File: ./src/com/fiuba/algoritmos/BellmanFord.java

```
1  package com.fiuba.algoritmos;
2
3  import com.fiuba.grafos.Arista;
4  import com.fiuba.grafos.Digrafo;
5
6  import java.util.*;
7
8  /**
9   * Created by marianovazquez on 6/4/17.
10   */
11  public class BellmanFord implements ShortestPathAlgorithm {
12
13      private Digrafo digrafo;
14
15      public BellmanFord(Digrafo digrafo) {
16          this.digrafo = digrafo;
17      }
18
19      /**
20       * Aplica el algoritmo de BellmanFord para obtener el camino
mínimo entre dos
21       * vértices del grafo.
22       *
23       * @param   src   el id del vértice de origen.
24       * @param   dest  el id del vértice de destino.
25       * @return  una lista con los vértices que conforman el camino,
incluyendo
26       *          el origen y el destino.
27       * @throws IllegalArgumentException si algún id de vértice es
inválido.
28       */
29      @Override
30      public List<Integer> getShortestPath(int src, int dest) {
31          Map<Integer, Integer> distancias = new HashMap<>();
32          Map<Integer, Integer> padres = new HashMap<>();
33
34          for (int i = 0; i < this.digrafo.cuentaDeVertices(); i++) {
35              distancias.put(i, 999999999);
36              padres.put(i, null);
37          }
38
39          distancias.put(src, 0);
40
41          for (int i = 0; i < this.digrafo.cuentaDeVertices() - 1;
i++) {
42              this.digrafo.getAristas().forEach(arista -> {
43                  int origen = arista.getSrc();
44                  int destino = arista.getDst();
45
46                  if (distancias.get(origen) + arista.getPeso() <
distancias.get(destino)) {
47                      distancias.put(destino, distancias.get(origen) +
arista.getPeso());
48                      padres.put(destino, origen);
49                  }
50              });
51          }
52
```

```
53          // Chequeamos que no haya ciclos negativos
54          boolean tieneCiclosNegativos = false;
55          this.digrafo.getAristas().forEach(arista -> {
56              int origen = arista.getSrc();
57              int destino = arista.getDst();
58
59              if (distancias.get(origen) + arista.getPeso() <
distancias.get(destino)) {
60                  System.out.print("El grafo contiene ciclos
negativos");
61              }
62          });
63
64          if (tieneCiclosNegativos) {
65              return new ArrayList<>();
66          }
67
68          return this.getCaminoMasCorto(src, dest, padres);
69      }
70
71      private List<Integer> getCaminoMasCorto(int src, int dest,
Map<Integer, Integer> padres) {
72          List<Integer> toReturn = new ArrayList<Integer>();
73          Integer current = dest;
74
75          while (current != null && !toReturn.contains(current)) {
76              toReturn.add(current);
77              current = padres.get(current);
78          }
79
80          Collections.reverse(toReturn);
81          return toReturn;
82      }
83  }
```

File: ./src/com/fiuba/algoritmos/Dijkstra.java

```
1  package com.fiuba.algoritmos;
2
3  import com.fiuba.grafos.Arista;
4  import com.fiuba.grafos.Digrafo;
5
6  import java.util.*;
7
8  /**
9   * Created by marianovazquez on 6/4/17.
10   */
11  public class Dijkstra implements ShortestPathAlgorithm {
12
13      private Digrafo digrafo;
14
15      public Dijkstra(Digrafo digrafo) {
16          this.digrafo = digrafo;
17      }
18
19      /**
20       * Aplica el algoritmo de Dijkstra para obtener el camino mínimo
entre dos
```

```
21       * vértices del grafo.
22       *
23       * @param   src   el id del vértice de origen.
24       * @param   dest  el id del vértice de destino.
25       * @return  una lista con los vértices que conforman el camino,
incluyendo
26       *          el origen y el destino.
