File: ./src/com/fiuba/galeShapley/GaleShapley.java

```
1  package com.fiuba.galeShapley;
2
3  import java.util.ArrayList;
4  import java.util.HashMap;
5  import java.util.Set;
6  import java.util.Stack;
7
8  /**
9   * Created by marianovazquez on 4/22/17.
10   */
11  public class GaleShapley {
12
13      HashMap<Integer, ArrayList<Integer>>
listaDePreferenciasPretendientes;
14      HashMap<Integer, ArrayList<Integer>>
listaDePreferenciasOferentes;
15      HashMap<Integer, Integer> parejas;
16
17      public GaleShapley(HashMap<Integer, ArrayList<Integer>>
listaDePreferenciasPretendientes, HashMap<Integer, ArrayList<Integer>>
listaDePreferenciasOferentes) {
18          this.listaDePreferenciasPretendientes =
listaDePreferenciasPretendientes;
19          this.listaDePreferenciasOferentes =
listaDePreferenciasOferentes;
20          this.parejas = new HashMap<>();
21          this.calcularParejas();
22      }
23
24      public HashMap<Integer, Integer> getParejas()  {
25
26          return this.parejas;
27      }
28
29      private void calcularParejas() {
30
31          Stack<Integer> pretendientesLibres = new Stack<>();
32
pretendientesLibres.addAll(listaDePreferenciasPretendientes.keySet());
33
34          HashMap<Integer, Stack<Integer>>
pretendientesPosiblesOferentes = new HashMap<>();
35          for (int i = 0; i < listaDePreferenciasPretendientes.size();
i++) {
36              ArrayList<Integer> preferenciaPretendiente =
listaDePreferenciasPretendientes.get(i);
37              Stack<Integer> posiblesOferentes = new Stack<>();
38              posiblesOferentes.addAll(preferenciaPretendiente);
39              pretendientesPosiblesOferentes.put(i,
posiblesOferentes);
40          }
41
42          Integer[] actualPretendienteDeOferentes = new
Integer[listaDePreferenciasOferentes.size()];
43
44          // Dado que el escenario es cuadrado (n pretendientes, n
oferentes)
45          // y todos los pretendientes tienen a las oferentes en su
```

```
lista de preferencia (y viceversa)
46          // puedo asumir que no se va a dar el caso donde un
pretendiente quede libre aún luego de proponersele a todos los oferentes
47          while (!pretendientesLibres.empty()) {
48
49              int pretendiente = pretendientesLibres.pop();
50              int oferente =
pretendientesPosiblesOferentes.get(pretendiente).pop();
51
52              // Chequeo si el oferente en pareja con alguien
53              Integer actualPretendienteDeOferente =
actualPretendienteDeOferentes[oferente];
54
55              if (actualPretendienteDeOferente == null) {
56
57                  // No está en pareja. Agrego la pareja de oferente-
pretendiente
58                  parejas.put(pretendiente, oferente);
59                  actualPretendienteDeOferentes[oferente] =
pretendiente;
60              } else {
61
62                  // Chequeo si el pretendiente es mejor pareja que la
actual
63                  ArrayList<Integer> preferenciasOferente =
this.listaDePreferenciasOferentes.get(oferente);
64                  if (preferenciasOferente.indexOf(pretendiente) >
preferenciasOferente.indexOf(actualPretendienteDeOferente)) {
65
66                      // Es mejor, libero al pretendiente anterior
67                      parejas.remove(actualPretendienteDeOferente);
68
pretendientesLibres.push(actualPretendienteDeOferente);
69
70                      // Y agrego la pareja de oferente-pretendiente
71                      parejas.put(pretendiente, oferente);
72                      actualPretendienteDeOferentes[oferente] =
pretendiente;
73                  } else {
74                      // No encontré pareja, el pretendiente sigue
libre
75                      pretendientesLibres.push(pretendiente);
76                  }
77              }
78          }
79      }
80  }
```

File: ./src/com/fiuba/galeShapley/HospitalRandomizer.java

```
1  package com.fiuba.galeShapley;
2
3  import java.io.*;
4  import java.net.URI;
5  import java.nio.file.Paths;
6  import java.util.ArrayList;
7  import java.util.Iterator;
8  import java.util.Random;
```

```
9
10  /**
11   * Created by lt5420 on 22/04/2017.
12   * Genera listas Aleatorias de numeros enteros
13   */
14  public class HospitalRandomizer {
15
16      private Integer cantEstudiantes, cantHospitales;
17
18      public HospitalRandomizer(int cantEstudiantes, int
cantHospitales) {
19          this.cantEstudiantes = cantEstudiantes;
20          this.cantHospitales = cantHospitales;
21      }
22
23      public void toFiles(String filename) {
24          try {
25              File file = new File(filename);
26              if (file.exists())
27                  file.delete();
28              file.createNewFile();
29
30              FileWriter fw = new FileWriter(file.getAbsoluteFile());
31              BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(new FileOutputStream(file), "Cp1252"));
32              //Imprimo data estudiantes
33              bw.write(cantEstudiantes.toString());
34              bw.newLine();
35              for (int i = 0; i < cantEstudiantes; i++){
36
bw.write(arrayListToString(Randomizer.CreateRandomUniqueVector(cantHospi
tales).iterator()));
37                  bw.newLine();
38                  if(i %1000 == 0)
39                      bw.flush();
40              }
41                  bw.flush();
42              //Imprimo data hospitales
43              bw.write(cantHospitales.toString());
44              bw.newLine();
45              for (int i = 0; i < cantHospitales; i++){
46
bw.write(arrayListToString(Randomizer.CreateRandomUniqueVector(cantEstud
iantes).iterator()));
47                  bw.newLine();
48                  if(i %1000 == 0)
49                      bw.flush();
50              }
51
52              //Imprimo capacidad Hospitales
53
bw.write(arrayListToString(Randomizer.CreateRandomVector(cantHospitales,
cantEstudiantes).iterator()));
54              bw.close();
55          } catch (IOException e) {
56              System.out.println("Error: " + e.getMessage());
57          }
58      }
59      private String arrayListToString(Iterator<Integer>it){
60              StringBuilder st = new StringBuilder();
61
```

```
            if(it.hasNext())
                st.append(it.next());

            while(it.hasNext()) {
                st.append(" ");
                st.append(it.next());
            }
            return st.toString();
        }

    }


