Actividad 6. Implementación y uso del TAD Map

Objetivo

Conocer el TAD MAP como estructura adecuada para almacenar una colección de objetos clave/valor, implementarlo y saber utilizar un Map para resolver problemas.

Procedimiento

Para resolver esta actividad se seguirá la *técnica del rompecabezas*.

1. Formar equipos de 4 personas, donde cada miembro se encargará de la resolución de un ejercicio, convirtiéndose en experto. El profesor tomará nota de la composición de los equipos en horas de grupo reducido (23, 24 o 25 de octubre) y el ejercicio a realizar por cada uno.
2. Todos los alumnos deben formarse en el diseño del TAD Map (revisar las transparencias que sobre el TAD Map están disponible en faitic, Documentos e Ligazóns/Teoría/Map.rar, que incluye lecturas recomendadas). Trabajo individual.
3. Tras la preparación inicial e individual se reunirán los alumnos del equipo para la puesta en común y unificación de conceptos (resolviendo dudas si las tienen). Trabajo grupal y presencial en grupo reducido. Como resultado de esta reunión el encargado del TAD Map subirá a faitic la especificación acordada, nombrando el fichero como **"Especificacion_Map_nombre de los cuatro alumnos.rar"**. Una vez que el profesor da el visto bueno, dicho alumno comenzará con la implementación del TAD Map (Ver ejercicio 1).
4. A partir del 30 de octubre, los otros tres miembros del equipo deben encargarse cada uno de resolver el ejercicio asignado de la actividad (ejercicios 2.a, 2.b y 2.c). Para ello deben revisar las transparencias sobre grafos y esquemas algorítmicos que están disponibles en faitic, Documentos e Ligazóns/Teoría/Grafos.pdf y Documentos e Ligazóns/Teoría/EsquemasAlgoritmicos.pdf. Trabajo individual.
5. Con la propuesta de resolución de los algoritmos, los expertos de cada equipo se reunirán para la puesta en común. Posteriormente cada experto explicará a los miembros de su grupo su algoritmo. Trabajo grupal y presencial en grupo reducido. Una vez implementado el algoritmo se subirá a faitic; cada uno sube el algoritmo en que es experto. Nombre de los ficheros **"Grafo_nombre de los cuatro alumnos.rar", "Abreviaturas_nombre de los cuatro alumnos.rar"** y **"Colorear_nombre de los cuatro alumnos.rar";** además el otro miembro del grupo debe subir la implementación del TAD Map con el siguiente nombre **"TAD_Map_nombre de los cuatro alumnos.rar"**. Fecha tope para subir estos documentos 16 de noviembre.

Evaluación

Evaluación **grupal** mediante tutoría acordada con el profesor una vez terminada la actividad, antes del 26 de noviembre.

Mediante una prueba **individual** el día 26 de noviembre, esta calificación representa el 20% de la nota final.

Tiempo estimado

7 horas

Ejercicios

1.- Implementación del TAD Map<K,V>. Para ello desarrolla un proyecto en java que contenga la interfaz Map y una clase que implemente dicha interfaz haciendo uso de una Tabla Hash.

La **tabla hash** se utiliza para almacenar la colección de objetos <clave, valor> y debe utilizar la estrategia de *encadenamiento abierto o separado* (un array de listas) para resolver el problema de las colisiones. Para trabajar con listas se puede usar el TAD Lista de la librería *aed1.jar* o la interface *List* de java.

Durante el proceso de búsqueda, inserción o borrado de un elemento de la tabla hash, el primer paso a realizar es transformar la clave en un índice del array, es decir, aplicar una función hash. Este índice indicará la lista en la que hay que buscar, insertar o eliminar el elemento correspondiente. Esta función debe implementarse mediante un método privado que, para esta actividad, codificará el *método de división*:

private int funcionHash (K clave)

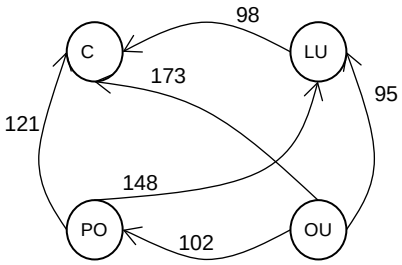
Un problema de esta función (método de división) es que para poder aplicarla es necesario que el objeto que se pasa como parámetro sea de tipo int. Para resolver este problema existe en java el método **hashCode()**, que convierte cualquier objeto en un int. Es decir, **clave.hashCode()** convierte el parámetro de entrada de la función a un valor entero, lo cual permite trabajar con cualquier objeto en la tabla hash.

