

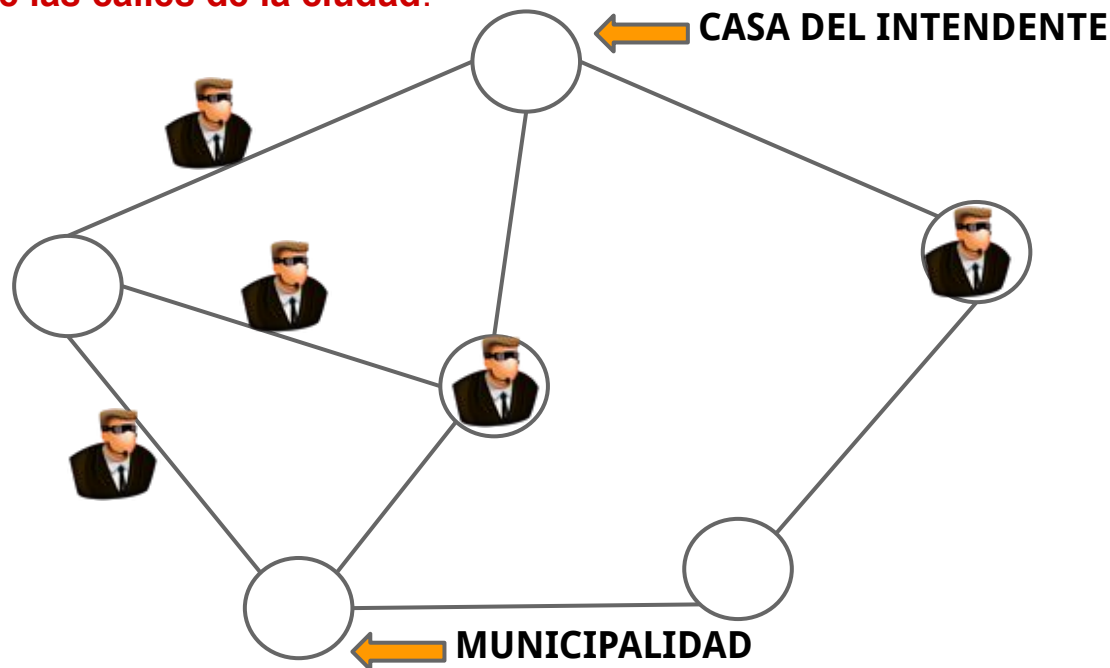
Grafos I

Ciudad Mafiosa

“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.

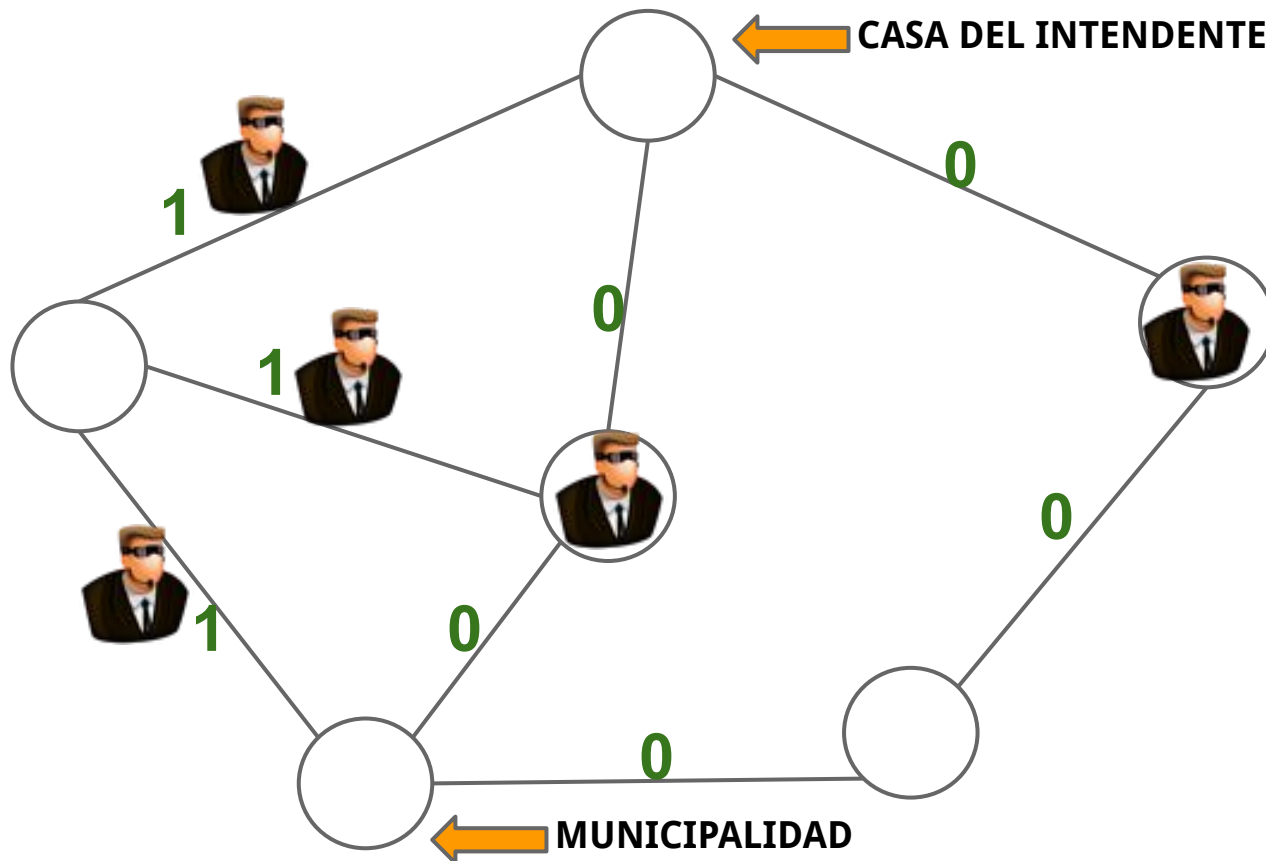
Ud. debe ayudar al intendente encontrando la ruta más segura para realizar su traslado diario implementando en Java un método que retorne la ruta que pase por el menor número de calles y sitios controlados por la mafia. (En caso de existir más de una ruta con retornar alguna de ellas alcanzará).

