Tercer Entrega Grafos
Jueves 24/11- T

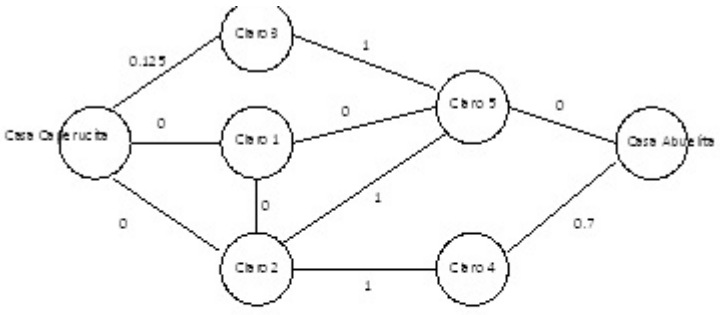
Un día, Caperucita Roja decide ir desde su casa hasta la de su abuelita, recolectando frutos del bosque del camino y tratando de hacer el paseo de la manera más segura posible. La casa de Caperucita está en un claro del extremo oeste del bosque, la casa de su abuelita en un claro del extremo este, y dentro del bosque entre ambas hay algunos otros claros con árboles frutales.
El bosque está representado por un grafo, donde los vértices representan los claros (identificados por un String) y las aristas los senderos que los unen. Cada arista contiene la probabilidad que el lobo aparezca en ese sendero.
Para no ser capturada por el lobo, Caperucita desea encontrar un camino que pase únicamente por los senderos con probabilidad 0. Como miembros de la sociedad Ayuda a Caperucita a Moverse (ACM), están llamados a ayudarla a encontrar ese camino, y cuando lo hayan hecho, colorín, colorado, este problema habrá terminado!

Su tarea es definir una clase llamada **BuscadorDeCamino**, con una variable de instancia llamada **"bosque"** de tipo **Grafo**, que representa el bosque descripto e implementar un método de instancia con la siguiente firma:

public ListaGenerica<String> recorridoMasSeguro()

que devuelva un listado con los claros que conforman el recorrido encontrado**.** Si existe más de uno, devuelva uno de ellos. Si no existe devuelva un listado vacío.

Nota: La casa de caperucita debe ser buscada antes de comenzar a buscar el recorrido, y tiene la identificación: "Casa Caperucita".



Caminos posibles:

1) Casa Caperucita - Claro 1 - Claro 5 - Casa Abuelita.
2) Casa Caperucita - Claro 2 – Claro 1 - Claro 5 - Casa Abuelita.

Entonces, el método podría devolver la opción 1 o la opción 2. (Es indiferente)

Posible solución al ejercicio planteado:

```
public class BuscadorDeCamino {
   private Grafo<String> bosque;

   public ListaGenerica<String> recorridoMasSeguro(){
     ListaGenerica<String> resultado = new ListaGenericaEnlazada<String>();
     ListaGenerica<String> listaActual = new ListaGenericaEnlazada<String>();
     ListaGenerica<Vertice<String>> listaVertices = bosque.listaDeVertices();
     boolean[] visitados = new boolean[listaVertices.tamanio() +1];
     listaVertices.comenzar();
     while (!listaVertices.fin()){
                Vertice<String> v = listaVertices.proximo();
                If (v.dato().equals ("Casa Caperucita"){
                     this.dfs(v, listaActual, resultado, visitados);
                     break;
                      }
           }
     return resultado;
     }


public void dfs (Vertice<String> v, ListaGenerica<String> lactual, ListaGenerica<String>
res, boolean[] visitados){
visitados[v.posicion()]=true;
lactual.agregarFinal(v.dato());
if(v.dato().equals("Casa Abuelita")){
          res.limpiar();
          res.agregarTodos(lactual);
          }
          else
          {
          ListaGenerica<Arista<String>> ady = bosque.listaDeAdyacentes(v);
          ady.comenzar();
          while (!ady.fin()){
              Arista<String> ar= ady.proximo();
              If ((ar.peso() == 0)&&(!visitados[ar.verticeDestino().posicion()])){
                    this.dfs(ar.verticeDestino(), listaActual, resultado, visitados);
                   }
              }
          }
         lactual.eliminar(v.dato());
    }


}
```