27       * @throws IllegalArgumentException si algún id de vértice es
inválido.
28       */
29      @Override
30      public List<Integer> getShortestPath(int src, int dest) {
31          PriorityQueue<Integer> colaDePrioridad = new
PriorityQueue<>();
32          Map<Integer, Integer> distancias = new HashMap<>();
33          Map<Integer, Integer> padres = new HashMap<>();
34
35          for (int i = 0; i < this.digrafo.cuentaDeVertices(); i++) {
36              distancias.put(i, 999999999);
37              padres.put(i, null);
38          }
39
40          distancias.put(src, 0);
41          colaDePrioridad.add(src);
42
43          while (!colaDePrioridad.isEmpty()) {
44              int vertice = colaDePrioridad.poll();
45              Iterator<Arista> aristas =
this.digrafo.getAdjList(vertice);
46
47              while (aristas.hasNext()) {
48                  Arista arista = aristas.next();
49                  int origen = arista.getSrc();
50                  int destino = arista.getDst();
51
52                  if (distancias.get(origen) + arista.getPeso() <
distancias.get(destino)) {
53                      distancias.put(destino, distancias.get(origen) +
arista.getPeso());
54                      padres.put(destino, origen);
55                      colaDePrioridad.remove(origen);
56                      colaDePrioridad.add(destino);
57                  }
58              }
59          }
60
61          return this.getCaminoMasCorto(dest, padres);
62      }
63
64      private int getNodoConMinimaDistancia(Set<Integer>
nodosNoVisitados, Map<Integer, Integer> distancias) {
65          Integer nodoMinimo = null;
66
67          for (int nodoNoVisitado : nodosNoVisitados) {
68              if (nodoMinimo == null) {
69                  nodoMinimo = nodoNoVisitado;
70              } else {
71                  if (distancias.get(nodoNoVisitado) <
distancias.get(nodoMinimo)) {
72                      nodoMinimo = nodoNoVisitado;
73                  }
```

```
74              }
75          }
76
77          return nodoMinimo;
78      }
79
80      private List<Integer> getCaminoMasCorto(int dest, Map<Integer,
Integer> padres) {
81          List<Integer> toReturn = new ArrayList<Integer>();
82          Integer current = dest;
83
84          while (current != null) {
85              toReturn.add(current);
86              current = padres.get(current);
87          }
88
89          Collections.reverse(toReturn);
90          return toReturn;
91      }
92  }
```

File: ./src/com/fiuba/algoritmos/FloydWarshall.java

```
1   package com.fiuba.algoritmos;
2
3   import com.fiuba.grafos.Digrafo;
4
5   import java.util.ArrayList;
6   import java.util.List;
7
8   /**
9    * Created by marianovazquez on 6/4/17.
10   */
11  public class FloydWarshall implements ShortestPathAlgorithm {
12
13      private Digrafo digrafo;
14
15      public FloydWarshall(Digrafo digrafo) {
16          this.digrafo = digrafo;
17      }
18
19      @Override
20      public List<Integer> getShortestPath(int src, int dest) {
21          double[][] distancias = new
double[this.digrafo.cuentaDeVertices()][this.digrafo.cuentaDeVertices()]
;
22          Integer[][] siguientes = new
Integer[this.digrafo.cuentaDeVertices()][this.digrafo.cuentaDeVertices()
];
23
24          for (int i = 0; i < this.digrafo.cuentaDeVertices(); i++) {
25              for (int j = 0; j < this.digrafo.cuentaDeVertices();
j++) {
26                  distancias[i][j] = i == j ? 0 : 999999999;
27                  siguientes[i][j] = null;
28              }
29          }
30
```

```
31          this.digrafo.getAristas().forEach(arista -> {
32              int origen = arista.getSrc();
33              int destino = arista.getDst();
34
35              distancias[origen][destino] = arista.getPeso();
36              siguientes[origen][destino] = destino;
37          });
38
39          for (int k = 0; k < this.digrafo.cuentaDeVertices(); k++) {