```

File: ./src/com/fiuba/galeShapley/Randomizer.java

```
package com.fiuba.galeShapley;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Random;

/**
 * Created by lt5420 on 22/04/2017.
 */
public class Randomizer {
    public static ArrayList<Integer> CreateRandomVector(Integer m, Integer randLimit){
        Random r = new Random();
        ArrayList<Integer> aux = new ArrayList<>(m);
        for (int j=0;j<m;j++) {
            if (randLimit != null)
                aux.add(r.nextInt(randLimit));
            else
                aux.add(r.nextInt());
        }
        return aux;
    }

    /**
     * Carga una Matriz de n+m cuyos valores enteros no superan randLimit
     * @param n
     * @param m
     * @param randLimit
     * @return
     */
    public static ArrayList<ArrayList<Integer>>CreateRandomMatrix(Integer n, Integer m, Integer randLimit){
        ArrayList<ArrayList<Integer>> matrix = new ArrayList<>(n);
        for (int i=0; i<n;i++)
            matrix.add(Randomizer.CreateRandomVector(m, randLimit));
        return matrix;
    }

    public static HashMap<Integer,ArrayList<Integer>>
```

```
CreateRandomUniqueMatrix(Integer n, Integer m){
39          HashMap<Integer, ArrayList<Integer>> matrix = new
HashMap<>(n);
40          for (int i=0; i<n;i++)
41              matrix.put(i,Randomizer.CreateRandomUniqueVector(m));
42          return matrix;
43      }
44
45      public static ArrayList<Integer>
CreateRandomUniqueVector(Integer n){
46          ArrayList<Integer> aux = new ArrayList<>();
47          for(int i=0; i<n;i++)
48              aux.add(i);
49          Collections.shuffle(aux);
50          return aux;
51      }
52  }
53
54
```

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;
11
12      Arista(Integer src, Integer dst) {
13          this.src = src;
14          this.dst = dst;
15      }
16
17      public Integer getSrc() {
18          return src;
19      }
20
21      public Integer getDst() {
22          return dst;
23      }
24
25
26  }
```

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

```
1   package com.fiuba.grafos;
2
3   import java.util.ArrayList;
4   import java.util.HashMap;
```

```
5  import java.util.Iterator;
6
7  /**
8   * Created by gatti2602 on 09/04/17.
9   * Implementa la clase Digrafo inmutable.
10   * Representa los vertices como una entrada en la lista de
adyacencia
11   * Cada vertice se representa por una lista de Aristas. Los vertices
empiezan en 0
12   * El digrafo es no pesado.
13   */
14  public class Digrafo {
15
16      /**
17       * Cada elemento contiene la lista de arsitas del vertice
18       * referenciado
19       */
20      private ArrayList<HashMap<Integer, Arista>> adjList;
21      private int cuentaAristas = 0;
22      private int cuentaVertices;
23
24      public Digrafo(int vertices) {
25          this.adjList = new ArrayList<>(vertices);
26          this.cuentaVertices = vertices;
27
28          for (int i = 0; i < vertices; i++) {
29              this.adjList.add(new HashMap<>());
30          }
31      }
32
33      /**
34       * Devuelve la cantidad de Vertices del Grafo
35       */
36      public Integer cuentaDeVertices() {
37          return cuentaVertices;
38      }
39
40      /**
41       * Devuelve la cantidad de Aristas del Grafoo
42       */
43      public Integer cuentaDeAristas() {
44          return cuentaAristas;
45      }
46
47      /**
48       * Devuelve un iterador sobre la lista de adyacencia del vertice
indicado.
49       * Devuelve null si el vertice no existe
50       */
51      public Iterator<Arista> getAdjList(int vertice) {
52          if (vertice >= adjList.size())
53              return null;
54
55          return adjList.get(vertice).values().iterator();
56
57      }
58
59      /**
60       * Agrega una Arista si el src esta dentro de los vertices del
grafo y la arista no existe
61       */
```

```
62      public void agregarArista(int src, int dst) {
63
64          if (src >= cuentaVertices || dst >= cuentaVertices)
65              return;
66
67          Arista previa = adjList.get(src).put(dst, new Arista(src,
dst));
68          if (previa == null)
69              cuentaAristas++;
70      }
71
72      public Boolean existeArista(int src, int dst) {
73          Boolean existe = false;
74          if (src < cuentaVertices) {
75              existe = this.adjList.get(src).containsKey(dst);
76          }
77          return existe;
78      }
79
80      public Digrafo transponer() {
81          Digrafo d = new Digrafo(this.cuentaDeVertices());
82
83          for (int src = 0; src < this.cuentaDeVertices(); src++) {
84              for (int dst = 0; dst < this.cuentaDeVertices(); dst++)
{
85                  if (!this.existeArista(src, dst))
86                      d.agregarArista(src, dst);
87              }
88          }
89          return d;
90      }
91
92      public Digrafo invertirArcos() {
93          Digrafo d = new Digrafo(this.cuentaDeVertices());
94
95          for (int i = 0; i < this.cuentaDeVertices(); i++) {
96              //System.out.println("Transponiendo vertice: " + i);
97              Iterator<Arista> it = this.getAdjList(i);
98              while (it.hasNext()) {
99                  Arista aux = it.next();
100                  d.agregarArista(aux.getDst(), aux.getSrc());
101              }
102          }
103          return d;
104      }
105  }
```

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

