UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

GRADO EN INGENIERÍA INFORMÁTICA

Técnicas de Programación Avanzada

Profesor:

Borja Monsalve Piqueras

Alumnos:

Paula Sáenz de Santa María Diez

Andrés Ramos García

Códigos

Función testCiclos

Esta es la función principal del código, en la cual se hace un bucle que recorre la lista de los vértices del grafo, y además metemos una condición de encontrado, para que si se encuentra un ciclo a mitad de ejecución este pueda salir sin problema.
Dentro del bucle se consulta primero el vértice actual, lo insertamos en una lista creada para tener constancia de los vértices visitados y se hace la llamada recursiva a la función auxiliar, en la que se pasa el grafo entero, el vértice actual y una lista vacía, todo esto se guarda en una variable booleana.
Por último retornamos el valor "encontrado".

```
public static  <Clave, InfoVertice, Coste> boolean testCiclos (Grafo<Clave,
InfoVertice, Coste> grafo){
      int i =1;
      boolean encontrado = false;
      Lista<Clave> visitados = new Lista<>();

      //pasamos por todos los vertices del grafo hasta que encuentre un ciclo
      while (i<=grafo.listaVertices().longitud() && encontrado == false){
         Clave v = grafo.listaVertices().consultar(i);
         visitados.insertar(visitados.longitud()+1, v);
         encontrado = testCiclos_aux(grafo, v, new Lista<>());
         i++;
      }
      return encontrado;
   }
```

Función testCiclos_aux

Esta función auxiliar se usa para recorrer todos los caminos posibles desde un vértice, se usa en otra función, ya que para su correcto funcionamiento, necesitamos pasar por parámetro la lista de vértices visitados (el camino actual que lleva recorrido) y el vértice actual, ya que este tiene que ir actualizando, además de pasar el grafo entero.

La parte principal del código es el bucle que maneja la recursividad de los caminos y el condicional al inicio de la función para ir comprobando si se ha encontrado o no, un ciclo.

El bucle consulta los sucesores en la posición "i" que va cambiando (para asegurarnos que va por todos los sucesores si fuese necesario), además se le asigna a la variable booleana la recursividad, a la cual se le pasa el grafo, el sucesor que se ha consultado previamente y la lista con los vértices visitados (el camino actual).

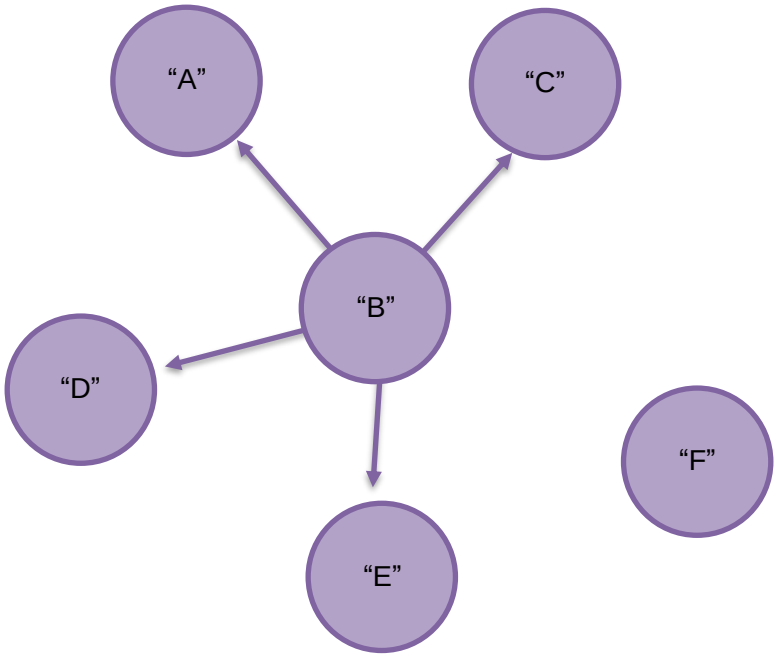
Para que esto funcione, si llegamos al final de un camino y no existe ciclo, tenemos que retornar al estado anterior, eliminando de la lista los vértices que hemos insertado para que pueda buscar otro camino. Esto es esencial ya que las listas se pasan por referencia y tenemos que asegurarnos que eliminamos los elementos.

```
private static <Clave, InfoVertice, Coste> boolean
testCiclos_aux(Grafo<Clave, InfoVertice, Coste> gr,
Clave inicio,Lista<Clave> camino){

        boolean encuentra = false;
        //encuentra ciclo en el camino actual
        if(camino.buscar(inicio)!=0){
            return true;
        } else{
            camino.insertar(camino.longitud()+1, inicio);
            int i = 1;
            Lista<Clave> sucesores = gr.listaSucesores(inicio);
            //pasa por todos los sucesores
            while (i <= sucesores.longitud() && !encuentra ){
                Clave v = sucesores.consultar(i);
                encuentra = testCiclos_aux(gr, v, camino);
                i++;
            }
            camino.borrar(camino.longitud());
            return encuentra;
        }
    }
```

