novática
CEPIS UPGRADE Revista de la Asociación de Técnicos de Informática

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software). ***Novática*** co-edita asimismo **UP**GRADE, revista digital de **CEPIS** (*Council of European Professional Informatics Societies*), en lengua inglesa, y es miembro fundador de **UP**ENET (***UP****GRADE* **E***uropean* **NET***work*).

<http://www.ati.es/novatica/>
<http://www.ati.es/reicis/>
<http://www.cepis.org/upgrade>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y es representante de España en **IFIP** (*International Federation for Information Processing*); tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*), así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2**, **ASTIC**, **RITSI** e **Hispalinux**, junto a la que participa en **ProInnova**.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn./fax 963330392 <secreval@ati.es>
Administración y Redacción ATI Cataluña
Via Laietana 46, ppal. 1ª. 08003 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Andalucía <secreand@ati.es>
Redacción ATI Galicia <secregal@ati.es>
Suscripción y Ventas <http://www.ati.es/novatica/interes.html>, ATI Cataluña, ATI Madrid
Publicidad
Padilla 66, 3º, dcha., 28006 Madrid
Tlnf 914029391; fax.913093685 <novatica@ati.es>
Imprenta: Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: Luces feéricas - Concha Arias Pérez / © ATI
Diseño: Fernando Agresta / © ATI 2003

Nº 210, marzo-abril 2011, año XXXVII

sumario

Monografía del próximo número: "*Business Intelligence*"

Mi número de Erdos

Julio Javier Castillo, Diego Javier Serrano
Laboratorio de Investigacion de Software MsLabs, Dpto. Ing. en Sistemas de Información, Facultad Regional Córdoba - Universidad Tecnológica Nacional (Argentina)

<jotacastillo@gmail.com>, <diegojserrano@gmail.com>

El enunciado de este problema apareció en el número 208 de ***Novática*** (noviembre-diciembre 2010, p. 76).

El programa presentado a continuación ha sido modelado desde el paradigma orientado a objetos, y en base al mismo, se definen las siguientes clases: *Amigo*, *Grafo* y *Main*. La idea básica para resolver el problema será modelar la red social como un grafo dirigido, y luego aplicar sobre él algún algoritmo que nos permita calcular el número de Erdos de cualquiera de los amigos de la red, respecto de otro conjunto de amigos de la red.

Para resolver este problema se ha elegido el algoritmo de Dijkstra o algoritmo del camino más corto, basándonos en la idea de que una relación de amistad es una arista dirigida de un nodo a otro de la red, y etiquetando todas las relaciones de la red con valor de uno. Así, podríamos situarnos en un nodo X, al que llamaremos nodo base-Erdos, y calcular el número X-Erdos de un conjunto de nodos en base a su relación con el nodo X.

Debido a que este proceso debe realizarse repetidas veces y cambiando la base-Erdos, es que se debe ejecutar el algoritmo de Dijkstra tantas veces como bases Erdos distintas nos exija el problema.

La clase *Amigo* contiene un identificador numérico, un nombre y un conjunto de Amigos con los cuales se mantiene una relación de amistad. Esta clase es la abstracción de un nodo de la red social que se está modelando. Vale notar que la relación de amistad es una relación unidireccional, pues A podría declararse amigo de B, pero no necesariamente B tiene que considerar que A es su amigo.

Tres campos privados adicionales (anterior, distanciaTotal, y finalizado) serán empleados por nuestro algoritmo de camino más corto, al momento de recorrer la red.

La clase *Grafo* contiene un conjunto de nodos, es decir, es la abstracción de nuestra red social. Esta clase es la que nos permitirá crear nodos, y establecer relaciones de amistad entre ellos, como así también, contendrá el método clave *dijkstra* que nos permitirá calcular el camino más corto entre un nodo origen y un nodo destino. Nótese, que la longitud de este camino será el numero de Erdos del nodo destino respecto del nodo origen, lo cual es justamente el problema que precisamos resolver.

La clase *Main* contiene el método principal *main*, mediante el cual solicitaremos el ingreso de datos de la entrada estándar. Se realiza la lectura de la cantidad de casos de prueba a procesar, la cantidad de miembros que habrá en la red social, y las relaciones de amistad de esa red. Toda esta información es esencial para construir la red social.

Como entrada, se ingresará la cantidad de miembros que serán base del cálculo (base-Erdos), y los miembros de la red a los que se les desea calcular su número de Erdos basado en el miembro de origen. Primero, se lee la cantidad de casos de prueba. Luego, por cada caso de prueba se leen la cantidad de miembros de esa red social (variable miembros), y la cantidad de relaciones entre ellos (variable aristas) todos se guardan convenientemente en estructuras de datos. Seguidamente, se leen cada uno de los miembros y su nombre, esto se refleja agregando un nuevo nodo en nuestro grafo g (método: *AgregarAmigo()*), por cada miembro. Luego, las relaciones entre esos miembros son agregados como nuevas aristas del grafo g (método: *AgregarAmistad()*).