40              for (int i = 0; i < this.digrafo.cuentaDeVertices();
i++) {
41                  for (int j = 0; j < this.digrafo.cuentaDeVertices();
j++) {
42                      if (distancias[i][j] > distancias[i][k] +
distancias[k][j]) {
43                          distancias[i][j] = distancias[i][k] +
distancias[k][j];
44                          siguientes[i][j] = siguientes[i][k];
45                      }
46                  }
47              }
48          }
49
50          return this.getCaminoMasCorto(src, dest, siguientes);
51      }
52
53      private List<Integer> getCaminoMasCorto(int src, int dest,
Integer[][] siguientes) {
54          ArrayList<Integer> toReturn = new ArrayList<>();
55
56          if (siguientes[src][dest] == null) {
57              return toReturn;
58          }
59
60          int current = src;
61          toReturn.add(current);
62
63          while (current != dest) {
64              current = siguientes[current][dest];
65              toReturn.add(current);
66          }
67
68          return toReturn;
69      }
70  }
71
```

File: ./src/com/fiuba/algoritmos/ShortestPathAlgorithm.java

```
1  package com.fiuba.algoritmos;
2
3  import java.util.List;
4
5  /**
6   * Created by marianovazquez on 6/4/17.
7   */
8  public interface ShortestPathAlgorithm {
9
```

```
10      /**
11       * Devuelve el camino mínimo entre dos vértices del grafo.
12       *
13       * @param   src   el id del vértice de origen.
14       * @param   dest  el id del vértice de destino.
15       * @return  una lista con los vértices que conforman el camino,
incluyendo
16       *          el origen y el destino.
17       * @throws IllegalArgumentException si algún id de vértice es
inválido.
18       */
19      public List<Integer> getShortestPath(int src, int dest);
20
21  }
```

File: ./src/com/fiuba/grafos/Arista.java

```
1   package com.fiuba.grafos;
2
3   /**
4    * Created by gatti2602 on 09/04/17.
5    * Arista Dirigida.
6    * Contiene campos src y dst inmutables al crear la arista.
7    */
8   public class Arista {
9
10      private Integer src, dst, peso;
11
12      Arista(Integer src, Integer dst, Integer peso) {
13          this.src = src;
14          this.dst = dst;
15          this.peso = peso;
16      }
17
18      public Integer getSrc() {
19          return this.src;
20      }
21
22      public Integer getDst() {
23          return this.dst;
24      }
25
26      public Integer getPeso() {
27          return this.peso;
28      }
29  }
```

File: ./src/com/fiuba/grafos/Digrafo.java

```
1   package com.fiuba.grafos;
2
3   import java.util.*;
4
5   /**
6    * Created by gatti2602 on 09/04/17.
```

```
7   * Implementa la clase Digrafo inmutable.
8   * Representa los vertices como una entrada en la lista de adyacencia
9   * Cada vertice se representa por una lista de Aristas. Los vertices
empiezan en 0
10   * El digrafo es no pesado.
11   */
12  public class Digrafo {
13
14      /**
15       * Cada elemento contiene la lista de arsitas del vertice
16       * referenciado
17       */
18      private ArrayList<HashMap<Integer, Arista>> adjList;
19      private int cuentaAristas = 0;
20      private int cuentaVertices;
21
22      public Digrafo(int vertices) {
23          this.adjList = new ArrayList<>(vertices);
24          this.cuentaVertices = vertices;
25
26          for (int i = 0; i < vertices; i++) {
27              this.adjList.add(new HashMap<>());
28          }
29      }
30
31      /**
32       * Devuelve la cantidad de Vertices del Digrafo
33       */
34      public Integer cuentaDeVertices() {
35          return cuentaVertices;
36      }
37
38      /**
39       * Devuelve la cantidad de Aristas del Grafoo
40       */
41      public Integer cuentaDeAristas() {
42          return cuentaAristas;
43      }
44
45      /**
46       * Devuelve un iterador sobre la lista de adyacencia del vertice
indicado.