```
1   package com.fiuba.grafos;
2
3   import java.util.Iterator;
4
5   /**
6    * Created by marianovazquez on 4/19/17.
7    */
8   public class Grafo {
9
```

```
10      private Digrafo digrafo;
11
12      public Grafo(int vertices) {
13          this.digrafo = new Digrafo(vertices);
14      }
15
16      /**
17       * Devuelve la cantidad de Vertices del Grafo
18       */
19      public Integer getCantidadDeVertices() {
20
21          return this.digrafo.cuentaDeVertices();
22      }
23
24      /**
25       * Devuelve la cantidad de Aristas del Grafoo
26       */
27      public Integer getCantidadDeAristas() {
28
29          return this.digrafo.cuentaDeAristas() / 2;
30      }
31
32      /**
33       * Devuelve un iterador sobre la lista de adyacencia del vertice
indicado.
34       * Devuelve null si el vertice no existe
35       */
36      public Iterator<Arista> getAdyacentes(int vertice) {
37
38          return this.digrafo.getAdjList(vertice);
39      }
40
41      /**
42       * Agrega una Arista si el src esta dentro de los vertices del
grafo y la arista no existe
43       */
44      public void agregarArista(int src, int dest) {
45
46          this.digrafo.agregarArista(src, dest);
47          this.digrafo.agregarArista(dest, src);
48      }
49  }
```

File: ./src/com/fiuba/kosaraju/Kosaraju.java

```
1   package com.fiuba.kosaraju;
2
3   import com.fiuba.grafos.Digrafo;
4
5   import java.io.IOException;
6   import java.nio.file.Files;
7   import java.nio.file.Path;
8   import java.util.ArrayList;
9   import java.util.Collections;
10   import java.util.Iterator;
11   import java.util.List;
12   import java.util.concurrent.TimeUnit;
13
```

```
14  /**
15   * Created by gatti2602 on 16/04/17.
16   */
17  public class Kosaraju {
18      private Digrafo d;
19      private ArrayList<ArrayList<Integer>> CFC;
20      private long totalTime;
21
22      public Kosaraju(Path path) {
23          List<String> file = new ArrayList<>();
24          long time = System.nanoTime();
25
26          try {
27              file = Files.readAllLines(path);
28          } catch (IOException e) {
29              e.printStackTrace();
30          }
31          time = System.nanoTime() - time;
32          time = TimeUnit.NANOSECONDS.toMillis(time);
33          //System.out.println("Kosaraju - File " + path.toString() +
" - Parseo Completo en " + time + " mSeg.");
34
35          time = System.nanoTime();
36          Digrafo d = new Digrafo(Integer.parseInt(file.get(0)));
37          for (int i = 2; i < file.size(); i++) {
38              String[] aux = file.get(i).split(" ");
39              d.agregarArista(Integer.parseInt(aux[0]),
Integer.parseInt(aux[1]));
40          }
41          time = System.nanoTime() - time;
42          time = TimeUnit.NANOSECONDS.toSeconds(time);
43          //System.out.println("Kosaraju - File " + path.toString() +
" - Digrafo Completo en " + time + " mSeg.");
44          System.gc();
45
46          this.calcKosaraju(d);
47      }
48
49      private void calcKosaraju(Digrafo d) {
50          this.d = d;
51          long time = System.nanoTime();
52          totalTime = time;
53          ArrayList<ArrayList<Integer>> componentes = new
RecorredorDeGrafo(d).DFS();
54
55          time = System.nanoTime() - time;
56          time = TimeUnit.NANOSECONDS.toMillis(time);
57          //System.out.println("Kosaraju - Algo " + " - Primer DFS
Completo en " + time + " mSeg.");
58          time = System.nanoTime();
59          //Ordenar componentes
60          ArrayList<Integer> orden = new ArrayList<>();
61          Iterator<ArrayList<Integer>> componente =
componentes.iterator();
62          while (componente.hasNext()) {
63              Iterator<Integer> vertices =
componente.next().iterator();
64              while (vertices.hasNext()) {
65                  Integer v = vertices.next();
66                  orden.add(v);
67              }
```

```
68          }
69
70          Collections.reverse(orden); //O(n)
71          time = System.nanoTime() - time;
72          time = TimeUnit.NANOSECONDS.toMillis(time);
73          //System.out.println("Kosaraju - Algo " + " - Ordenamiento
Completo en " + time + " mSeg.");
74          time = System.nanoTime();
75
76          Digrafo t = d.invertirArcos();
77          time = System.nanoTime() - time;
78          time = TimeUnit.NANOSECONDS.toMillis(time);
79          //System.out.println("Kosaraju - Algo " + " - Inversion en "
+ time + " mSeg.");
80          time = System.nanoTime();
81
82          this.CFC = new RecorredorDeGrafo(t).DFS(null, orden);
83          time = System.nanoTime() - time;
84          totalTime = System.nanoTime() - totalTime;
85          totalTime = TimeUnit.NANOSECONDS.toMillis(totalTime);
86          time = TimeUnit.NANOSECONDS.toMillis(time);
87          System.out.println("Kosaraju - Algo Vertices: "+
this.d.cuentaDeVertices() + " Aristas: " + this.d.cuentaDeAristas() +
88                  " Finalizado OK en " + totalTime + " mSeg.");
89          System.out.println("Kosaraju - Algo " + "Componentes
fuertemente conexos: " + CFC.size());
90      }
91
92      public Integer cuentaComponentesConexas() {
93          return this.CFC.size();
94      }
95
96      public Digrafo getDigrafo() {
97          return d;
98      }
99
100      public ArrayList<Integer>
getComponenteFuertementeConexa(Integer componente) {
101          return (componente >= CFC.size()) ? null :
CFC.get(componente);
102      }
103  }
```

File: ./src/com/fiuba/kosaraju/RecorredorDeGrafo.java

