Algoritmos
y
Estructuras de Datos

Cursada 2015

Grafos

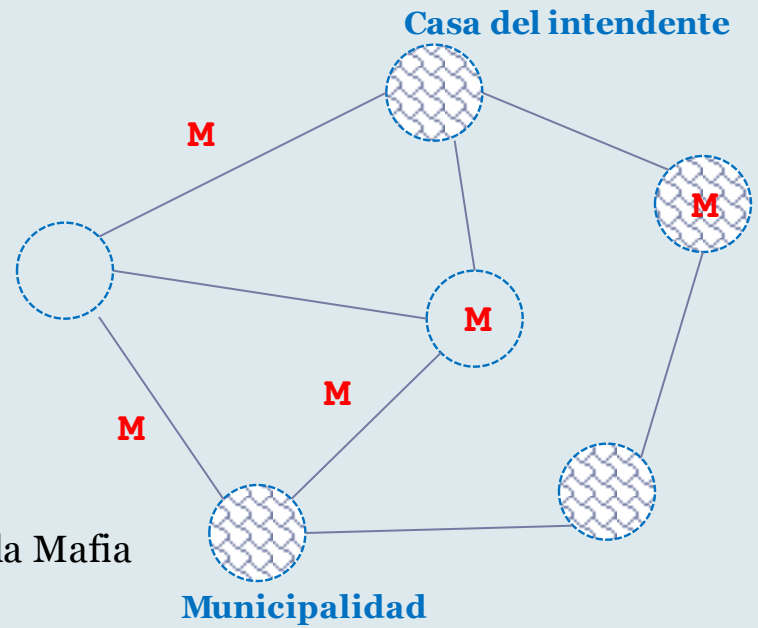
Ejercicio Parcial (curso 2012) - Enunciado

"El Paso City", años 20. Las mafias controlan varios sitios y calles de la ciudad. El intendente que debe desplazarse diariamente en su auto desde su residencia a la municipalidad, está seriamente amenazado.

Ud. debe ayudar al intendente encontrando la ruta más segura para realizar su traslado diario implementando en Java un método que retorne la ruta que pase por el menor número de calles y sitios controlados por la mafia. (En caso de existir más de una ruta con retornar alguna de ellas alcanzará).

La ciudad se describe como un conjunto de n sitios y varias calles bidireccionales que unen esos sitios. Cada sitio tiene la información si está controlado por la mafia o no. Lo mismo sucede con cada una de las calles de la ciudad.

Ejemplo



La ruta sombreada es la ruta más segura ya que pasa por un único lugar controlado por la mafia.

Referencia
M: Controlado por la Mafia

Grafos
Explicación del Problema

El problema se puede modelar representando a la ciudad con un grafo no dirigido pesado, donde los vértices son los sitios y las aristas las calles que los conectan:

- Cada vértice (sitio) guardará la información sobre la existencia de mafia en el mismo.
- Al igual que cada arista (calle).

Para poder distinguir a la "casa del intendente" y a la "municipalidad" del resto de los sitios, cada vértice almacenará además un nombre que lo identifique.

Entonces el problema consiste en buscar el camino con el menor número de mafias entre dos vértices del grafo: el vértice que representa a la "casa del intendente" y el que representa a la "municipalidad".

Esto se logra recorriendo el grafo exhaustivamente de forma de obtener todos los caminos posibles entre un origen y un destino y quedarse con el que contabilice el mínimo número de mafias.

Qué recorrido usar: DFS ó BFS ???

Grafos

Cuál Recorrido Usamos DFS o BFS??

