Algoritmos
y
Estructuras de Datos

Cursada 2015

Grafos- Aplicación Recorridos
TP Nº 9 Ejercicio 10 – Grados de Separación

En nuestro interconectado mundo se especula que dos personas cualesquiera están relacionadas entre sí a lo sumo por 6 grados de separación. En este problema, debemos realizar un método para encontrar el **máximo grado de separación** en una red de personas, donde una arista entre dos personas representa la relación de conocimiento entre ellas, la cual es simétrica.

Entre dos personas, el grado de separación es el mínimo número de relaciones que son necesarias para conectarse entre ellas.

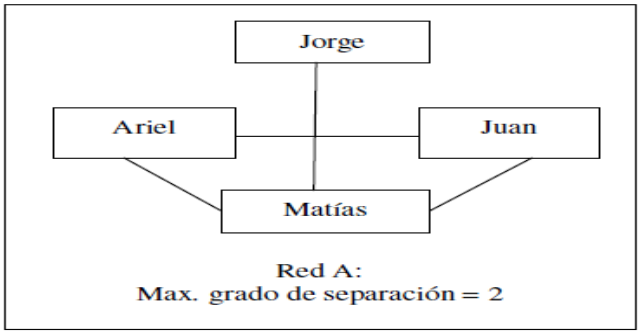
Si en la red hay dos personas que no están conectadas por una cadena de relaciones, el grado de separación entre ellas se considerará igual a 0.

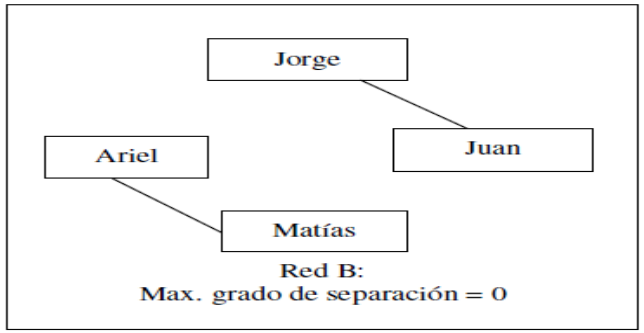
En una red, el **máximo grado de separación** es el mayor grado de separación entre dos personas cualesquiera de la red.

Implemente en JAVA una clase llamada **GradosDeSeparacion** ubicada dentro del paquete **ejercicio10**, cumpliendo la siguiente especificación:

maximoGradoDeSeparacion(Grafo<String> grafo) : int // Retorna el máximo grado de separación del grafo recibido como parámetro. Si en el grafo hubiera dos personas cualesquiera que no están conectadas por una cadena de relaciones entonces se retorna 0.

Ejemplo:





TP Nº 9 Ejercicio 10 – Grados de Separación
Análisis de Resolución

Del enunciado deducimos que la red de relaciones entre las personas está representada por un **Grafo No Dirigido**. Donde los **vértices** representan a las *personas* y las **aristas** el *conocimiento* entre ellas.

¿Cómo calculamos el máximo grado de separación del grafo?

- Para cada vértice del grafo deberíamos encontrar su **mayor grado de separación** con respecto a todos los demás vértices del grafo
- y quedarnos con el **máximo** de ellos, que sería el valor a retornar como **grado de separación del grafo.**

Entonces, el problema se reduce a:

¿Cómo calculamos el grado de separación de un vértice?

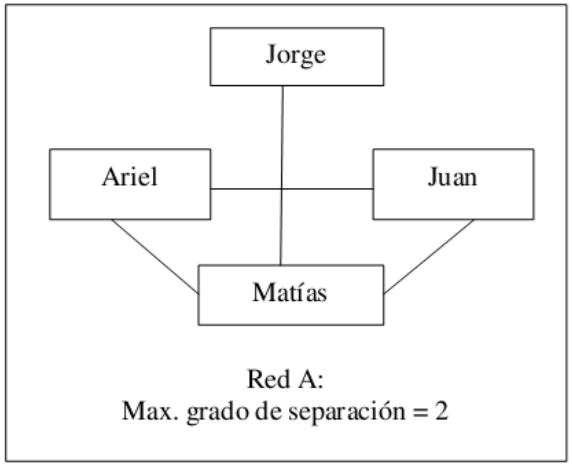
TP Nº 9 Ejercicio 10 – Grados de Separación Análisis de Resolución

Analicemos cómo calcular el **grado de separación de un vértice**:

Primero, recordemos su definición:

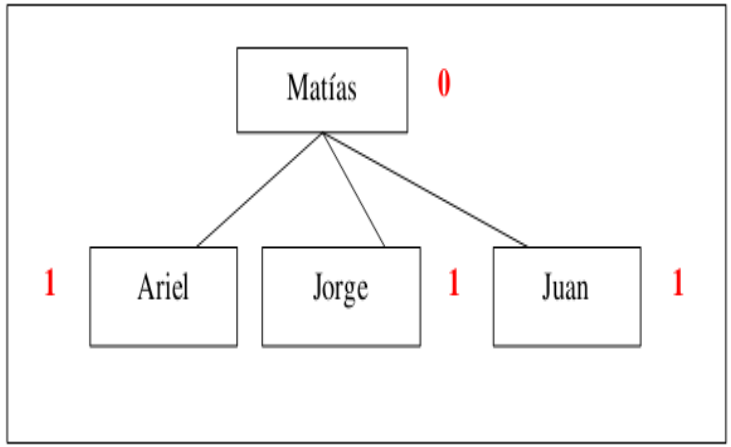
"Entre dos vértices (personas), el grado de separación es el **mínimo** número de relaciones que son necesarias para conectarse entre ellas".

Para el ejemplo propuesto:



Intuitivamente:

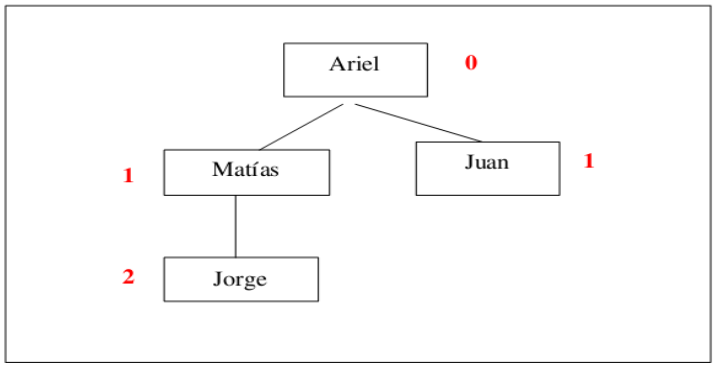
El **grado de separación** de "Matías" con los demás vértices será:



Un posible árbol que representa el recorrido de G a partir de "Matías". En el árbol se muestra fuera de los nodos la **distancia mínima** (**grado de separación**) de cada uno de los vértices a la raíz.

TP Nº 9 Ejercicio 10 – Grados de Separación Análisis de Resolución

Mientras que el **grado de separación** de "Ariel" con los restantes vértices será:



Un posible árbol que representa el recorrido de G a partir de "Ariel".

¿Qué estrategia aplicar para calcular estos grados de separación?

- Partimos del vértice a calcular su grado de separación.
- Recorremos el grafo usando un recorrido sistemático calculando la **mínima distancia** que lo conecta con los restantes vértices del grafo.

TP Nº 9 Ejercicio 10 – Grados de Separación Recorrido a Usar

La pregunta ahora es:

¿Qué recorrido de grafos conviene utilizar?

DFS o BFS????

TP Nº 9 Ejercicio 10 – Grados de Separación
Recorrido a Usar

El recorrido **BFS** (o búsqueda en amplitud) nos ofrece la solución más eficiente comparada con el DFS (o búsqueda en profundidad) para obtener los vértices más cercanos a otro vértice.

TP Nº 9 Ejercicio 10 – Grados de Separación
Análisis de Resolución

En el **BFS** los vértices se visitan en amplitud o también conocido por niveles. Este orden implica que, fijado un vértice, **primero** se visitan los vértices que se encuentran a distancia mínima 1 de él, luego los que se encuentran a distancia mínima 2, etc.

Entonces …

Si lanzamos un **BFS** desde un vértice v, calcularemos la distancia mínima entre v y todos los demás vértices del grafo G que es : "El grado de separación del vértice v con los restantes vértices!!"

Basta con devolver el mayor de todas esas distancias y ya tendremos el **grado de separación de v**.

Luego, el **grado de separación de G** será el máximo de los grados de separación de todos los vértices del mismo.

TP Nº 7 Ejercicio 10 – Grados de Separación Análisis de Resolución

Consideraciones:

- "Si en la red hay dos personas que no están conectadas por una cadena de relaciones, el **grado de separación** entre ellas se considerará igual a 0"

Esto significa que el **BFS lanzado desde un vértice NO pudo visitar todos los demás vértices del grafo.** En este caso el BFS devolverá 0.