```
1  package com.fiuba.kosaraju;
2
3  import com.fiuba.grafos.Arista;
4  import com.fiuba.grafos.Digrafo;
5
6  import java.util.ArrayList;
7  import java.util.Iterator;
8
9  /**
10   * Created by marianovazquez on 4/22/17.
11   */
12  public class RecorredorDeGrafo {
13
```

```
14      Digrafo digrafo;
15
16      public RecorredorDeGrafo(Digrafo digrafo) {
17          this.digrafo = digrafo;
18      }
19
20      /**
21       * Devuelve una lista con los vertices visitados desde el src
22       *
23       * @param src Si es null inicia del vertice 0, si es mayor o
igual a cuentaVertices devuelve null
24       * @param ordenDeVertices Opcional, indica el orden en que se
deben recorrer los vertices al terminar la
25       *                           exploracion de todos los caminos
posibles del vertice actual.
26       * @return Devuelve una lista de listas de vertices. Cada lista
es una compponente conexa visitada con DFS
27       */
28      public ArrayList<ArrayList<Integer>> DFS(Integer src,
ArrayList<Integer> ordenDeVertices) {
29
30          if (src == null)
31              src = (ordenDeVertices != null) ? ordenDeVertices.get(0)
: 0;
32
33          if (ordenDeVertices != null && ordenDeVertices.size() !=
this.digrafo.cuentaDeVertices())
34              ordenDeVertices = null;
35
36          if (src >= this.digrafo.cuentaDeVertices())
37              return null;
38
39          ArrayList<ArrayList<Integer>> listaVerticesVisitados = new
ArrayList<>();
40          boolean[] verticeVisitado = new
boolean[this.digrafo.cuentaDeVertices()];
41
42          //Hago DFS desde el vertice indicado
43          ArrayList<Integer> componenteConexa1 = new ArrayList<>();
44          listaVerticesVisitados.add(componenteConexa1);
45          DFS_Visitar(src, componenteConexa1, verticeVisitado);
46
47          //Hago DFS desde los demas vertices
48          for (int i = 0; i < this.digrafo.cuentaDeVertices(); i++) {
49              //Si hay orden de vertices voy chequeando en ese orden,
sino orden natural.
50              int proximoVertice = (ordenDeVertices != null) ?
ordenDeVertices.get(i) : i;
51
52              if (!verticeVisitado[proximoVertice]) {
53                  ArrayList<Integer> componenteConexa = new
ArrayList<>();
54                  listaVerticesVisitados.add(componenteConexa);
55                  DFS_Visitar(proximoVertice, componenteConexa,
verticeVisitado);
56              }
57
58          }
59          return listaVerticesVisitados;
60      }
61
```

```
62      public ArrayList<ArrayList<Integer>> DFS() {
63          return DFS(null, null);
64      }
65
66      private void DFS_Visitar(Integer v, ArrayList<Integer>
listaVerticesVisitados, boolean[] verticeVisitado) {
67
68          verticeVisitado[v] = true;
69          listaVerticesVisitados.add(v);
70
71          //El orden de visita depende de este iterador
72          Iterator<Arista> it = this.digrafo.getAdjList(v);
73          while (it.hasNext()) {
74              Arista a = it.next();
75              if (!verticeVisitado[a.getDst()]) {
76                  DFS_Visitar(a.getDst(), listaVerticesVisitados,
verticeVisitado);
77              }
78          }
79      }
80  }
```

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

```
1  package com.fiuba.main;
2
3  import com.fiuba.galeShapley.GaleShapley;
4  import com.fiuba.galeShapley.HospitalRandomizer;
5  import static com.fiuba.galeShapley.Randomizer.*;
6
7  import com.fiuba.galeShapley.Randomizer;
8  import com.fiuba.grafos.Grafo;
9  import com.fiuba.kosaraju.Kosaraju;
10  import com.fiuba.tarjan.Tarjan;
11
12  import java.io.IOException;
13  import java.nio.file.Files;
14  import java.nio.file.Path;
15  import java.nio.file.Paths;
16  import java.util.*;
17  import java.util.concurrent.TimeUnit;
18
19  public class Main {
20
21      public static void main(String[] args) throws IOException {
22
23          System.out.println("-------------------------------------");
24          System.out.println("\uD83D\uDE80 \uD83D\uDE80 TP1 Teoría de
Algoritmos \uD83D\uDE80 \uD83D\uDE80");
25          System.out.println("-------------------------------------");
26          System.out.println();
27          System.out.println("Elige el algoritmo a correr:");
28          System.out.println();
29          System.out.println("1. Asignación de residencias");
30          System.out.println("2. Puntos de falla");
31          System.out.println("3. Comunidades en redes");
32          System.out.println("4. Exportar Archivos Paciente -
Hospitales");
```

