TP - Unidad 06
Grafos

Ejercicio 1

Dada la siguiente interface para grafos:

```
package core_service;



import java.util.Collection;


// same interface for graph, digraph, multigraph, weighted graph, etc

// V participa de hashing. Redefinir equals y hashcode para que
detecte
// correctamente repetidos segun se desee


// E precisa la redefinicion de equals para que en remove lo encuentre
// y lo pueda borrar. Actualmente no participa de un hashing. Esta
encapsulado
// dentro de un objeto InternalEdge que lo contiene junto con el V
destino.
// Esa clase InternalEdge sí tiene hashcode y equals implementado.

public interface GraphService<V,E> {

      enum Multiplicity { SIMPLE, MULTIPLE};
      enum EdgeMode { UNDIRECTED, DIRECTED};
      enum SelfLoop { NO, YES};
      enum Weight{ NO, YES  };
      enum Storage { SPARSE, DENSE };

      // devuelve caracteristicas de la forma en que fue creado
      public String getType();


      // if exists lo ignora
      // else generate a new vertex
      public void addVertex(V aVertex);


      public int numberOfVertices();


      public Collection<V> getVertices();


      // parameters cannot be null
      // if any of the vertices is not present, the node is created
automatically
```

```
     // in the case of a weighted graph, E might implements the method
double getWeight()
     // otherwise an exception will be thrown

     // if the graph is not "multi" and there exists other edge with
same source and target,
     // throws exception
     public void addEdge(V aVertex, V otherVertex, E theEdge);


     public int numberOfEdges();



     // if the vertex does not exists returns false
     // else remove the vertex and its incident edges and returns true
     public boolean removeVertex(V aVertex);

     // removes the only edge that connects aVertex and otherVertex in
a simple/default graph
     // if the edge does not exists returns false
     // else returns removes it and returns true
     // the method is not allowed in multi graph/digraph=> throws
exception
     public boolean removeEdge(V aVertex, V otherVertex);



     // removes all the edges that connects aVertex and otherVertex in
multi graph/digraph with theEdge properties
     // removes the only edge that connects aVertex and otherVertex in
a simple graph/digraph,
     // without considering the edge properties
     // if the edge does not exists returns false
     // else returns returns true
     public boolean removeEdge(V aVertex, V otherVertex, E theEdge);


     // dump all the informations and structure of vertexes and edges
in any order
     public void dump();


     // if directed or vertex does not exists: throw exception
     // if self loop contributes twice
     public int degree(V vertex);

     // if undirected or vertex does not exists: throw exception
     // if self loop contributes twice
     public int inDegree(V vertex);

     // if undirected or vertex does not exists: throw exception
     // if self loop contributes twice
     public int outDegree(V vertex);
```

```
}
```

Y la version GraphFactory que permite que el usuario indique las características del grafo (denso/no denso) e instancia la clase correcta:

```
package core;

import core_service.GraphService;
import core_service.GraphService.*;

abstract public class GraphFactory<V, E> {

     public static <V, E> GraphService<V, E> create(Multiplicity
edgeMultiplicity, EdgeMode theEdgeMode,
                SelfLoop acceptSelfLoops, Weight hasWeight, Storage
theStorage) {

          if (theStorage== Storage.SPARSE)  // manejando 8 tipos con 2
clases concretas
                if (edgeMultiplicity== Multiplicity.SIMPLE)
                     return new
SimpleOrDefault<V,E>(theEdgeMode==EdgeMode.DIRECTED,

     acceptSelfLoops==SelfLoop.YES,

hasWeight==Weight.YES );
                else
                     return new
Multi<V,E>(theEdgeMode==EdgeMode.DIRECTED,

     acceptSelfLoops==SelfLoop.YES,

     hasWeight==Weight.YES );

          // todavia no lo hemos implementado en forma Densa Matriz
          //   return new AdjacencymatrixGraph<V,E>(isSimple,
isDirected, isWeighted );


          throw new RuntimeException("Not yet implemented");
     }

     private GraphFactory() {
     }
}
```

Y
GraphBuilder

```
package core_service;

import core.GraphFactory;
import core_service.GraphService.*;

public class GraphBuilder<V,E> {
      private Multiplicity multiplicity= Multiplicity.SIMPLE;
      private EdgeMode edgeMode= EdgeMode.DIRECTED;
      private SelfLoop acceptSelfLoops= SelfLoop.NO;
      private Weight hasWeight= Weight.NO;
      private Storage implementation= Storage.SPARSE;

      public GraphBuilder<V,E> withMultiplicity(Multiplicity param) {
            this.multiplicity= param;
            return this;
      }

      public GraphBuilder<V,E> withDirected(EdgeMode param) {
            this.edgeMode= param;
            return this;
      }

      public GraphBuilder<V,E> withAcceptSelfLoop(SelfLoop param) {
            this.acceptSelfLoops= param;
            return this;
      }

      public GraphBuilder<V,E> withAcceptWeight(Weight param) {
            this.hasWeight= param;
            return this;
      }

      public GraphBuilder<V,E> withStorage(Storage param) {
            this.implementation= param;
            return this;
      }

      public GraphService<V,E> build() {
        return GraphFactory.create(multiplicity, edgeMode,
acceptSelfLoops, hasWeight, implementation);
  }

}
```