A continuación, se lee de la entrada la cantidad de miembros que serán base de cálculo del número de Erdos; y luego, se leen los miembros de la red a los cuales se les calculara su numero de Erdos, tomando esa base como pivot. A medida que se van leyendo estos miembros, invocamos al método *dijkstra* que nos permite calcular el camino más corto entre el miembro base y el miembro actual. Nótese que la red fue inicializada con pesos unitarios.

Luego, por cada base que cambiamos necesitamos volver a reinicializar el grafo, volviendo a crear los nodos y las conexiones de la red social. Finalmente, se imprime por pantalla la salida en el formato indicado por el problema. La leyenda "*Infinito*" será impresa por pantalla, en el caso de que no haya relaciones entre esos miembros de la red, es decir cuando esos miembros no se conocen.

```
import  java.util.HashMap;
import  java.util.HashSet;
import  java.util.LinkedList;
import  java.util.Scanner;

class Amigo {
   public  String  nombre;
   public  HashSet<Integer>  amigos;
   public  int  idMiembro;
   public  Amigo  anterior;
   public  int  distanciaTotal  =  -1;
   public  boolean  finalizado;

   public  Amigo(int  idMiembro,String  nombre)  {
      amigos  =  new  HashSet<Integer>();
      this.idMiembro  =  idMiembro;
      this.nombre=nombre;
   }

   public  void  limpiar()  {
      anterior  =  null;
      distanciaTotal  =  -1;
      finalizado  =  false;
   }
}

class Grafo {
   private  HashMap<Integer,  Amigo>  miembros;

   public  Grafo()
   {
      miembros  =  new  HashMap<Integer,  Amigo>();
   }

   public  void  agregarAmigo(Integer  idMiembro,String  nombre)  {
      miembros.put(idMiembro,  new  Amigo(idMiembro,nombre));
   }

   public  void  agregarAmistad(Integer  idAmigo1,  Integer  idAmigo2)  {
```

```
        miembros.get(idAmigo1).amigos.add(idAmigo2);
    }

    public String obtenerNombre(Integer idAmigo) {
        return miembros.get(idAmigo).nombre;
    }

    public Amigo dijkstra(Integer amigo1, Integer amigo2) {
        Amigo origen = miembros.get(amigo1), destino = miembros.get(amigo2);

        LinkedList<Amigo> cola = new LinkedList<Amigo>();
        cola.add(origen);
        origen.distanciaTotal = 0;
        Amigo x = null;
        while(!cola.isEmpty()){
            x = cola.removeFirst();
            for(Integer am: x.amigos) {
                Amigo y = miembros.get(am);
                if (!y.finalizado) {
                    cola.addLast(y);
                    int dt = x.distanciaTotal + 1;
                    if (y.distanciaTotal == -1 || dt < y.distanciaTotal){
                        y.anterior = x;
                        y.distanciaTotal = x.distanciaTotal + 1;
                    }
                }
            }
            x.finalizado = true;
        }
        return destino;
    }

    public void reiniciar() {
        for(Amigo x:miembros.values()) x.limpiar();
    }
}

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int i,j, miembros, aristas;
        String nombre;
        int idMiembro, origen, destino;
        int cantErdos, baseErdos, miembrosErdos;
        int casos = sc.nextInt();

        for(int c = 0; c < casos; c++) {
            Grafo g = new Grafo();
            miembros = sc.nextInt();
            aristas = sc.nextInt();
            for(i=0; i<miembros; i++) {
                idMiembro = sc.nextInt();
                nombre = sc.nextLine().trim();
                g.agregarAmigo(idMiembro,nombre);
            }

            for(i=0; i<aristas; i++) {
                origen = sc.nextInt();
                destino = sc.nextInt();
                g.agregarAmistad(origen ,destino);
            }

            cantErdos = sc.nextInt();   ///cant. de num. de Erdos a calcular
            for(i=0; i<cantErdos; i++) {
                baseErdos = sc.nextInt();
                miembrosErdos = sc.nextInt();
                System.out.println(g.obtenerNombre(baseErdos) + ":");
                for(j=0; j<miembrosErdos; j++) {
                    int idMiembroACalcular = sc.nextInt();
                    Amigo x = g.dijkstra(baseErdos,idMiembroACalcular);

                    int val=-1;
                    while (x != null) {
                        val++;
                        x = x.anterior;
                    }
                    System.out.println(g.obtenerNombre(idMiembroACalcular)      +
"="+((val==0)?"Infinito": (val)));
                }
                g.reiniciar();
            }
        }
    }
}
```