Dibujos con código

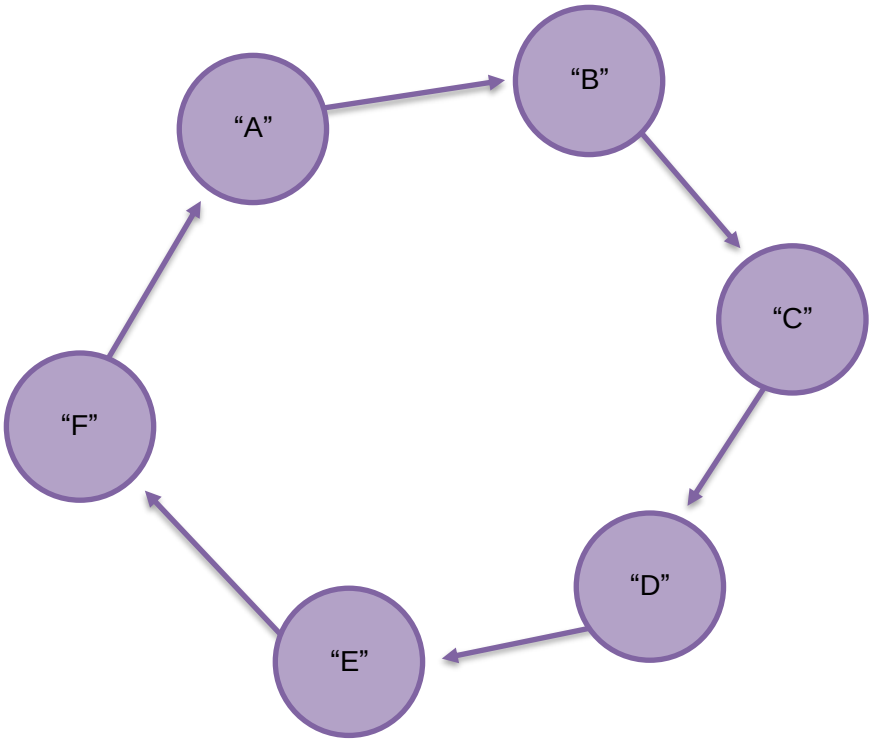
Grafo gPrueba




```
gPrueba.insertarVertice("A", "A");
gPrueba.insertarVertice("B", "B");
gPrueba.insertarVertice("C", "C");
gPrueba.insertarVertice("D", "D");
gPrueba.insertarVertice("E", "E");
gPrueba.insertarVertice("F", "F");


gPrueba.insertarArista("B", "A", 1);
gPrueba.insertarArista("B", "C", 1);
gPrueba.insertarArista("B", "D", 1);
gPrueba.insertarArista("B", "E", 1);
```

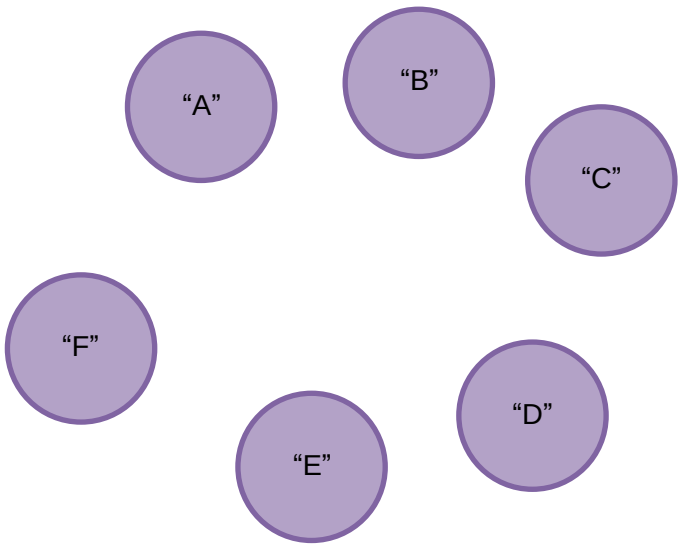
Grafo gPrueba1




```
gPrueba1.insertarVertice("A", "A");
gPrueba1.insertarVertice("B", "B");
gPrueba1.insertarVertice("C", "C");
gPrueba1.insertarVertice("D", "D");
gPrueba1.insertarVertice("E", "E");
gPrueba1.insertarVertice("F", "F");


gPrueba1.insertarArista("A", "B", 1);
gPrueba1.insertarArista("B", "C", 1);
gPrueba1.insertarArista("C", "D", 1);
gPrueba1.insertarArista("D", "E", 1);
gPrueba1.insertarArista("E", "F", 1);
gPrueba1.insertarArista("F", "A", 1);
```

Grafo gPrueba2




```
gPrueba1.insertarVertice("A", "A");
gPrueba1.insertarVertice("B", "B");
gPrueba1.insertarVertice("C", "C");
gPrueba1.insertarVertice("D", "D");
gPrueba1.insertarVertice("E", "E");
gPrueba1.insertarVertice("F", "F");
```