```
33          System.out.println();
34
35          Scanner scanner = new Scanner(System.in);
36          int opcion = Integer.parseInt(scanner.nextLine());
37          System.out.println();
38
39          switch(opcion) {
40              case 1:
41                  System.out.println("Asignación de residencias");
42                  System.out.println("-------------------------");
43                  System.out.println();
44                  resolveResidences();
45                  break;
46              case 2:
47                  System.out.println("Puntos de Falla");
48                  System.out.println("---------------");
49                  System.out.println();
50                  resolveTarjan();
51                  break;
52              case 3:
53                  System.out.println("Comunidades en redes");
54                  System.out.println("--------------------");
55                  System.out.println();
56                  resolveKosaraju();
57                  break;
58              case 4:
59                  System.out.println("Exportar Casos Estudiantes-
Hospitales");
60                  System.out.println("--------------------");
61                  System.out.println();
62                  exportResidencesInstance();
63                  break;
64
65          }
66
67          System.out.println();
68          System.out.println("\uD83D\uDC7E \uD83D\uDC7E Fin!
\uD83D\uDC7E \uD83D\uDC7E");
69      }
70
71      public static void exportResidencesInstance() {
72          System.out.println("Cantidad de Estudiantes: ");
73          Scanner scanner1 = new Scanner(System.in);
74          int est = Integer.parseInt(scanner1.nextLine());
75          System.out.println("Cantidad de Hospitales: ");
76          Scanner scanner = new Scanner(System.in);
77          int hosp = Integer.parseInt(scanner.nextLine());
78
79          HospitalRandomizer hR = new HospitalRandomizer(est, hosp);
80          String filename = "Hospitales.txt";
81          hR.toFiles(filename);
82          System.out.println("Se genero file: " + filename);
83
84
85
86      }
87
88      public static void resolveResidences(){
89
90          for(int i=100;i<=10000;i*=10) {
91              HashMap<Integer, ArrayList<Integer>>
```

```
listaDePreferenciasPretendientes = CreateRandomUniqueMatrix(i, i);
92              HashMap<Integer, ArrayList<Integer>>
listaDePreferenciasOferentes = CreateRandomUniqueMatrix(i, i);
93
94              long tiempoDelAlgoritmo = System.nanoTime();
95              GaleShapley gs = new
GaleShapley(listaDePreferenciasPretendientes,
listaDePreferenciasOferentes);
96              tiempoDelAlgoritmo = System.nanoTime() -
tiempoDelAlgoritmo;
97              System.out.println( "Gale Shapley n=m=" +
String.valueOf(i) + ". Tiempo de Ejecucion: " +
98
TimeUnit.NANOSECONDS.toMillis(tiempoDelAlgoritmo) + " mSeg.");
99          }
100
101     }
102
103     public static void resolveTarjan() {
104
105         int cantidadDeArchivos = 6;
106
107         for (int i = 0; i < cantidadDeArchivos; i++) {
108             String nombreArchivo = "data/tarjan/d" + (i + 1) +
".txt";
109             Path pathArchivo = Paths.get(nombreArchivo);
110
111             try {
112                 List<String> lineas =
Files.readAllLines(pathArchivo);
113                 int vertices = Integer.parseInt(lineas.get(0));
114                 int aristas = Integer.parseInt(lineas.get(1));
115                 Grafo grafo = new Grafo(vertices);
116
117                 for (int j = 2; j < aristas; j++) {
118                     String[] aristaInfo =
lineas.get(j).split("\\s+");
119
grafo.agregarArista(Integer.parseInt(aristaInfo[0]),
Integer.parseInt(aristaInfo[1]));
120                 }
121
122                 System.out.println((i + 1) + ": Grafo creado con "
+ grafo.getCantidadDeVertices() + " vertices y " +
grafo.getCantidadDeAristas() + " aristas.");
123
124                 try {
125                     long tiempoInicio = System.nanoTime();
126                     Tarjan tarjan = new Tarjan(grafo);
127                     long tiempoDelAlgoritmo = System.nanoTime() -
tiempoInicio;
128
129                     Set<Integer> puntosDeArticulacion =
tarjan.getArticulationPoints();
130
131                     System.out.println("Puntos de articulación: " +
puntosDeArticulacion.toString());
132                     System.out.println("Tiempo de algoritmo: " +
TimeUnit.NANOSECONDS.toMillis(tiempoDelAlgoritmo) + " (ms) - " +
tiempoDelAlgoritmo + "(ns)");
133                 } catch(Exception e) {
```

```
134                     System.out.println("Excepción al recorrer
grafo: " + e.getMessage());
135                 } catch(Error e) {
136                     System.out.println("Error al recorrer grafo: "
+ e.getMessage());
137                 }
138 
139                 System.out.println();
140 
141             } catch(IOException e) {
142                 System.out.println("Excepción al leer archivo" +
nombreArchivo);
143             }
144         }
145     }
146 
147     public static void resolveKosaraju() {
148         for(int i=1;i<7;i++) {
149             Path path =
Paths.get("data/Kosaraju/d"+String.valueOf(i)+".txt");
150             Kosaraju k = new Kosaraju(path);
151         }
152     }
153 }
```

File: ./src/com/fiuba/tarjan/Tarjan.java

```
1  package com.fiuba.tarjan;
2 
3  import java.util.*;
4 
5  import com.fiuba.grafos.Arista;
6  import com.fiuba.grafos.Grafo;
7 
8  /**
9   * Created by marianovazquez on 4/19/17.
10   */
11 public class Tarjan {
12 
13     private Grafo grafo;
14     private Set<Integer> articulationPoints;
15 
16     public Tarjan(Grafo g) {
17         this.grafo = g;
18         this.articulationPoints = new HashSet<Integer>();
19         this.calculateArticulationPoints();
20     }
21 
22     /**
23      * Devuelve el conjunto de vértices que son puntos de
articulación del grafo.
24      * @return   El conjunto de vértices que son puntos de
articulación.
25      */
26     public Set<Integer> getArticulationPoints()  {
27 
28         return this.articulationPoints;
29     }
```

