# Resolución Entrega Grafos
# Turno Miércoles Mañana

Redictado Algoritmos y Estructuras de Datos

Facultad de Informática, Universidad Nacional de La Plata, Argentina

30 de noviembre de 2016

# Enunciado

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 no pase por los senderos con probabilidad 1. 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!

# Enunciado

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 ya descripto. Si existe más de uno, devuelva uno de ellos. Si no existe devolverá un listado vacío.

**Nota: La casa de caperucita debe ser buscada antes de comenzar a buscar el recorrido**.

# Información que se dispone

- crear una clase **BuscadorDeCamino**
- **bosque** se representa con un **Grafo<String>**
- **bosque** es una variable de instancia de **BuscadorDeCamino**
- implementar
  **public ListaGenerica<String> recorridoMasSeguro()**

```
public class BuscadorDeCamino {

    private Grafo<String> bosque;

    public ListaGenerica <String> recorridoMasSeguro(){

    }
}
```

# Información que se dispone

Además:

- Los vértices de inicio y fin existen.
- Se debe buscar primero la casa de caperucita.
- Las aristas son pesadas, el peso determina una probabilidad.
- Puede que no haya un camino que cumpla.
- Devolver un recorrido que no pase por una arista con peso 1.

# Solución

```
public class BuscadorDeCamino {

    private Grafo<String> bosque;
    private ListaGenerica<String> camino;
    // camino se utiliza para guardar el camino a devolver

    public ListaGenerica<String> recorridoMasSeguro(){
        ListaGenerica<Vertice<String>> vertices = bosque.listaDeVertices();
        boolean seguir = true;
        ListaGenerica<String> caminoActual = new ListaEnlazadaGenerica<String>();
        //caminoActual es la lista donde se va guardando el camino actual
        camino = new ListaEnlazadaGenerica<String>();
        Vertice<String> vIni = null;
        vertices.comenzar();
        /* Se busca el vertice de inicio donde se dispara el recorrido.
         * Se sabe que la Casa de Caperucita existe */
        while(!vertices.fin() && seguir) {
            Vertice<String> v = vertices.proximo();
            if (v.dato().equals("Casa Caperucita"))
                vIni = v;
            if (vIni != null)
                seguir = false;
        }
        dfsRecorridoMasSeguro(vIni, caminoActual);
        return camino;

    }
```

# Solución

```
public class BuscadorDeCamino {

  ...
  private void dfsRecorridoMasSeguro(Vertice<String> vertice, ListaGenerica<String> caminoActual) {

      caminoActual.agregarFinal ( vertice .dato());
      if ( vertice .dato()=="Casa Abuelita") { // es lo mismo que (vertice.dato().equals("Casa Abuelita"))
          this .camino = caminoActual.clonar(); // Actualiza el camino
      }
      else {
          ListaGenerica<Arista<String>> adyacentes = this.bosque.listaDeAdyacentes(vertice);
          adyacentes.comenzar ();
          for (int j = 0; j < adyacentes.tamanio(); j++) { //Recorrer con un while es lo mismo
              Arista<String> arista = adyacentes.proximo ();
              if (!caminoActual.incluye ( arista . verticeDestino ().dato())) { // Si el vertice destino esta en
                    la lista es porque ya lo visite , es una alternativa para no usar un arreglo de
                    booleanos
                  if ( arista .peso() != 1){ // Si la arista tiene prob 1 no sigo recorriendo
                      dfsRecorridoMasSeguro( arista . verticeDestino (), caminoActual);

                  }
              }
          }

      }
      caminoActual.eliminarEn(caminoActual.tamanio());
  }
```

# Solución 2

Otra manera, sería chequear que si ya tengo un camino guardado, no lo reemplazo.

```
public class BuscadorDeCamino {

  ...
  private void dfsRecorridoMasSeguro(Vertice<String> vertice, ListaGenerica<String> caminoActual) {

      caminoActual.agregarFinal (vertice.dato());
      if (vertice.dato()=="Casa Abuelita") {
          if (this.camino.esVacia()) // Si ya tengo un camino no lo reemplazo
              this.camino = caminoActual.clonar(); // Guardo el camino
      }
      else {
          ListaGenerica<Arista<String>> adyacentes = this.bosque.listaDeAdyacentes(vertice);
          adyacentes.comenzar ();
          for (int j = 0; j < adyacentes.tamanio(); j++) {
              Arista<String> arista = adyacentes.proximo ();
              if (!caminoActual.incluye (arista.verticeDestino().dato())) {
                  if (arista.peso() != 1){
                      dfsRecorridoMasSeguro(arista.verticeDestino (), caminoActual);

                  }
              }
          }

      }
      caminoActual.eliminarEn(caminoActual.tamanio());
  }
```

# Chequear la solución

```
public static void main(String[] args) {
    Grafo<String> bosque = new GrafoImplListAdy<String>(); // Se crea el grafo
    Vertice<String> casaCaperucita=new VerticeImplListAdy<String>("Casa Caperucita");
    Vertice<String> claro3=new VerticeImplListAdy<String>("Claro 3");
    Vertice<String> claro1=new VerticeImplListAdy<String>("Claro 1");
    Vertice<String> claro2=new VerticeImplListAdy<String>("Claro 2");
    Vertice<String> claro5=new VerticeImplListAdy<String>("Claro 5");
    Vertice<String> claro4=new VerticeImplListAdy<String>("Claro 4");
    Vertice<String> casaAbuela=new VerticeImplListAdy<String>("Casa Abuelita");
    bosque.agregarVertice(casaCaperucita);
    bosque.agregarVertice(claro3);
    bosque.agregarVertice(claro1);
    bosque.agregarVertice(claro2);
    bosque.agregarVertice(claro5);
    bosque.agregarVertice(claro4);
    bosque.agregarVertice(casaAbuela);
    bosque.conectar(casaCaperucita, claro3, 0.125);
    bosque.conectar(casaCaperucita, claro1, 0.3);
    bosque.conectar(casaCaperucita, claro2, 0.45);
    bosque.conectar(claro3, claro5, 1);
    bosque.conectar(claro1, claro5, 0.5 );
    bosque.conectar(claro2, claro5, 1);
    bosque.conectar(claro2, claro1, 0.125);
    bosque.conectar(claro2, claro4, 1);
    bosque.conectar(claro5, casaAbuela, 0.2);
    bosque.conectar(claro4, casaAbuela, 0.7);
    //Se crea instancia de BuscadorDeCamino pasandole el grafo creado
    BuscadorDeCamino buscador = new BuscadorDeCamino(bosque);
    ListaGenerica<String> recorrido = buscador.recorridoMasSeguro();
    recorrido.comenzar();
    while(!recorrido.fin()){ //Se imprime el recorrido
        System.out.print(recorrido.proximo() + " - ");
    }
```