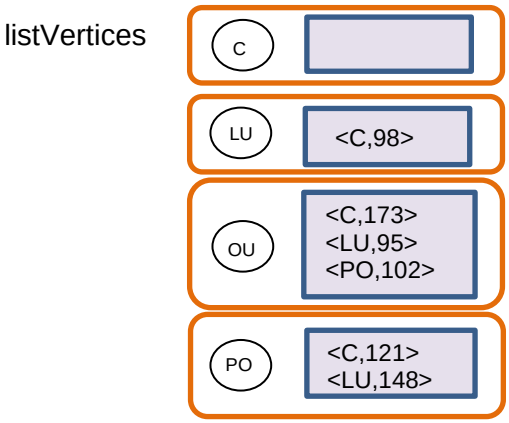
2.- Haciendo uso del TAD Map, implementa los algoritmos que se indican a continuación:

a) Implementa el TAD Grafo haciendo uso de un MAP.

Una implementación eficiente del TAD Grafo<E,T> es posible mediante **mapas de adyacencia**. Es una implementación similar a la implementación mediante listas de adyacencia, es decir, se mantiene una lista de vértices, y para cada vértice se guarda la relación de vértices adyacentes. Pero en esta ocasión, en lugar de guardar los adyacentes en una lista, se guardan en un TAD Map<K,V>, donde el vértice adyacente actúa como clave y la etiqueta del arco actúa como valor.

Ejemplo:





Una posible clase que implemente la interfaz Grafo<E,T> haciendo uso de dicha representación sería:

```
public class MapAdyacencia<E,T> implements Grafo<E,T>{
private List<VerticeConMap<E,T>> listVertices;
```

Donde la clase VerticeConMap<E,T> es una clase con dos atributos, como se puede observar a continuación:

```
public class VerticeConMap<E,T>  {
   private Vertice<E> origen;                  // vértice del grafo
   private Map<Vertice<E>, T> mapAdy;          // mapa de adyacentes del vértice origen

   VerticeConMap(Vertice<E> v) {
      origen = v;
      mapAdy = new HashMap<>();
   }
   Vertice<E> getVertice(){
      return origen;
   }
   Map<Vertice<E>, T> getAdy(){
        return mapAdy;
   }
}
```

Es importante aclarar que el mapa guarda pares <clave, valor>, siendo en este caso la clave el vértice adyacente, y el valor, la etiqueta del arco.

Se pide, haciendo uso de dicha representación, implementar los métodos de la interfaz Grafo<E,T> que se indican a continuación:

public Iterator<Vertice<E>> adyacentes (Vertice<E> v)

//Produce: devuelve un iterador sobre los vértices adyacentes al vértice *v* que se pasa como parámetro.

public boolean estaArco(Arco<E,T> a)

//Produce: devuelve cierto si el arco *a* que se pasa como parámetro pertenece al conjunto de arcos del grafo

La clase Arco<E,T> se muestra a continuación:

```
public class Arco <E,T>{
  private Vertice<E> origen, destino;
  private T etiqueta;
  public Arco(Vertice<E> o, Vertice<E> d, T c)
  public Vertice<E>E getOrigen()
  public Vertice<E> getDestino()
  public T getEtiqueta()
}
```

Una vez realizada la implementación de los métodos, prueba su correcto funcionamiento haciendo uso del framework JUnit y de la clase de prueba **MapAdyacencia*Test.java*** que se proporciona en faitic, *Documentos e Ligazons/Actividades/Test (JUnit).*