```
30
31      private void calculateArticulationPoints() {
32
33          // Inicio el recorrido del grafo en un vértice predefinido
34          int numeroVisita = 0;
35          VerticeTarjan inicio = new VerticeTarjan(0, -1,
numeroVisita, numeroVisita);
36
37          Map<Integer, VerticeTarjan> infoVerticesVisitados = new
HashMap<>();
38          infoVerticesVisitados.put(inicio.id, inicio);
39          this.findArticulationPoints(inicio, numeroVisita,
infoVerticesVisitados);
40      }
41
42      /**
43       * Se recorre el grafo usando DFS a fin de identificar los
puntos de articulación del grafo.
44       */
45      private void findArticulationPoints(VerticeTarjan verticeTarjan,
int numeroVisita, Map<Integer, VerticeTarjan> infoVerticesVisitados) {
46
47          numeroVisita++;
48          int hijosDelVertice = 0;
49          boolean isArticulationPoint = false;
50
51          Iterator<Arista> aristas =
this.grafo.getAdyacentes(verticeTarjan.id);
52
53          while (aristas.hasNext()) {
54              int verticeAdyacenteId = aristas.next().getDst();
55
56              // Chequeo que la arista no vaya hacia el mismo vértice
57              if (verticeAdyacenteId == verticeTarjan.id) {
58                  continue;
59              }
60
61              // Chequeo que el nodo adyacente no haya sido visitado
62              VerticeTarjan adyacente =
infoVerticesVisitados.get(verticeAdyacenteId);
63              if (adyacente == null) {
64                  hijosDelVertice++;
65                  VerticeTarjan hijoAdyacente = new
VerticeTarjan(verticeAdyacenteId, verticeTarjan.id, numeroVisita,
numeroVisita);
66                  infoVerticesVisitados.put(verticeAdyacenteId,
hijoAdyacente);
67                  this.findArticulationPoints(hijoAdyacente,
numeroVisita, infoVerticesVisitados);
68
69                  if (verticeTarjan.numeroVisita <=
hijoAdyacente.numeroBajo) {
70                      isArticulationPoint = true;
71                  } else {
72                      verticeTarjan.numeroBajo =
Math.min(verticeTarjan.numeroBajo, hijoAdyacente.numeroBajo);
73                  }
74              } else {
75                  verticeTarjan.numeroBajo =
Math.min(verticeTarjan.numeroBajo, adyacente.numeroVisita);
76              }
```

```
77          }
78
79          // Es un punto de articulación si:
80          // - Luego del recorrido DFS: numeroVisita <= numeroBajo
(hallado en el loop)
81          // - Es raiz y tiene más de 2 hijos
82          if(verticeTarjan.padreId == -1 && hijosDelVertice >= 2
83              || verticeTarjan.padreId != -1 && isArticulationPoint) {
84              this.articulationPoints.add(verticeTarjan.id);
85          }
86      }
87  }
88
89
```

File: ./src/com/fiuba/tarjan/VerticeTarjan.java

```
1   package com.fiuba.tarjan;
2
3   /**
4    * Created by marianovazquez on 4/20/17.
5    */
6   /**
7    * Created by marianovazquez on 4/20/17.
8    */
9   public class VerticeTarjan {
10      public int id;
11      public int padreId;
12      public int numeroVisita;
13      public int numeroBajo;
14
15      public VerticeTarjan(int id, int padreId, int numeroVisita, int
numeroBajo) {
16          this.id = id;
17          this.padreId = padreId;
18          this.numeroVisita = numeroVisita;
19          this.numeroBajo = numeroBajo;
20      }
21  }
```

File: ./test/com/fiuba/galeShapley/GaleShapleyTest.java

```
1   package com.fiuba.galeShapley;
2
3   import org.junit.Test;
4
5   import java.util.ArrayList;
6   import java.util.Arrays;
7   import java.util.HashMap;
8
9   import static org.junit.Assert.assertEquals;
10
11  /**
12   * Created by marianovazquez on 4/22/17.
13   */
```

```
14  public class GaleShapleyTest {
15
16      @Test
17      public void getParejas_basic_1() {
18          HashMap<Integer, ArrayList<Integer>>
listaDePreferenciasPreferentes = new HashMap<>();
19          listaDePreferenciasPreferentes.put(0, new
ArrayList<Integer>(Arrays.asList(0, 1, 2)));
20          listaDePreferenciasPreferentes.put(1, new
ArrayList<Integer>(Arrays.asList(0, 1, 2)));
21          listaDePreferenciasPreferentes.put(2, new
ArrayList<Integer>(Arrays.asList(0, 1, 2)));
22
23          HashMap<Integer, ArrayList<Integer>>
listaDePreferenciasOferentes = new HashMap<>();
24          listaDePreferenciasOferentes.put(0, new
ArrayList<Integer>(Arrays.asList(0, 1, 2)));
25          listaDePreferenciasOferentes.put(1, new
ArrayList<Integer>(Arrays.asList(0, 1, 2)));
26          listaDePreferenciasOferentes.put(2, new
ArrayList<Integer>(Arrays.asList(0, 1, 2)));
27
28          GaleShapley galeShapley = new
GaleShapley(listaDePreferenciasPreferentes,
listaDePreferenciasOferentes);
29          HashMap<Integer, Integer> resultado =
galeShapley.getParejas();
30
31          HashMap<Integer, Integer> esperado = new HashMap<>();
32          esperado.put(0, 0);
33          esperado.put(1, 1);
34          esperado.put(2, 2);
35
36          assertEquals(esperado, resultado);
37      }
38
39      @Test
40      public void getParejas_basic_2() {
41          HashMap<Integer, ArrayList<Integer>>
listaDePreferenciasPreferentes = new HashMap<>();
42          listaDePreferenciasPreferentes.put(0, new
ArrayList<Integer>(Arrays.asList(0, 1, 2)));
43          listaDePreferenciasPreferentes.put(1, new
ArrayList<Integer>(Arrays.asList(0, 1, 2)));
44          listaDePreferenciasPreferentes.put(2, new
ArrayList<Integer>(Arrays.asList(0, 1, 2)));
45
46          HashMap<Integer, ArrayList<Integer>>
listaPreferenciasOferentes = new HashMap<>();
47          listaPreferenciasOferentes.put(0, new
ArrayList<Integer>(Arrays.asList(2, 1, 0)));
48          listaPreferenciasOferentes.put(1, new
ArrayList<Integer>(Arrays.asList(2, 1, 0)));
49          listaPreferenciasOferentes.put(2, new
ArrayList<Integer>(Arrays.asList(2, 1, 0)));
50
51          GaleShapley galeShapley = new
GaleShapley(listaDePreferenciasPreferentes, listaPreferenciasOferentes);
52          HashMap<Integer, Integer> resultado =
galeShapley.getParejas();
53
```