Analizar el código de la clase AdjacencyListGraph y sus subclases Multi y SimpleOrDefault

1.1) Implementar el método **public int addEdge(V v1, V v2, E ege)**

1.2) Implementar **public int numberOfEdges()**

Caso de Uso A

```
g.addEdge('E', 'B', new EmptyEdgeProp());
g.addEdge('A', 'B', new EmptyEdgeProp());
g.addEdge('F', 'B', new EmptyEdgeProp());
g.addVertex('D');
g.addVertex('G');
g.addEdge('E', 'F', new EmptyEdgeProp());
g.addEdge('F', 'A', new EmptyEdgeProp());
g.addEdge('F', 'G', new EmptyEdgeProp());
g.addEdge('U', 'G', new EmptyEdgeProp());
g.addEdge('T', 'U', new EmptyEdgeProp());
g.addEdge('C', 'G', new EmptyEdgeProp());
System.out.println( g.numberOfEdges() );  // 9
```

Caso de Uso B

```
g.addEdge('E', 'B', new EmptyEdgeProp());
g.addEdge('A', 'B', new EmptyEdgeProp());
g.addEdge('F', 'B', new EmptyEdgeProp());
g.addVertex('D');
g.addVertex('G');
g.addEdge('E', 'F', new EmptyEdgeProp());
g.addEdge('F', 'A', new EmptyEdgeProp());
g.addEdge('F', 'G', new EmptyEdgeProp());
g.addEdge('U', 'G', new EmptyEdgeProp());
g.addEdge('T', 'U', new EmptyEdgeProp());
g.addEdge('C', 'G', new EmptyEdgeProp());
g.addEdge('F', 'F', new EmptyEdgeProp());
System.out.println( g.numberOfEdges() );  // 10
```

Caso de Uso C

```
g.addEdge('E', 'B', new WeightedEdge(3));
g.addEdge('A', 'B', new WeightedEdge(1));
g.addEdge('F', 'B', new WeightedEdge(2));
g.addVertex('D');
g.addVertex('G');
g.addEdge('E', 'F', new WeightedEdge(-2));
g.addEdge('F', 'A', new WeightedEdge(8));
g.addEdge('F', 'G', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('T', 'U', new WeightedEdge(8));
g.addEdge('C', 'G', new WeightedEdge(1));
```

```
g.addEdge('G', 'U', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));
System.out.println( g.numberOfEdges() );  // 12
```

Caso de Uso D

```
g.addVertex('D');
g.addVertex('G');
g.addEdge('G', 'F', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('U', 'G', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));
System.out.println( g.numberOfEdges() );  // 5
```

1.3) Implementar **public int degree(V), public int inDegree(V), public intOutDegree(V). Cuando no aplica lanza excepcion.**

Caso de Uso A

```
g.addVertex('D');
g.addVertex('G');
g.addEdge('G', 'F', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('U', 'G', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));
System.out.println( g.degree('G') );
//  excepcion:  degree aplica para no dirigido
```

Caso de Uso B

```
g.addVertex('D');
g.addVertex('G');
g.addEdge('G', 'F', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('U', 'G', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));
System.out.println( g.inDegree('G') );     // 2
System.out.println( g.outDegree('G') );  // 1
System.out.println( g.inDegree('F') );     // 3
System.out.println( g.outDegree('F') );  //  2
```

Caso de Uso C

```
g.addEdge('E', 'B', new WeightedEdge(3));
g.addEdge('A', 'B', new WeightedEdge(1));
g.addEdge('F', 'B', new WeightedEdge(2));
g.addVertex('D');
g.addVertex('G');
```

```
g.addEdge('E', 'F', new WeightedEdge(-2));
g.addEdge('F', 'A', new WeightedEdge(8));
g.addEdge('F', 'G', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('T', 'U', new WeightedEdge(8));
g.addEdge('C', 'G', new WeightedEdge(1));
g.addEdge('G', 'U', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));
System.out.println( g.degree('G') );     // 4
System.out.println( g.degree('F') );  // 8
```

1.4) Implementar **public boolean removeVertex(V), evitando recorrer innecesariamente**

Caso de Uso A

```
g.addVertex('D');
g.addVertex('G');
g.addEdge('G', 'F', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('U', 'G', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));
g.removeVertex('G' );
```

Caso de Uso B

```
g.addVertex('D');
g.addVertex('U');
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));
g.removeVertex('D' );
```

Caso de Uso C

```
g.addVertex('D');
g.addVertex('G');
g.addEdge('G', 'F', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('U', 'G', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));
g.removeVertex('G' );
```

1.5) Implementar

public boolean removeEdge(V aVertex, V otherVertex, E theEdge)

public boolean removeEdge(V aVertex, V otherVertex)

Ejercicio 2

Agregar a la interface los siguientes métodos e implementarlos en las clases grafos.