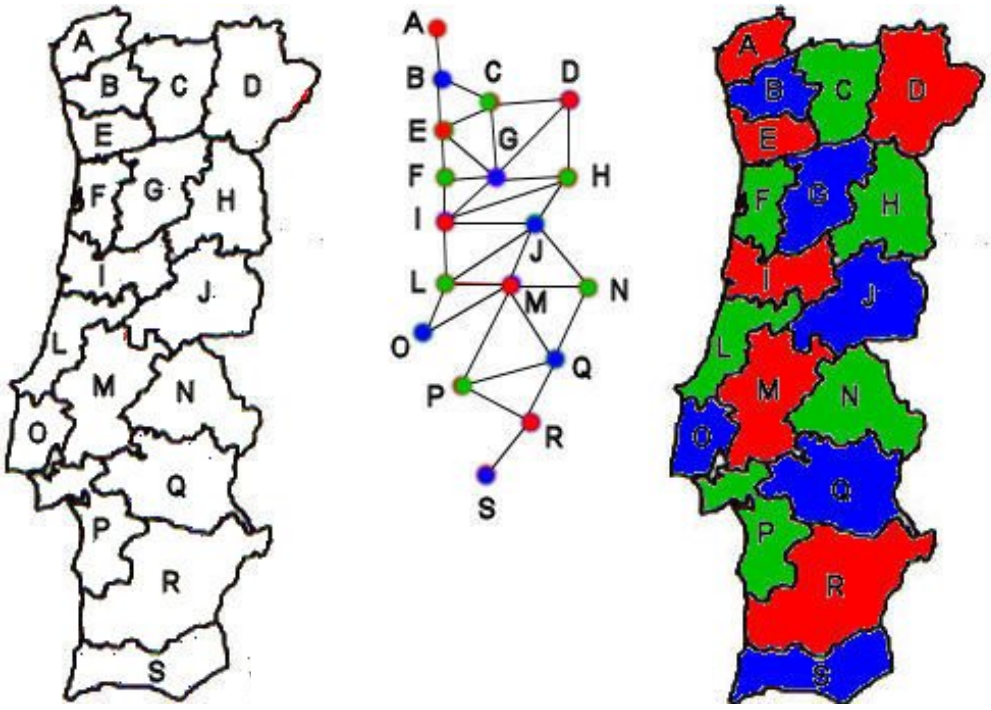
b) Usando el TAD Map desarrolla un proyecto que dado un texto con abreviaturas, devuelva un nuevo texto con esas abreviaturas extendidas. Para ello descarga de faitic *Documentos e Ligazons/Actividades/* **ExtenderAbreviaturas.rar**
Este fichero contiene:

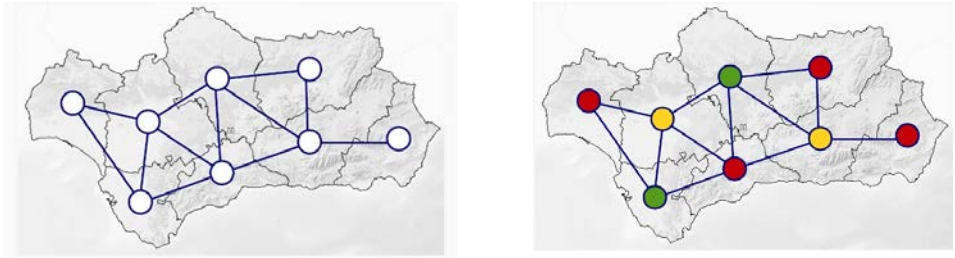
- tres ficheros **.json** que son, respectivamente, tres diccionarios de abreviaturas junto con su extensión, en los idiomas inglés, español y francés. Con ellos se puede extender las abreviaturas presentes en textos escritos en los tres idiomas.
- un fichero **.java** que contiene el código necesario para guardar la información de los diccionarios de abreviaturas en una estructura de datos MAP. Esta estructura de datos es la que se usará para realizar la extensión en los textos.
- Dos ficheros **.jar**, que son dos librerías necesarias para poder trabajar con los ficheros json

Una vez terminado el proyecto prueba su correcto funcionamiento haciendo uso del framework JUnit y de la clase de prueba **ExtendAbreviaturas*Test.java*** que se proporciona en faitic, *Documentos e Ligazons/Actividades/Test (JUnit).*

c) El **algoritmo de número cromático** indica el número de colores mínimo necesario para colorear los vértices de un grafo. El *teorema de los cuatro colores* justifica que el número cromático de un grafo plano es menor o igual que cuatro.

El objetivo de este ejercicio consiste en, dado un grafo plano que representa un mapa político, indicar cómo deben colorearse cada uno de los vértices de dicho grafo, empleando el menor número posible de colores (según el teorema citado anteriormente como máximo cuatro). Emplea una estrategia **voraz** para resolver el ejercicio.





```
public static <E> Map<Vertice<E>,String> colorearMapa (Grafo<E,Integer>
g, String [] colores)
```

Una vez realizada la implementación de los algoritmos, prueba su correcto funcionamiento haciendo uso del framework JUnit y de la clase de prueba **ColorearMapa*Test.java*** que se proporciona en faitic, *Documentos e Ligazons/Actividades/Test (JUnit).*