeMontagem(8, 5 ,8);
            int vec_A5[] = {10,6,3,8,5,3,7,12};
            for(int i = 1; i <= 8; i++) {
                  linhaA5.valorestacao(i,vec_A5[i-1]);
            }
            LinhaDeMontagem linhaA6 = new LinhaDeMontagem(8, 7 ,9);
```

```
            int vec_A6[] = {3,5,3,7,6,4,9,10};
            for(int i = 1; i <= 8; i++) {
                  linhaA6.valorestacao(i,vec_A6[i-1]);
            }
            LinhaDuplaComTransicao caso3 = new
LinhaDuplaComTransicao(linhaA5, linhaA6);

            int vec_TA3[] = {4,2,7,2,5,8,2};
            for(int i = 1; i <= 7; i++) {
                  caso3.t_AB(i, vec_TA3[i-1]);
            }
            int vec_TB3[] = {6,1,7,3,6,4,5};
            for(int i = 1; i <= 7; i++) {
                  caso3.t_BA(i, vec_TB3[i-1]);
            }
            // criar as classes de solucao
            ProgramacaoDinamica teste2 = new ProgramacaoDinamica(caso2);
            ProgramacaoDinamica teste3 = new ProgramacaoDinamica(caso3);

            Guloso teste2_Guloso = new Guloso(caso2);
            Guloso teste3_Guloso = new Guloso(caso3);
            //
            ////////////
            /// RESPOSTAS
            System.out.println("\n\n");
            System.out.println("Resposta Caso da instancia 1 Programacao
Dinamica");
            teste2.obterResposta();
            System.out.println();
            System.out.println("Instancia 1 Programacao Gulosa: ");
            teste2_Guloso.obterResposta();
            System.out.println("\n\n");

            System.out.println("Resposta Caso da instancia 2 Programacao
Dinamica");
            teste3.obterResposta();
            System.out.println();
            System.out.println("Instancia 2 Programacao Gulosa: ");
            teste3_Guloso.obterResposta();

      }
```

5. Resultados:

```
Resposta Caso Exemplo Programacao Dinamica
Linha de montagem 1, Estacao 1
Linha de montagem 2, Estacao 2
Linha de montagem 1, Estacao 3
Linha de montagem 2, Estacao 4
Linha de montagem 2, Estacao 5
Linha de montagem 1, Estacao 6
O caminho tem o valor de: 38
Resposta Caso Exemplo Guloso
Linha de montagem 1, Estacao 1
Linha de montagem 2, Estacao 2
Linha de montagem 1, Estacao 3
Linha de montagem 1, Estacao 4
Linha de montagem 1, Estacao 5
Linha de montagem 1, Estacao 6
O caminho tem o valor de: 39
```

```
Resposta Caso da instancia 1 Programacao Dinamica
Linha de montagem 2, Estacao 1
Linha de montagem 2, Estacao 2
Linha de montagem 2, Estacao 3
Linha de montagem 2, Estacao 4
Linha de montagem 2, Estacao 5
Linha de montagem 2, Estacao 6
Linha de montagem 2, Estacao 7
Linha de montagem 1, Estacao 8
Linha de montagem 1, Estacao 9
O caminho tem o valor de: 65

Instancia 1 Programacao Gulosa:
Linha de montagem 1, Estacao 1
Linha de montagem 2, Estacao 2
Linha de montagem 2, Estacao 3
Linha de montagem 2, Estacao 4
Linha de montagem 2, Estacao 5
Linha de montagem 2, Estacao 6
Linha de montagem 2, Estacao 7
Linha de montagem 1, Estacao 8
Linha de montagem 1, Estacao 9
O caminho tem o valor de: 68
```

```
Resposta Caso da instancia 2 Programacao Dinamica
Linha de montagem 2, Estacao 1
Linha de montagem 2, Estacao 2
Linha de montagem 1, Estacao 3
Linha de montagem 1, Estacao 4
Linha de montagem 1, Estacao 5
Linha de montagem 1, Estacao 6
Linha de montagem 1, Estacao 7
Linha de montagem 1, Estacao 8
O caminho tem o valor de: 62

Instancia 2 Programacao Gulosa:
Linha de montagem 2, Estacao 1
Linha de montagem 2, Estacao 2
Linha de montagem 2, Estacao 3
Linha de montagem 2, Estacao 4
Linha de montagem 2, Estacao 5
Linha de montagem 2, Estacao 6
Linha de montagem 2, Estacao 7
Linha de montagem 2, Estacao 8
O caminho tem o valor de: 63
```

6. Análise dos resultados:

Os resultados obtidos dos menores caminhos do problema das esteiras, evidenciam um comportamento já esperado, que a solução por programação dinâmica é sempre a melhor solução, é o algoritmo ótimo, já a solução por algoritmo guloso demonstra uma solução boa, porém nem sempre a solução ótima para o problema.

Ambos os algoritmos dividem o problema em pequenas partes, a diferença é que na programação dinâmica há uma memória em que se obtem o melhor resultado para um problema anterior, que será necessário em algum eventual momento, assim, com a memória garante a solução ótima.

A programação gulosa, somente considera o melhor valor para a interação atual, o que resolve o problema, porém não da melhor maneira possível.

O desafio desse projeto são as ecolhas para a representação e instanciação das linhas de montagem, o que pode ser feita de diversas formas, porém após a escolha é bem interessante implementar e ver os diferentes comportamentos da Programação Dinâmica e da Programação Gulosa.