47       * Devuelve null si el vertice no existe
48       */
49      public Iterator<Arista> getAdjList(int vertice) {
50          if (vertice >= adjList.size())
51              return null;
52
53          return adjList.get(vertice).values().iterator();
54
55      }
56
57      /**
58       * Devuelve las aristas del grafo
59       */
60      public List<Arista> getAristas() {
61          List<Arista> toReturn = new ArrayList<>();
62          this.adjList.forEach(adjList ->
toReturn.addAll(adjList.values()));
63
64          return toReturn;
```

```
65      }
66
67      /**
68       * Devuelve las aristas del grafo
69       */
70      public List<Arista> getAristas(int vertice) {
71          return new
ArrayList<Arista>(this.adjList.get(vertice).values());
72      }
73
74
75
76      /**
77       * Agrega una Arista si el src esta dentro de los vertices del
grafo y la arista no existe
78       */
79      public void agregarArista(int src, int dst, int peso) {
80
81          if (src >= cuentaVertices || dst >= cuentaVertices)
82              return;
83
84          Arista previa = adjList.get(src).put(dst, new Arista(src,
dst, peso));
85          if (previa == null)
86              cuentaAristas++;
87      }
88  }
```

File: ./src/com/fiuba/grafos/DigrafoLoader.java

```
1   package com.fiuba.grafos;
2
3   import java.io.IOException;
4   import java.nio.file.Files;
5   import java.nio.file.Path;
6   import java.nio.file.Paths;
7   import java.util.List;
8
9   /**
10    * Created by marianovazquez on 6/4/17.
11    */
12  public class DigrafoLoader {
13
14      public static Digrafo loadDigrafo(String nombreDelGrafo) {
15          String nombreArchivo = "data/" + nombreDelGrafo + ".txt";
16          Path pathArchivo = Paths.get(nombreArchivo);
17          Digrafo digrafo = null;
18
19          try {
20              List<String> lineas = Files.readAllLines(pathArchivo);
21              int vertices = Integer.parseInt(lineas.get(0));
22              int aristas = Integer.parseInt(lineas.get(1));
23              digrafo = new Digrafo(vertices);
24
25              for (int j = 0; j < aristas; j++) {
26                  String[] aristaInfo = lineas.get(j +
2).split("\\s+");
27                  digrafo.agregarArista(
```

```
28                      Integer.parseInt(aristaInfo[0]),
29                      Integer.parseInt(aristaInfo[1]),
30                      Integer.parseInt(aristaInfo[2])
31                  );
32              }
33          } catch(IOException e) {
34              System.out.println("Problema al leer el grafo " +
nombreDelGrafo);
35          }
36
37          return digrafo;
38      }
39  }
```

File: ./src/com/fiuba/grafos/GeneradorDeDigrafos.java

```
1   package com.fiuba.grafos;
2
3   import java.util.Random;
4
5   /**
6    * Created by marianovazquez on 6/4/17.