2.1) void printBFS(V vertex);
Que lista el recorrido del grafo en BFS a partir del vértices solicitado. Si el mismo no existe, lanza excepcion.

2.2) void printDFS(V vertex);
Que lista el recorrido del grafo en DFS a partir del vértices solicitado. Si el mismo no existe, lanza excepcion.

2.3) Iterable<V> getBFS(Vertext vertex);
Que devuelve un iterador sobre los nodos en recorrido BFS a partir de vertex

2.4) Iterable<V> getDFS(Vertext vertex);
Que es la versión iterativa (no recursiva) del 2.2)

2.5) Iterable<V> getDFS(Vertext vertex);
Que devuelve un iterador sobre los nodos en recorrido DFS a partir de vertex

Ejercicio 3

Solo para grafos Simples y sin lazos (sino lanzar excepción) agregar el método
printAllPaths(Vertex start, Vertex end);

El mismo imprime TODAS los caminos posibles desde start hasta end.

Ejercicio 4

Implementar el método

```
public DijkstraPath<V,E> dijsktra(V source);
```

el cual devuelve no solo el peso del camino mas corto entre nodo source y todos los demás. Tambien se quiere saber cual fue dicha ruta, por lo cual usar la siguiente clase auxiliar:

```
package core;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
```

```
import java.util.Map;

public class DijkstraPath<V,E> {

      private Map<V,Integer> distancesFromSource;
      private Map<V,V> prevVertex;

      public DijkstraPath(V source, Map<V,Integer> distancesFromSource
, Map<V,V> prevVertex) {
      //    this.source= source;
            this.distancesFromSource= distancesFromSource;
            this.prevVertex= prevVertex;

      }

      @Override
      public String toString() {
            String rta= "";
            for(V aV: distancesFromSource.keySet()) {
                  if ( distancesFromSource.get(aV) == Integer.MAX_VALUE
)
                        rta+= "INF: []\n";
                  else
                        rta+= distancesFromSource.get(aV)+ ": " +
getShortestPathTo(aV) + "\n";
            }

            return rta;
      }



      public String getShortestPathTo(V targetVertex){
         List<V> path = new ArrayList<>();

         for(V
vertex=targetVertex;vertex!=null;vertex=prevVertex.get(vertex)){
             path.add(vertex);
         }

         Collections.reverse(path);
         return path.toString();
     }



}
```

Ejercicio 5

Agregar a la interface el método

boolean isBipartite()

que detecte si el grafo es o no bipartito. Implementarlo.

Ejercicio 6

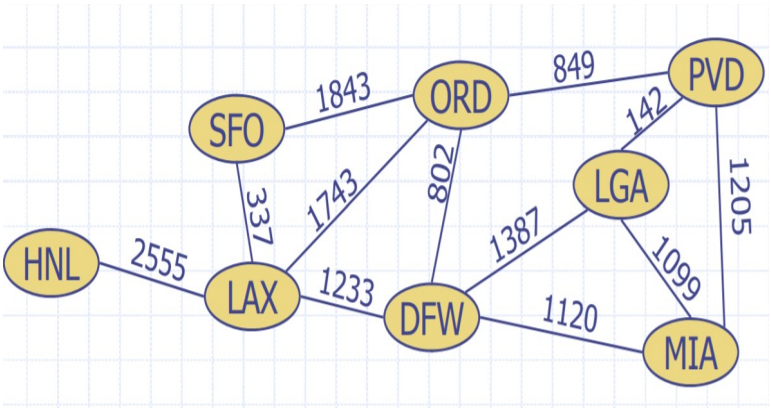
Agregar a la interface el método

boolean hasCycle()

que detecte si el grafo tiene ciclos. Implementarlo

Ejercicio 7

Se quieren representar las siguientes rutas aéreas



7.1) Usar el grafo anterior para calcular el camino mínimo con fuente "PVD" y todos los demás. Se quiere conocer: cuál es ese camino (si existe) y su peso.
En el ejemplo deberíamos obtener:

