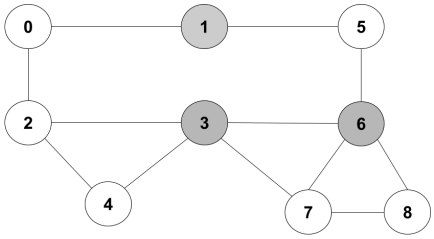
Examen de Algorítmica y Complejidad (Plan 2014)

1 de diciembre de 2023

N° matrícula:_____ **Nombre:** _____

Apellidos: _____

Problema (**2.5 puntos**). Tenemos un conjunto de estaciones conectadas unas con otras, representadas como nodos en un grafo **no** dirigido **no** valorado, y queremos encontrar el mínimo número de estaciones necesarias para poder llegar desde éstas al resto de estaciones mediante una única conexión. Ejemplo: en el siguiente grafo con 9 estaciones:



la selección de las estaciones {1,3,6} permite conectarse directamente al resto de estaciones del grafo (las estaciones 0 y 5 desde 1, las estaciones 2, 4 y 7 desde 3, y las estaciones 5, 7 y 8 desde 6).

SE PIDE: Implementar un algoritmo en Java, basado en el ***esquema Voraz***, que ofrezca esta funcionalidad. El algoritmo deberá tener la siguiente cabecera:

```
ArrayList<Integer> estacionesMinimo(boolean[][] grafo)
```

donde *grafo* es la matriz de adyacencia que indica si dos estaciones están conectadas (true) o no (false). El número de vértices del grafo determina el número de estaciones del problema (numeradas consecutivamente del 0 en adelante). El método deberá devolver un ArrayList de *Integer* con los índices de las estaciones seleccionadas. Se podrán implementar todos los métodos adicionales que se consideren necesarios.

```
// La estrategia voraz considerada va seleccionando la estación que esté conectada
a más estaciones no incluidas ni en la solución ni entre las ya conectadas a
través de algún elemento de la solución.

// Se requiere llevar un conteo (mediante array de cubiertos) de qué estaciones
ya han sido incluidas en la solución o bien están conectadas a través de esta.
Se ha encontrado solución cuando en el array de cubiertos están todas a true

ArrayList<Integer> estacionesMinimo(boolean[][] grafo){
    ArrayList<Integer> candidatos = new ArrayList<Integer>();
    for (int i = 0; i < grafo.length; i++) candidatos.add(i);
    ArrayList<Integer> solucion = new ArrayList<Integer>();
    boolean[] cubiertos = new boolean[grafo.length];
    for (int i=0; i< cubiertos.length; i++) cubiertos[i]=false;
    int c;
    while (!todosCubiertos(cubiertos)){
        c = seleccionarCandidato(candidatos, grafo, cubiertos);
        candidatos.remove(new Integer(c));
        solucion.add(c);
        cubrirNodos(c, grafo, cubiertos);
    }
    return solucion;
}
```

```java
boolean todosCubiertos(boolean[] cubiertos){
    boolean ok=true;
    int i=0;
    while ((i<cubiertos.length) && ok) {
        ok = cubiertos[i];
        i++;
    }
    return ok;
}

int contarConectadosNoCubiertos(int nodo, boolean[][] grafo, boolean[] cubiertos){
    int cont = cubiertos[nodo] ? 0 : 1; // se cuenta también el nodo candidato
    for (int i = 0; i < grafo.length; i++){
        if (!cubiertos[i] && grafo[nodo][i])
            cont++;
    }
    return cont;
}

int seleccionarCandidato(ArrayList<Integer> candidatos, boolean[][] grafo,
                         boolean[] cubiertos){
    int cMejor = candidatos.get(0);
    int mejor= contarConectadosNoCubiertos(cMejor, grafo, cubiertos);
    int actual;
    for (int i=1; i<candidatos.size(); i++){
        actual = contarConectadosNoCubiertos(candidatos.get(i), grafo, cubiertos);
        if (actual > mejor){
            cMejor = candidatos.get(i);
            mejor = actual;
        }
    }
    return cMejor;
}

void cubrirNodos(int nodo, boolean[][] grafo, boolean[] cubiertos){
    cubiertos[nodo] = true;
    for (int i = 0; i < grafo.length; i++){
        if (!cubiertos[i] && grafo[nodo][i])
            cubiertos[i] = true;
    }
}
```

SOLUCIÓN V2

```
// Se usa la misma estrategia voraz, pero se crea también un array para contar
el número de conexiones de cada nodo (que va decrementando según se van cubriendo)
y un entero que cuente el número de nodos no cubiertos (se encuentra solución
cuando llega a 0)

ArrayList<Integer> estacionesMinimo(boolean[][] grafo){
    ArrayList<Integer> solucion = new ArrayList<Integer>();
    int[] numConexiones = new int[grafo.length];
    int numNodosNoConectados = grafo.length;
    boolean[] cubiertos = new boolean[grafo.length];
    inicializar(grafo, numConexiones, cubiertos);
    while (numNodosNoConectados > 0){
        int c = seleccionarCandidato(numConexiones);
        solucion.add(c);
        numConexiones[c] = -1; //aseguramos que no se volverá a seleccionar c
        for (int i=0; i<grafo.length;i++) {
            //cubrimos el nodo seleccionado y los nodos adyacentes a c
            if (!cubiertos[i] && (i==c || grafo[c][i])) {
                numNodosNoConectados--;
                cubrirNodo(i,cubiertos,grafo,numConexiones);
            }
        }
    }
    return solucion;
}

void inicializar(boolean[][] grafo, int[] numConexiones, boolean[] cubiertos){
    for (int i = 0; i < grafo.length; i++){
        numConexiones[i] = 0;
        cubiertos[i]= false;
        for (int j = 0; j < grafo.length; j++) {
            if (grafo[i][j])
                numConexiones[i]++;
        }
    }
}

int seleccionarCandidato(int[] numConexiones){
    int mejor = 0;
    for (int i=0; i<numConexiones.length; i++){
        if (numConexiones[i] > numConexiones[mejor])
            mejor = i;
    }
    return mejor;
}

void cubrirNodo(int v, boolean[] cubiertos, boolean[][] grafo,
                int[] numConexiones){
        cubiertos[v]=true;
        for (int i=0; i<grafo.length; i++){
            if (grafo[i][v])
                numConexiones[i]--;
        }
}
```