```
54          HashMap<Integer, Integer> esperado = new HashMap<>();
55          esperado.put(0, 2);
56          esperado.put(1, 1);
57          esperado.put(2, 0);
58
59          assertEquals(esperado, resultado);
60      }
61
62      @Test
63      public void getParejas_basic_3() {
64          HashMap<Integer, ArrayList<Integer>>
listaDePreferenciasPreferentes = new HashMap<>();
65          listaDePreferenciasPreferentes.put(0, new
ArrayList<Integer>(Arrays.asList(0, 1, 2)));
66          listaDePreferenciasPreferentes.put(1, new
ArrayList<Integer>(Arrays.asList(0, 1, 2)));
67          listaDePreferenciasPreferentes.put(2, new
ArrayList<Integer>(Arrays.asList(0, 1, 2)));
68
69          HashMap<Integer, ArrayList<Integer>>
listaPreferenciasOferentes = new HashMap<>();
70          listaPreferenciasOferentes.put(0, new
ArrayList<Integer>(Arrays.asList(2, 0, 1)));
71          listaPreferenciasOferentes.put(1, new
ArrayList<Integer>(Arrays.asList(2, 0, 1)));
72          listaPreferenciasOferentes.put(2, new
ArrayList<Integer>(Arrays.asList(2, 0, 1)));
73
74          GaleShapley galeShapley = new
GaleShapley(listaDePreferenciasPreferentes, listaPreferenciasOferentes);
75          HashMap<Integer, Integer> resultado =
galeShapley.getParejas();
76
77          HashMap<Integer, Integer> esperado = new HashMap<>();
78          esperado.put(0, 1);
79          esperado.put(1, 2);
80          esperado.put(2, 0);
81
82          assertEquals(esperado, resultado);
83      }
84  }
```

File: ./test/com/fiuba/galeShapley/HospitalRandomizerTest.java

```
1   package com.fiuba.galeShapley;
2
3   import org.junit.Test;
4
5   import static org.junit.Assert.*;
6
7   /**
8    * Created by lt5420 on 22/04/2017.
9    */
10  public class HospitalRandomizerTest {
11      @Test
12      public void toFiles() throws Exception {
13          HospitalRandomizer hosp = new
HospitalRandomizer(10000,2000);
```

```
14
15          hosp.toFiles("fileprueba.txt");
16
17          assertTrue(true);
18      }
19
20  }
```

File: ./test/com/fiuba/grafos/DigrafoTest.java

```
1   package com.fiuba.grafos;
2
3   import com.fiuba.kosaraju.RecorredorDeGrafo;
4   import org.junit.Test;
5
6   import java.util.ArrayList;
7   import java.util.Iterator;
8
9   import static org.junit.Assert.*;
10
11   /**
12    * Created by gatti2602 on 09/04/17.
13    */
14   public class DigrafoTest {
15       @Test
16       public void cuentaDeVertices() throws Exception {
17           Digrafo d = new Digrafo(5);
18
19           assertEquals(Integer.valueOf(5), d.cuentaDeVertices());
20       }
21
22       @Test
23       public void cuentaDeAristas() throws Exception {
24           Digrafo d = new Digrafo(5);
25           d.agregarArista(1, 4);
26           d.agregarArista(1, 2);
27           d.agregarArista(0, 3);
28           assertEquals(Integer.valueOf(3), d.cuentaDeAristas());
29
30           //Agrego arista invalida
31           d.agregarArista(5, 2);
32           assertEquals(Integer.valueOf(3), d.cuentaDeAristas());
33
34       }
35
36       @Test
37       public void agregarAristasDuplicadasNoDuplica() throws Exception
{
38           Digrafo d = new Digrafo(5);
39           d.agregarArista(1, 4);
40           assertEquals(Integer.valueOf(1), d.cuentaDeAristas());
41           d.agregarArista(1, 4);
42           assertEquals(Integer.valueOf(1), d.cuentaDeAristas());
43       }
44
45       @Test
46       public void getAdjList() throws Exception {
47           Digrafo d = new Digrafo(5);
```