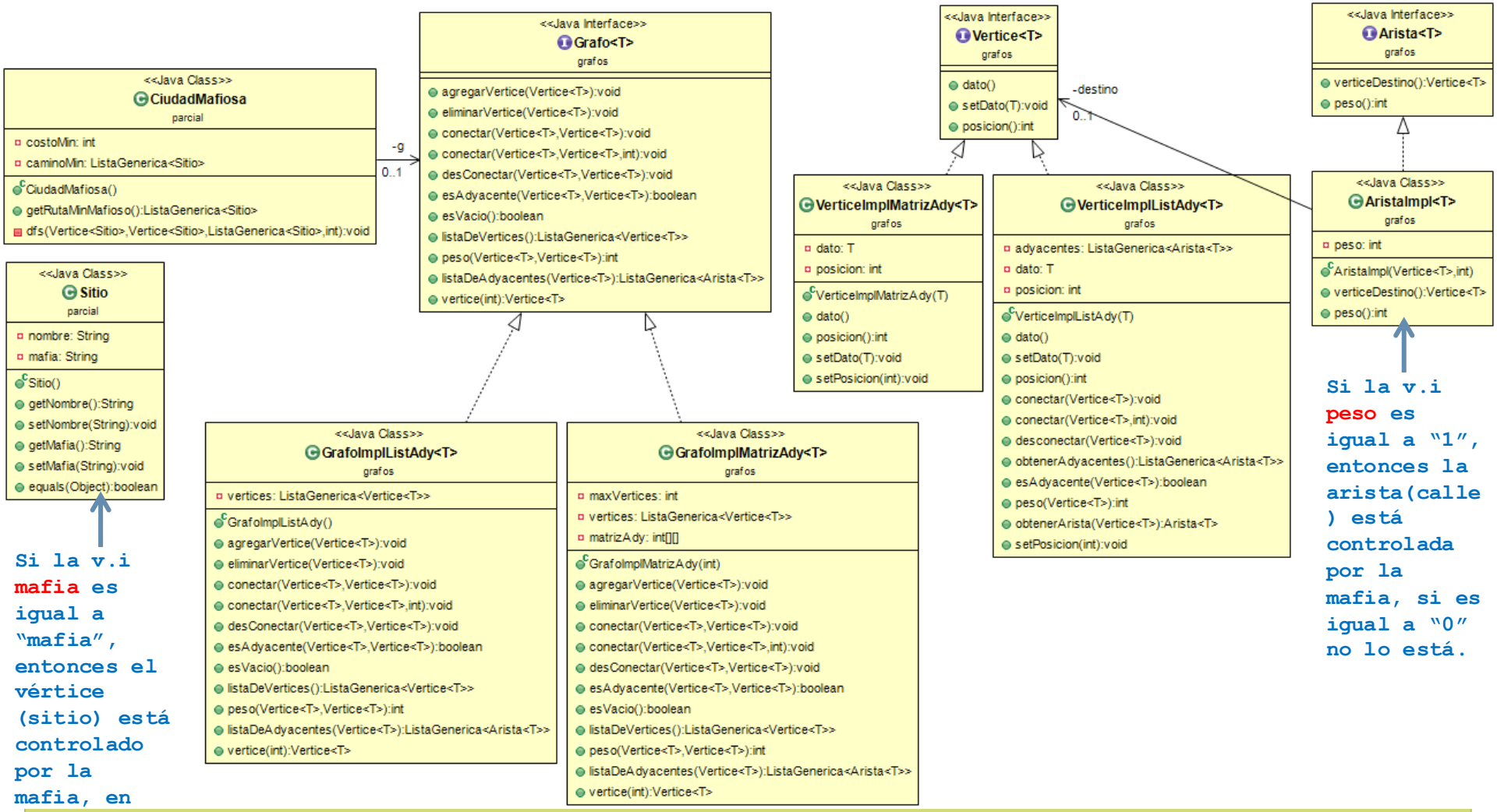
La forma mas directa de resolverlo es un con un **DFS**, probando todas las posibles combinaciones de nodos y aristas del grafo, partiendo desde un vértice inicial y llegando a un vértice destino.

El recorrido DFS se ejecuta de forma recursiva sobre cada nodo del grafo haciendo **una búsqueda exhaustiva** por cada posible camino que se puede tomar entre ese nodo y el resto de los nodos del grafo.

A diferencia del DFS, en un recorrido **BFS** **NO** se evalúan todas las posibles combinaciones de caminos. En un DFS, cada vértice puede ser visitado mas de una vez, mientras que en un BFS cada vértice puede ser visitado a lo sumo una única vez.

Grafos

Modelado de Clases Java (Versión 1)



Grafos
Implementación Java


```
public class Sitio {
        private String nombre;
        private String mafia;

       public Sitio() {
       }
       public String getNombre() {
            return nombre;
       }
       public void setNombre(String nombre) {
            this.nombre = nombre;
       }
       public String getMafia() {
           return mafia;
       }
       public void setMafia(String mafia) {
           this.mafia = mafia;
       }
       @Override
       public boolean equals(Object argo) {
            if (argo instanceof Sitio)
                return this.getNombre().equals(((Sitio) argo).getNombre());
            return false;
       }
}
```

Grafos
Implementación Java

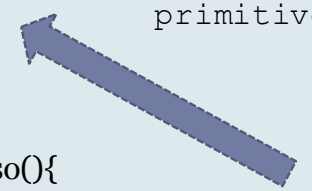

```
public class CiudadMafiosa {

    private Grafo<Sitio> ciudad;
    private int costoMin;
    private ListaGenerica<Sitio> caminoMin;


...
public ListaGenerica<Sitio> getRutaMinMafioso(){
      // Implementado en el siguiente slide
    }

...
}
```

costoMin: una variable de instancia para mantener la cantidad de mafias mínimas del camino. Pasaje de parámetros de tipos primitivos.



caminoMin: una variable de instancia para mantener el camino mínimo al volver de la recursión.