7    */
8   public class GeneradorDeDigrafos {
9       public static Digrafo generarDigrafoCompleto(int
numeroDeVertices) {
10          Digrafo digrafo = new Digrafo(numeroDeVertices);
11
12          for (int origen = 0; origen < numeroDeVertices; origen++) {
13              for (int destino = 0; destino < numeroDeVertices;
destino++) {
14                  if (origen != destino) {
15                      // Genero un peso random entre 1 y 100 para la
arista
16                      int peso = new Random().nextInt(100) + 1;
17                      digrafo.agregarArista(origen, destino, peso);
18                  }
19              }
20          }
21
22          return digrafo;
23      }
24  }
```

File: ./src/com/fiuba/Main.java

```
1   package com.fiuba;
2
3   import com.fiuba.algoritmos.BellmanFord;
4   import com.fiuba.algoritmos.Dijkstra;
5   import com.fiuba.algoritmos.FloydWarshall;
6   import com.fiuba.grafos.Digrafo;
7   import com.fiuba.grafos.GeneradorDeDigrafos;
8
9   import java.io.IOException;
```

```
10  import java.util.List;
11  import java.util.Random;
12  import java.util.Scanner;
13  import java.util.concurrent.TimeUnit;
14
15  public class Main {
16
17      public static void main(String[] args) {
18
19          System.out.println("-------------------------------------");
20          System.out.println("\uD83D\uDE80 \uD83D\uDE80 TP1 Teoría de
Algoritmos \uD83D\uDE80 \uD83D\uDE80");
21          System.out.println("-------------------------------------");
22          System.out.println();
23          System.out.println("Elige el número de vértices del digrafo
completo:");
24          System.out.println();
25
26          Scanner scanner = new Scanner(System.in);
27          int numeroDeVertices = Integer.parseInt(scanner.nextLine());
28          System.out.println();
29
30          Random random = new Random();
31          int origen = random.nextInt(numeroDeVertices) + 1;
32          int destino = random.nextInt(numeroDeVertices) + 1;
33
34          while (origen == destino) {
35              destino = random.nextInt(numeroDeVertices) + 1;
36          }
37
38          Digrafo digrafo =
GeneradorDeDigrafos.generarDigrafoCompleto(numeroDeVertices);
39
40          System.out.println("Bellman Ford");
41          System.out.println("------------");
42          resolveBellmanFord(digrafo,origen, destino);
43          System.out.println();
44
45          System.out.println("Dijkstra");
46          System.out.println("--------");
47          resolveDijkstra(digrafo,origen, destino);
48          System.out.println();
49
50          System.out.println("Floyd Warshall");
51          System.out.println("--------------");
52          resolveFloydWarshall(digrafo,origen, destino);
53          System.out.println();
54
55          System.out.println();
56          System.out.println("\uD83D\uDC7E \uD83D\uDC7E Fin!
\uD83D\uDC7E \uD83D\uDC7E");
57      }
58
59      public static void resolveBellmanFord(Digrafo digrafo, int
origen, int destino) {
60          BellmanFord algoritmo = new BellmanFord(digrafo);
61
62          long tiempoDeInicio = System.nanoTime();
63          List<Integer> caminoMásCorto =
algoritmo.getShortestPath(origen, destino);
64          long tiempoDelAlgoritmo = System.nanoTime() -
```

```
tiempoDeInicio;
65
66          System.out.println(
67              "BellmanFord con n=" + digrafo.cuentaDeVertices() +
68              ". Tiempo de Ejecucion: " +
TimeUnit.NANOSECONDS.toMillis(tiempoDelAlgoritmo) + " mSeg."
69          );
70      }
71
72      public static void resolveDijkstra(Digrafo digrafo, int origen,
int destino) {
73          Dijkstra algoritmo = new Dijkstra(digrafo);
74
75          long tiempoDeInicio = System.nanoTime();
76          List<Integer> caminoMásCorto =
algoritmo.getShortestPath(origen, destino);
77          long tiempoDelAlgoritmo = System.nanoTime() -
tiempoDeInicio;
78
79          System.out.println(
80              "Dijkstra con n=" + digrafo.cuentaDeVertices() +
81              ". Tiempo de Ejecucion: " +
TimeUnit.NANOSECONDS.toMillis(tiempoDelAlgoritmo) + " mSeg."
82          );
83      }
84
85      public static void resolveFloydWarshall(Digrafo digrafo, int
origen, int destino) {
86          FloydWarshall algoritmo = new FloydWarshall(digrafo);
87
88          long tiempoDeInicio = System.nanoTime();
89          List<Integer> caminoMásCorto =
algoritmo.getShortestPath(origen, destino);
90          long tiempoDelAlgoritmo = System.nanoTime() -
tiempoDeInicio;
91
92          System.out.println(
93              "FloydWarshall con n=" + digrafo.cuentaDeVertices() +
94              ". Tiempo de Ejecucion: " +
TimeUnit.NANOSECONDS.toMillis(tiempoDelAlgoritmo) + " mSeg."