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.**

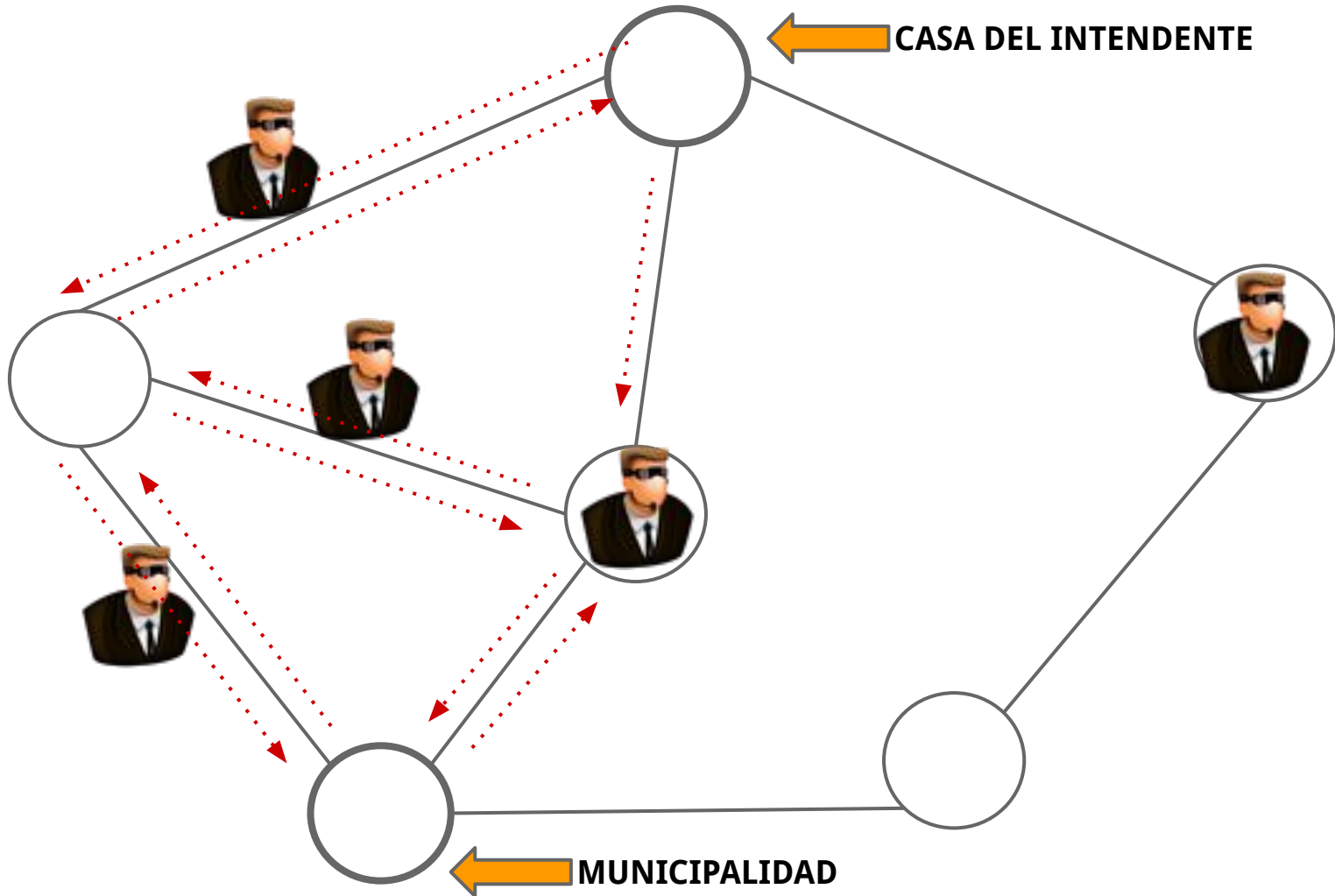


Definiciones

- Grafo de “**Sitio**”
- Cada **Sitio** tiene un nombre (String) sabe si tiene mafia (boolean)
- La calle sabe si tiene mafia (peso de la arista)



Recorrido



```
public class Sitio {
```

```
    private String nombre;  
    private boolean tieneMafia;
```

```
    public String getNombre() {  
        return nombre;  
    }
```

```
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }
```

```
    public boolean getTieneMafia() {  
        return tieneMafia;  
    }
```

```
    public void setTieneMafia(boolean tieneMafia) {  
        this.tieneMafia = tieneMafia;  
    }
```

```
@Override
```

```
    public boolean equals(Object arg0) {  
        if (arg0 instanceof Sitio)  
            return this.getNombre().equals(((Sitio) arg0).getNombre());  
        return false;  
    }
```

```
}
```

```
public ListaGenerica<Sitio> getRutaMinMafia() {  
  
    ListaGenerica<Sitio> camino = new ListaGenericaEnlazada<Sitio>();  
    int mafias = 0;  
  
    Vertice<Sitio> vIni = null;  
    Vertice<Sitio> vFin = null;  
  
    ListaGenerica<Sitio> caminoMin = new ListaGenericaEnlazada<Sitio>();  
    int[] mafiasMin = { Integer.MAX_VALUE };  
  
    ListaGenerica<Vertice<Sitio>> vertices = gCiudad.listaDeVertices();  
    boolean[] visitados = new boolean[vertices.tamano()];  
    vertices.comenzar();  
    while (!vertices.fin() && (vIni == null || vFin == null)) {  
        Vertice<Sitio> v = vertices.proximo();  
        if (v.dato().getNombre().equals("casa del intendente"))  
            vIni = v;  