```
48          d.agregarArista(1, 4);
49          Iterator<Arista> it = d.getAdjList(1);
50
51          assertTrue(it.hasNext());
52
53          Arista a = it.next();
54          assertFalse(it.hasNext());
55          assertEquals(Integer.valueOf(1), a.getSrc());
56          assertEquals(Integer.valueOf(4), a.getDst());
57      }
58
59      @Test
60      public void agregarArista() throws Exception {
61          Digrafo d = new Digrafo(5);
62          d.agregarArista(1, 4);
63          Arista a = d.getAdjList(1).next();
64
65          assertEquals(Integer.valueOf(1), a.getSrc());
66          assertEquals(Integer.valueOf(4), a.getDst());
67      }
68
69      @Test
70      public void transponerGrafo() throws Exception {
71          Digrafo d = new Digrafo(2);
72          d.agregarArista(0, 1);
73          d.agregarArista(1, 0);
74          Digrafo t = d.transponer();
75
76          assertEquals(Integer.valueOf(2), t.cuentaDeAristas());
77          assertEquals(d.cuentaDeVertices(), t.cuentaDeVertices());
78          assertTrue(t.existeArista(1, 1));
79          assertTrue(t.existeArista(0, 0));
80          assertNotSame(d, t);
81      }
82
83      @Test
84      public void DFSDevuelveListaCorrecta() throws Exception {
85          Digrafo d = new Digrafo(4);
86          d.agregarArista(0, 1);
87          d.agregarArista(1, 2);
88
89          //Componentes conexas 0->1->2 y 3
90          ArrayList<ArrayList<Integer>> componentesConexas = new
RecorredorDeGrafo(d).DFS(null, null);
91
92          assertEquals(Integer.valueOf(2), (Integer)
componentesConexas.size());
93          //Chequeo Componente 1
94          assertEquals(Integer.valueOf(3), (Integer)
componentesConexas.get(0).size());
95          assertEquals(Integer.valueOf(0), (Integer)
componentesConexas.get(0).get(0));
96          assertEquals(Integer.valueOf(1), (Integer)
componentesConexas.get(0).get(1));
97          assertEquals(Integer.valueOf(2), (Integer)
componentesConexas.get(0).get(2));
98          //Chequeo Componente 2
99          assertEquals(Integer.valueOf(1), (Integer)
componentesConexas.get(1).size());
100          assertEquals(Integer.valueOf(3), (Integer)
componentesConexas.get(1).get(0));
```

```
101
102      }
103
104      @Test
105      public void DFSDevuelveListaCorrectaConOrden() throws Exception
{
106          Digrafo d = new Digrafo(4);
107          d.agregarArista(0, 1);
108          d.agregarArista(1, 2);
109          ArrayList<Integer> orden = new ArrayList<>();
110          orden.add(3);
111          orden.add(0);
112          orden.add(1);
113          orden.add(2);
114          //Componentes conexas 0->1->2 y 3
115          ArrayList<ArrayList<Integer>> componentesConexas = new
RecorredorDeGrafo(d).DFS(null, orden);
116
117          assertEquals(Integer.valueOf(2), (Integer)
componentesConexas.size());
118          //Chequeo Componente 1
119          assertEquals(Integer.valueOf(3), (Integer)
componentesConexas.get(1).size());
120          assertEquals(Integer.valueOf(0), (Integer)
componentesConexas.get(1).get(0));
121          assertEquals(Integer.valueOf(1), (Integer)
componentesConexas.get(1).get(1));
122          assertEquals(Integer.valueOf(2), (Integer)
componentesConexas.get(1).get(2));
123          //Chequeo Componente 2
124          assertEquals(Integer.valueOf(1), (Integer)
componentesConexas.get(0).size());
125          assertEquals(Integer.valueOf(3), (Integer)
componentesConexas.get(0).get(0));
126
127      }
128
129  }
```

File: ./test/com/fiuba/tarjan/TarjanTest.java

```
1  package com.fiuba.tarjan;
2
3  import com.fiuba.grafos.Arista;
4  import com.fiuba.grafos.Grafo;
5  import com.fiuba.tarjan.Tarjan;
6  import org.junit.Test;
7  import java.util.Set;
8  import java.util.HashSet;
9
10  import static org.junit.Assert.assertEquals;
11  import static org.junit.Assert.assertTrue;
12
13  /**
14   * Created by marianovazquez on 4/19/17.
15   */
16  public class TarjanTest {
17
```

```
18      @Test
19      public void getArticulationPoints_basic_1() {
20          /*
21              Grafo:
22                      0---1---2---3
23
24              Puntos de articulación: {1, 2}
25          */
26          Grafo grafo = new Grafo(4);
27          grafo.agregarArista(0, 1);
28          grafo.agregarArista(1, 2);
29          grafo.agregarArista(2, 3);
30
31          Tarjan tarjan = new Tarjan(grafo);
32          Set<Integer> resultado = tarjan.getArticulationPoints();
33          Set<Integer> esperado = new HashSet<>(2);
34          esperado.add(1);
35          esperado.add(2);
36
37          assertEquals(esperado, resultado);
38      }
39
40      @Test
41      public void getArticulationPoints_basic_2() {
42          /*
43              Grafo:
44                      1-----0-----3
45                      |    /      |
46                      2---/       4
47
48              Puntos de articulación: {0, 3}
49          */
50          Grafo grafo = new Grafo(5);
51          grafo.agregarArista(0, 1);
52          grafo.agregarArista(0, 2);
53          grafo.agregarArista(0, 3);
54          grafo.agregarArista(1, 2);
55          grafo.agregarArista(3, 4);
56
57          Tarjan tarjan = new Tarjan(grafo);
58          Set<Integer> resultado = tarjan.getArticulationPoints();
59          Set<Integer> esperado = new HashSet<>(2);
60          esperado.add(0);
61          esperado.add(3);
62
63          assertEquals(esperado, resultado);
64      }
65
66      @Test
67      public void getArticulationPoints_basic_3() {
68          /*
69              Grafo:
70                      0-----1-----3-----5
71                      |    /|\         /
72                      2---/ 6 \---4---/
73
74
75              Puntos de articulación: {1}
76          */
77          Grafo grafo = new Grafo(7);
78          grafo.agregarArista(0, 2);
```

```
        grafo.agregarArista(0, 1);
        grafo.agregarArista(1, 2);
        grafo.agregarArista(1, 6);
        grafo.agregarArista(1, 4);
        grafo.agregarArista(1, 3);
        grafo.agregarArista(3, 5);
        grafo.agregarArista(4, 5);

        Tarjan tarjan = new Tarjan(grafo);
        Set<Integer> resultado = tarjan.getArticulationPoints();
        Set<Integer> esperado = new HashSet<>(1);
        esperado.add(1);

        assertEquals(esperado, resultado);
    }
}
```