95          );
96      }
97  }
```

File: ./src/com/fiuba/tests/BellmanFordTest.java

```
1  package com.fiuba.tests;
2
3  import com.fiuba.algoritmos.BellmanFord;
4  import com.fiuba.grafos.Digrafo;
5  import com.fiuba.grafos.DigrafoLoader;
6  import org.junit.Test;
7
8  import java.util.ArrayList;
9  import java.util.Arrays;
10  import java.util.List;
11
12  import static org.junit.Assert.assertEquals;
```

```
13
14  /**
15   * Created by marianovazquez on 6/4/17.
16   */
17  public class BellmanFordTest {
18
19      @Test
20      public void getShortestPath_d1() {
21          Digrafo digrafo = DigrafoLoader.loadDigrafo("d1");
22          BellmanFord algoritmo = new BellmanFord(digrafo);
23
24          List<Integer> result1 = algoritmo.getShortestPath(0, 1);
25          List<Integer> expected1 = new
ArrayList<Integer>(Arrays.asList(0, 1));
26          assertEquals(expected1, result1);
27
28          List<Integer> result2 = algoritmo.getShortestPath(0, 2);
29          List<Integer> expected2 = new
ArrayList<Integer>(Arrays.asList(0, 1, 2));
30          assertEquals(expected2, result2);
31
32          List<Integer> result3 = algoritmo.getShortestPath(1, 0);
33          List<Integer> expected3 = new
ArrayList<Integer>(Arrays.asList(1, 2, 0));
34          assertEquals(expected3, result3);
35      }
36
37      @Test
38      public void getShortestPath_d2() {
39          Digrafo digrafo = DigrafoLoader.loadDigrafo("d2");
40          BellmanFord algoritmo = new BellmanFord(digrafo);
41
42          List<Integer> result1 = algoritmo.getShortestPath(0, 1);
43          List<Integer> expected1 = new
ArrayList<Integer>(Arrays.asList(0, 1));
44          assertEquals(expected1, result1);
45
46          List<Integer> result2 = algoritmo.getShortestPath(0, 2);
47          List<Integer> expected2 = new
ArrayList<Integer>(Arrays.asList(0, 1, 2));
48          assertEquals(expected2, result2);
49
50          List<Integer> result3 = algoritmo.getShortestPath(2, 3);
51          List<Integer> expected3 = new
ArrayList<Integer>(Arrays.asList(2, 1, 0, 3));
52          assertEquals(expected3, result3);
53      }
54  }
```

File: ./src/com/fiuba/tests/DijkstraTest.java

```
1  package com.fiuba.tests;
2
3  import com.fiuba.algoritmos.Dijkstra;
4  import com.fiuba.grafos.Digrafo;
5  import com.fiuba.grafos.DigrafoLoader;
6  import org.junit.Test;
7
```

```
8  import java.util.ArrayList;
9  import java.util.Arrays;
10  import java.util.List;
11
12  import static org.junit.Assert.assertEquals;
13  import static org.junit.Assert.assertTrue;
14
15  /**
16   * Created by marianovazquez on 6/4/17.