        if (v.dato().getNombre().equals("municipalidad"))  
            vFin = v;  
    }  
    dfs(vIni, vFin, visitados, camino, caminoMin, mafias, mafiasMin);  
    return caminoMin;  
}
```



```

private void dfs(Vertice<Sitio> entrada, Vertice<Sitio> salida,
    boolean[] visitados, ListaGenerica<Sitio> camino,
    ListaGenerica<Sitio> caminoMin, int mafias, int[] mafiasMin) {

    visitados[entrada.posicion()] = true;
    camino.agregarFinal(entrada.dato());
    if (entrada.dato().getTieneMafia())
        mafias++;
    if (entrada.equals(salida)) {
        if (mafias < mafiasMin[0]) {
            mafiasMin[0] = mafias; // Actualiza resultados
            this.copiar(camino, caminoMin);
        }
    } else {
        ListaGenerica<Arista<Sitio>> adyacentes = gCiudad.listaDeAdyacentes(entrada);
        adyacentes.comenzar();
        while (!adyacentes.fin()) {
            Arista<Sitio> arista = adyacentes.proximo();
            if (!visitados[arista.verticeDestino().posicion()]) {
                dfs(arista.verticeDestino(), salida, visitados, camino, caminoMin,
                    mafias + arista.peso(), mafiasMin);
            }
        }
    }
    visitados[entrada.posicion()] = false;
    camino.eliminarEn(camino.tamano()-1);
}

```

```
private void copiar(ListaGenerica<Sitio> listaAux, ListaGenerica<Sitio> listaFinal) {  
    int tamanoListaFinal = listaFinal.tamano();  
    for (int i = 1; i <= tamanoListaFinal; i++)  
        listaFinal.eliminarEn(0);  
  
    int tamanoListaAux = listaAux.tamano();  
    for (int i = 0; i < tamanoListaAux; i++)  
        listaFinal.agregarFinal(listaAux.elemento(i));  
}
```