Shortest path from PVD to HNL:
[(ORD : PVD), (LAX : ORD), (HNL : LAX)]
5147.0
Shortest path from PVD to SFO:
[(ORD : PVD), (SFO : ORD)]
2692.0

Shortest path from PVD to LAX:
[(ORD : PVD), (LAX : ORD)]
2592.0
Shortest path from PVD to ORD:
[(ORD : PVD)]
849.0
Shortest path from PVD to DFW:
[(LGA : PVD), (DFW : LGA)]
1529.0
Shortest path from PVD to LGA:
[(LGA : PVD)]
142.0
Shortest path from PVD to PVD:
[]
0.0
Shortest path from PVD to MIA:
[(MIA : PVD)]
1205.0

7.2) Explicar qué hubiera sucedido si en vez de
flight.setEdgeWeight(flight.addEdge("LGA", "PVD"), 142);

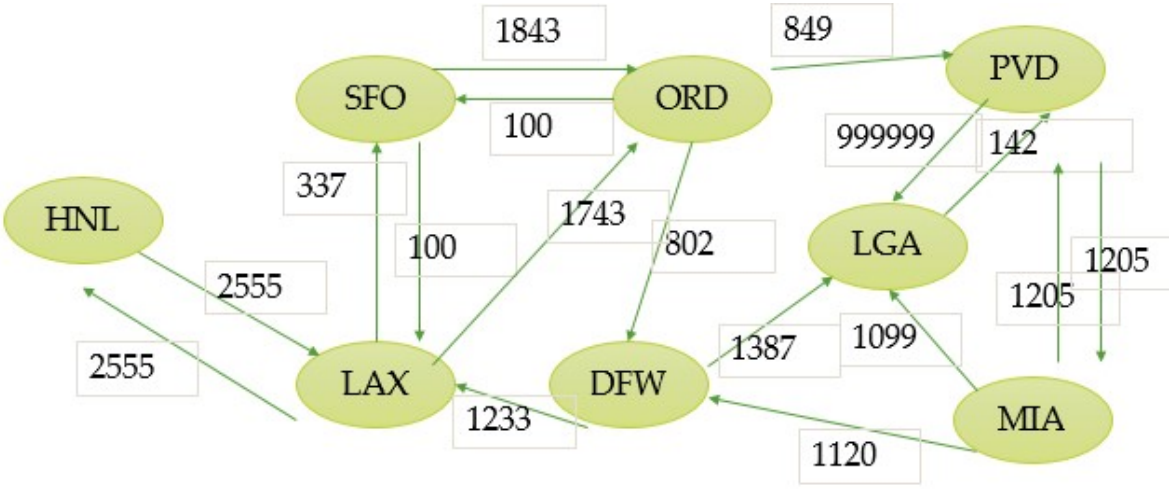
definimos

flight.setEdgeWeight(flight.addEdge("PVD", "LGA"), 142);

¿cambia el resultado?

Crear un grafo cuyos nodos son de tipo Person y Music (colocar propiedades acorde). Los ejes deben representar que persona escucho alguna vez que música.

7.3) Repetir el ejercicio de rutas aéreas pero cambiar el grafo simple por un dirigido. El digrafo esperado es:



Calcular nuevamente el camino más corto desde PVD a los otros nodos.

En el ejemplo deberíamos obtener:
DefaultGraphType [directed=true, undirected=false, self-loops=false, multiple-edges=false, weighted=true, allows-cycles=true, modifiable=true]
([HNL, SFO, LAX, ORD, DFW, LGA, PVD, MIA], [(HNL,LAX), (LAX,HNL), (LAX,SFO), (SFO,LAX), (LAX,ORD), (DFW,LAX), (SFO,ORD), (ORD,SFO), (ORD,DFW), (DFW,LGA), (MIA,DFW), (ORD,PVD), (LGA,PVD), (PVD,LGA), (MIA,LGA), (MIA,PVD), (PVD,MIA)])
Shortest path from PVD to HNL
[(PVD : MIA), (MIA : DFW), (DFW : LAX), (LAX : HNL)]
6113.0
Shortest path from PVD to SFO
[(PVD : MIA), (MIA : DFW), (DFW : LAX), (LAX : SFO)]
3895.0
Shortest path from PVD to LAX
[(PVD : MIA), (MIA : DFW), (DFW : LAX)]
3558.0
Shortest path from PVD to ORD
[(PVD : MIA), (MIA : DFW), (DFW : LAX), (LAX : ORD)]
5301.0
Shortest path from PVD to DFW
[(PVD : MIA), (MIA : DFW)]
2325.0
Shortest path from PVD to LGA
[(PVD : MIA), (MIA : LGA)]
2304.0
Shortest path from PVD to PVD
[]
0.0
Shortest path from PVD to MIA
[(PVD : MIA)]
1205.0