17   */
18  public class DijkstraTest {
19
20      @Test
21      public void getShortestPath_d1() {
22          Digrafo digrafo = DigrafoLoader.loadDigrafo("d1");
23          Dijkstra algoritmo = new Dijkstra(digrafo);
24
25          List<Integer> result1 = algoritmo.getShortestPath(0, 1);
26          List<Integer> expected1 = new
ArrayList<Integer>(Arrays.asList(0, 1));
27          assertEquals(expected1, result1);
28
29          List<Integer> result2 = algoritmo.getShortestPath(0, 2);
30          List<Integer> expected2 = new
ArrayList<Integer>(Arrays.asList(0, 1, 2));
31          assertEquals(expected2, result2);
32
33          List<Integer> result3 = algoritmo.getShortestPath(1, 0);
34          List<Integer> expected3 = new
ArrayList<Integer>(Arrays.asList(1, 2, 0));
35          assertEquals(expected3, result3);
36      }
37
38      @Test
39      public void getShortestPath_d2() {
40          Digrafo digrafo = DigrafoLoader.loadDigrafo("d2");
41          Dijkstra algoritmo = new Dijkstra(digrafo);
42
43          List<Integer> result1 = algoritmo.getShortestPath(0, 1);
44          List<Integer> expected1 = new
ArrayList<Integer>(Arrays.asList(0, 1));
45          assertEquals(expected1, result1);
46
47          List<Integer> result2 = algoritmo.getShortestPath(0, 2);
48          List<Integer> expected2 = new
ArrayList<Integer>(Arrays.asList(0, 1, 2));
49          assertEquals(expected2, result2);
50
51          List<Integer> result3 = algoritmo.getShortestPath(2, 3);
52          List<Integer> expected3 = new
ArrayList<Integer>(Arrays.asList(2, 1, 0, 3));
53          assertEquals(expected3, result3);
54      }
55  }
```

File: ./src/com/fiuba/tests/FloydWarshallTest.java

```
1  package com.fiuba.tests;
```

```
2
3  import com.fiuba.algoritmos.FloydWarshall;
4  import com.fiuba.grafos.Digrafo;
5  import com.fiuba.grafos.DigrafoLoader;
6  import org.junit.Test;
7
8  import java.util.ArrayList;
9  import java.util.Arrays;
10  import java.util.List;
11
12  import static org.junit.Assert.assertEquals;
13
14  /**
15   * Created by marianovazquez on 6/4/17.
16   */
17  public class FloydWarshallTest {
18
19      @Test
20      public void getShortestPath_d1() {
21          Digrafo digrafo = DigrafoLoader.loadDigrafo("d1");
22          FloydWarshall algoritmo = new FloydWarshall(digrafo);
23
24          List<Integer> result1 = algoritmo.getShortestPath(0, 1);
25          List<Integer> expected1 = new
ArrayList<Integer>(Arrays.asList(0, 1));
26          assertEquals(expected1, result1);
27
28          List<Integer> result2 = algoritmo.getShortestPath(0, 2);
29          List<Integer> expected2 = new
ArrayList<Integer>(Arrays.asList(0, 1, 2));
30          assertEquals(expected2, result2);
31
32          List<Integer> result3 = algoritmo.getShortestPath(1, 0);
33          List<Integer> expected3 = new
ArrayList<Integer>(Arrays.asList(1, 2, 0));
34          assertEquals(expected3, result3);
35      }
36
37      @Test
38      public void getShortestPath_d2() {
39          Digrafo digrafo = DigrafoLoader.loadDigrafo("d2");
40          FloydWarshall algoritmo = new FloydWarshall(digrafo);
41
42          List<Integer> result1 = algoritmo.getShortestPath(0, 1);
43          List<Integer> expected1 = new
ArrayList<Integer>(Arrays.asList(0, 1));
44          assertEquals(expected1, result1);
45
46          List<Integer> result2 = algoritmo.getShortestPath(0, 2);
47          List<Integer> expected2 = new
ArrayList<Integer>(Arrays.asList(0, 1, 2));
48          assertEquals(expected2, result2);
49
50          List<Integer> result3 = algoritmo.getShortestPath(2, 3);
51          List<Integer> expected3 = new
ArrayList<Integer>(Arrays.asList(2, 1, 0, 3));
52          assertEquals(expected3, result3);
53      }
54  }
```