Grafos
Implementación Java


```
public ListaGenerica<Sitio>getRutaMinMafioso() {

    ListaGenerica<Vertice<Sitio>> vertices  = g.listaDeVertices();
    boolean seguir = true;
    ListaGenerica<Sitio> camino = new ListaEnlazadaGenerica<Sitio>>();
    caminoMin = new ListaEnlazadaGenerica<Sitio>>();
    Vertice<Sitio> vIni = null;
    Vertice<Sitio> vFin = null;
    this.costoMin = Integer.MAX_VALUE;
    int mafias = 0;
    vertices.comenzar();
    while(!vertices.fin()  &&  seguir) {
        Vertice<Sitio> v = vertices.proximo();
        if (v.dato().getNombre().equals ("casa del intendente"))
            vIni = v;
        if (v.dato().getNombre().equals("municipalidad"))
            vFin = v;
        if (vIni != null  &&  vFin  != null)
            seguir = false;
    }
    dfs (vIni,  vFin,  camino,  mafias);
    return caminoMin;
}
```

Buscando los vértices que representan la casa del intendente y la intendencia

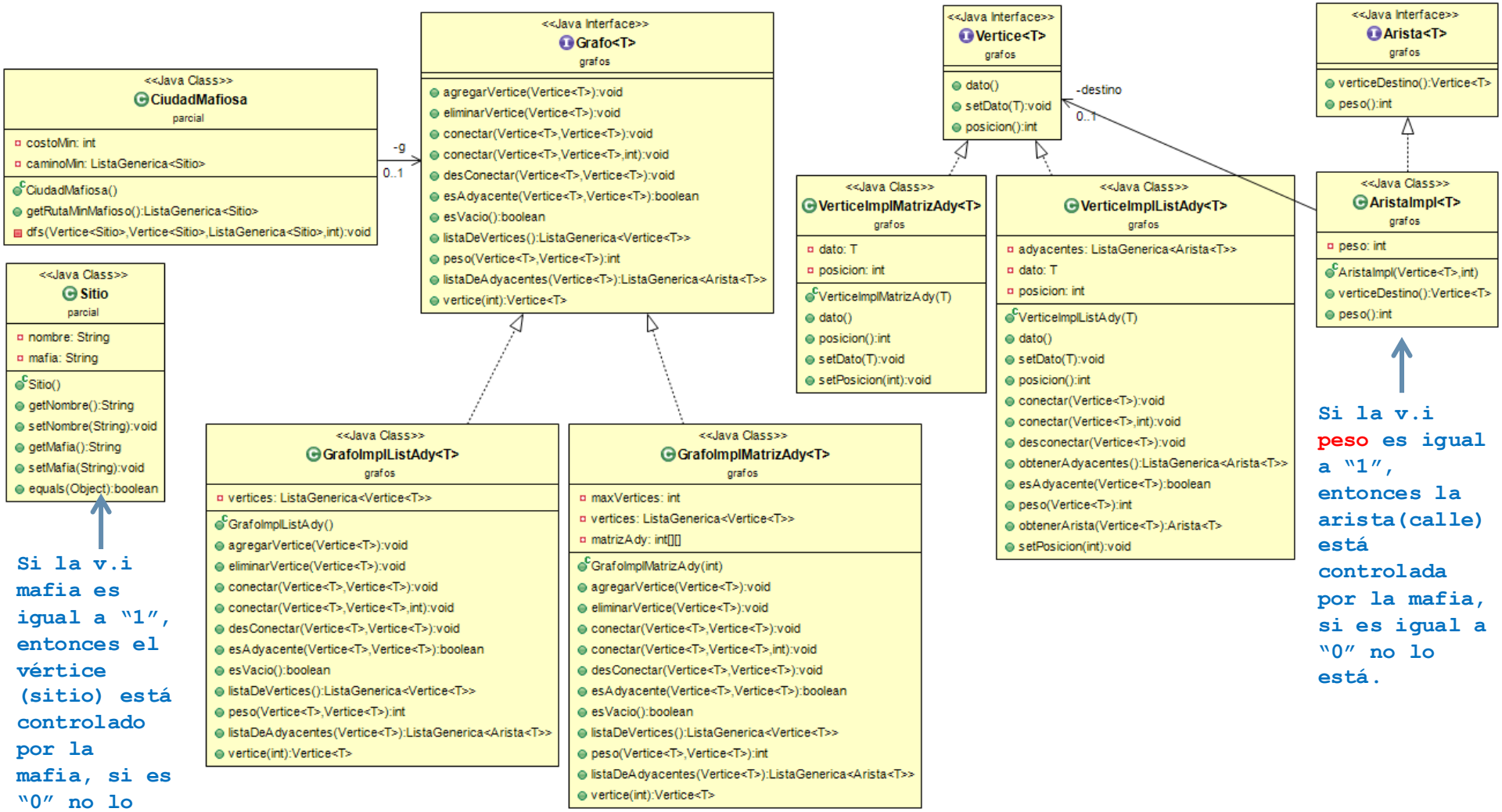
Grafos

Implementación Java


```
private void dfs(Vertice<Sitio> entrada, Vertice<Sitio> salida, ListaGenerica<Sitio> camino, int mafia){
    Boolean AM = false;    Boolean VM = false;
    camino.agregarFinal(entrada.dato());
    if(entrada.equals(salida)) {
       if (mafias < this.costoMin) {this.costoMin = mafias; this.caminoMin = camino.clonar();}        // Actualiza el costo mínimo y el camino
    } else {
          ListaGenerica<Arista<Sitio>> adyacentes =   g.listaDeAdyacentes(entrada);
          adyacentes.comenzar();
          for (int j  = 0; j < adyacentes.tamanio(); j++) {
                    Arista<Sitio> arista = adyacentes.proximo();
                    if (!camino.incluye(arista.verticeDestino())) {
                        if (arista.peso() ==  1){                                        // Calle controlada por la mafia
                              mafias ++;   AM = true;}
                        if (arista.verticeDestino().dato().getMafia().equals("mafia")){       // Sitio controlado por la mafia
                            mafias ++;   VM = true;}
                       dfs(arista.verticeDestino(), salida, camino, mafias);
                       if (AM) {
                            mafias--;   AM = false; }
                       if (VM) {
                            mafias--;    VM = false; }