- "Si en el grafo hubiera dos personas cualesquiera que no están conectadas por una cadena de relaciones entonces **maximoGradoDeSeparacion(Grafo<String> grafo) : int** retorna 0."

Esto significa que el grado de separación de un vértice dio 0. En este caso, **NO** seguimos calculando todos los restantes grados de separación de los demás vértices. **Cortamos el proceso** devolviendo 0.

TP Nº 7 Ejercicio 10 – Grados de Separación Implementación en Java


```
package ejercicio10;

  public class GradosDeSeparacion {

     public int maximoGradoDeSeparacion (Grafo<String> grafo) {

        ListaGenerica<Vertice<String>> vertices = grafo.listaDeVertices();
        boolean[] visitados = new boolean[vertices.tamanio()];
        int mgs = 0;
        boolean seguir = true;
        vertices.comenzar();
        while (!vertices.fin() && seguir) {
                Vertice<String> v = vertices.proximo();
                int gsv = gradoDeSeparacionVertice (grafo, v, visitados);
                if (gsv > mgs)
                   mgs = gsv;
                if (gsv == 0){
                   seguir = false;
                   mgs = 0;
                }
        }
        return mgs;
   }
```

TP Nº 9 Ejercicio 10 – Grados de Separación Implementación en Java Versión 1


```
private int gradoDeSeparacionVertice(Grafo<String> grafo, Vertice<String> origen, boolean[] visitados) {
        int cantVis = 0;    ColaGenerica< Vertice<String> > q = new ColaGenerica< Vertice<String> >();
        for (int i = 0; i<visitados.length; i++){
            visitados[i] = false;
        }
        q.encolar (origen);  q.encolar (null);  visitados[origen.posicion()] = true;   cantVis++;   int gradoSep = 0;
        while (!q.esVacia()){
              Vertice<String> elem = q.desencolar();
              if (elem!=null){
                   ListaGenerica<Arista<String>> adyacentes = grafo.listaDeAdyacentes((elem);
                   adyacentes.comenzar();
                   while (!adyacentes.fin()){
                         Arista<String> arista = adyacentes.proximo();
                         if (!visitados[arista.verticeDestino().posicion()]){
                               visitados[arista.verticeDestino().posicion()] = true;   cantVis++;
                               q.encolar (arista.verticeDestino());
                         }
                   } // del while
              } // del if
              else { gradoSep++;    //  elem  es null entonces procesé todo el nivel anterior!!!
                        if (!q.esVacia())
                             q.encolar(null);
                    }
        } // del primer while . La cola está vacía. Se procesaron todos los nodos alcanzables desde el origen
        if (cantVis < grafo.listaDeVertices().tamanio())
                return 0;
        else   return gradoSep;
  }
}
```

TP Nº 9 Ejercicio 10 – Grados de Separación Implementación en Java

Observación:

En la implementación del método **gradoDeSeparacionVertice** para distinguir la **separación de niveles** usamos una cola de Vertice <String> en la cuál para distinguir la separación de niveles encolamos null, a fin de mejorar la implementación presentamos en el siguiente slide una versión "*más purista*" que maneja con un *for* la cantidad de nodos existentes en cada nivel.

TP Nº 9 Ejercicio 10 – Grados de Separación Implementación en Java Versión 2


```
private int gradoDeSeparacionVertice (Grafo<String> grafo, Vertice<String> origen, boolean[] visitados) {

        int cantVis = 0;    ColaGenerica<Vertice<String>> q = new ColaGenerica<Vertice<String>>();
        for (int i = 0; i<visitados.length;i++){
                visitados[i] = false;
        }
        q.encolar (origen);  visitados[origen.posicion()] = true;   cantVis++;   int gradoSep = 0;
        while (!q.esVacia()){
            int tamanio = q.tamanio();
            for (int i = 0; i <tamanio; i++ ) {
               Vertice<String> elem = q.desencolar ();
                ListaGenerica<Arista<String>> adyacentes = grafo.listaDeAdyacentes(elem);
                adyacentes.comenzar();
                while (!adyacentes.fin()){
                    Arista<String> arista = adyacentes.proximo();
                    if (!visitados[arista.verticeDestino().posicion()]){
                        visitados[arista.verticeDestino().posicion()] = true;   cantVis++;
                        q.encolar (arista.verticeDestino());
                    }

                }  // del while
            }  // del for

            // procesé todo un nivel !!!
            gradoSep++;

        } // primer while . La cola está vacía. Se procesaron todos los nodos alcanzables desde el origen
        if (cantVis < grafo.listaDeVertices().tamanio())
             return 0;
         else   return gradoSep;
    }
```