                    }
          }
     }
    camino.eliminarEn(camino.tamanio() - 1);
}
```

Grafos

Modelado de Clases Java (Versión 2)



Grafos
Implementación Java


```
public class Sitio {
        private String nombre;
        private int mafia;

       public Sitio() {
       }
       public String  getNombre () {
            return nombre;
       }
       public void setNombre (String nombre) {
            this.nombre = nombre;
       }
       public int getMafia () {
           return mafia;
       }
       public void setMafia (int mafia) {
           this.mafia = mafia;
       }
       @Override
       public boolean equals(Object argo) {
            if (argo instanceof Sitio)
                return this.getNombre().equals(((Sitio) argo).getNombre());
            return false;
       }
}
```

Grafos
Implementación Java


```
private void dfs(Vertice<Sitio> entrada, Vertice<Sitio> salida, ListaGenerica<Sitio> camino, int mafias){
    Boolean AM = false;    Boolean VM = false;
    camino.agregarFinal (entrada.dato());
    mafias = mafias + entrada.dato().getMafia();          // Si es sitio mafioso suma 1, sino suma 0

    if (entrada.equals (salida)) {
       if (mafias  <  this.costoMin) {
            this.costoMin = mafias;                    // Actualiza el costo mínimo y el camino
            this.caminoMin = camino.clonar();
       }
    }
    else {
        ListaGenerica<Arista<Sitio>> adyacentes =   g.listaDeAdyacentes(entrada);
        adyacentes.comenzar();
        for (int  j = 0; j < adyacentes.tamanio();  j++) {
                Arista<Sitio> arista =  adyacentes.proximo());
                if (! camino.incluye(arista.verticeDestino())) {
                     mafias = mafias + arista.peso();                          // Si la calle es mafiosa sumará 1, sino 0
                     dfs(arista.verticeDestino(), salida, camino, mafias,);
                     mafias = mafias - arista.peso ();
                }
        }
    }
    camino.eliminarEn (camino.tamanio() - 1);
    mafias= mafias - entrada.dato ().getMafia();
}
```

Grafos
Implementación Java

Para pensar:

Si quisiéramos que **getRutaMinMafioso()** retornara el camino mínimo y el costo mínimo, qué cambios habría que realizar en el modelado de clases e implementación del método???