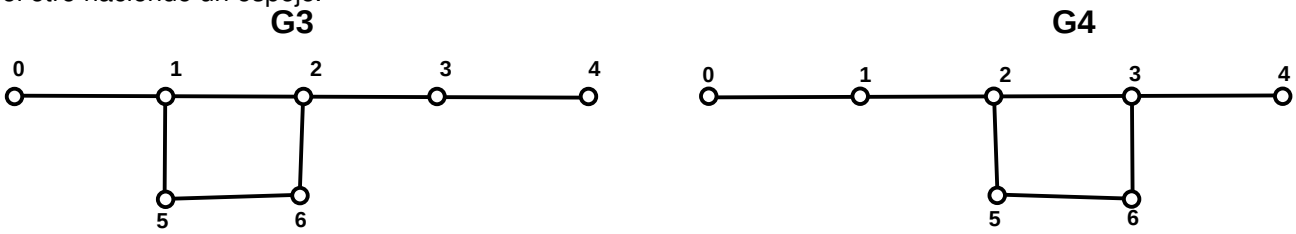
Curso de Programación en C++.
TPL3. Trabajo Práctico de Laboratorio 3. [2015-08-12]

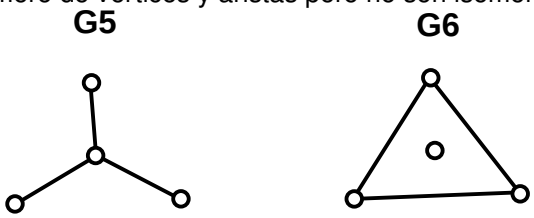
PASSWD PARA EL ZIP: MWC 48U 4E5 EN2

Ejercicios

[Ej. 1] **[isomorph]** Dos grafos `G1`, `G2` son isomorfos si existe una renumeración de los índices de los vértices tal que lleva `G1` a `G2`. Por ejemplo los dos grafos `G3` y `G4` de la figura son isomorfos, ya que uno se obtiene del otro haciendo un espejo.



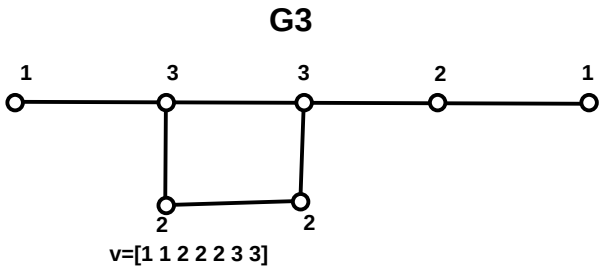
Dados dos grafos `G1,G2` no existen algoritmos de tiempo polinomial para determinar si son isomorfos, el algoritmo más simple conceptualmente consiste en explorar todas las posibles permutaciones de índices del grafo `G2` y ver si coincide con `G1`. Este algoritmo es al menos $O(N_v \cdot N_v!)$. Pero podemos encontrar tests que pueden descartar en forma relativamente rápida si dos grafos NO son isomorfos. Esto se basa en determinar propiedades que son **invariantes** ante una renumeración de los índices, por ejemplo la cantidad de vértices, y la cantidad de aristas. Sin embargo con esto no basta porque por ejemplo los dos grafos `G5,G6` tienen el mismo número de vértices y aristas pero no son isomorfos.

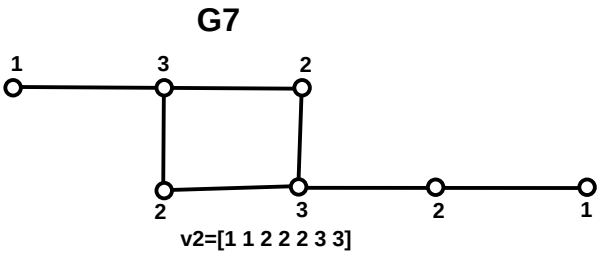


Esto es fácil de ver ya que hay otro invariante que es la **valencia** de sus vértices. La valencia de un vértice es la cantidad de aristas que salen del vértice, es decir el tamaño de su adyacencia. En `G5` hay un vértice con valencia 3 mientras que en `G6` hay uno con valencia 0. Entonces un predicado que puede determinar cuando los grafos NO son isomorfos podría ser determinar el vector de valencias ordenadas de menor a mayor y compararlos. En el ejemplo anterior tenemos

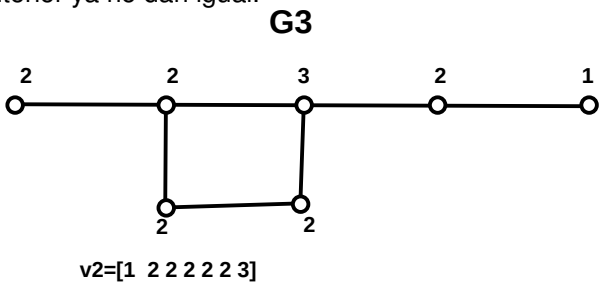
```
G5 -> [1,1,1,3]
G6 -> [0,2,2,2]
```

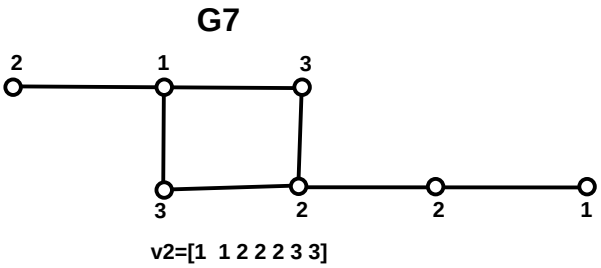
De todas formas, puede haber casos donde este criterio falle. Por ejemplo para los grafos `G4,G7` de la figura, las valencias ordenadas son `[1,1,2,2,2,3,3]`. (El número en el nodo es su valencia).





Una definición alternativa de valencia es que es el número de vértices a distancia 1. Podemos tomar como otro invariante el número de vecinos $v2$ a distancia 2. Podemos ver entonces que en el caso anterior ya no dan igual.





Por lo tanto sugerimos el siguiente algoritmo para determinar (en forma aproximada) si dos grafos son isomorfos

- Chequear si sus números de vértices son iguales.
- Calcular para cada vértice el número de vecinos a distancia 1 (valencia **v1**), ordenarlos y compararlos.
- Calcular para cada vértice el número de vecinos a distancia 2, ordenarlos y compararlos (**v2**).

Consigna: Es escribir una función
`bool isomorph(graph_t &G1,graph &G2);` que determina (en forma aproximada) si dos grafos son isomorfos.

Nota 1: El algoritmo descripto es **aproximado** en el sentido de que sólo puede dar falsos positivos, es decir, si **`isomorph(G1,G2)`** da **`false`** entonces los grafos seguramente NO son isomorfos. Pero si da positivo puede ser todavía que no sean ismorfos.

Nota 2: Se acepta cualquier otra propuesta (**`isomorph2()`**), mientras sea mejor o igual que la anterior. Es decir de un conjunto de falsos positivos menor o igual que el anterior. Es decir,

- Si son isomorfos debe dar **`true`**. (No debe haber falsos negativos).
- Si no son isomorfos y **`isomorph(G1,G2)`** da **`false`**, entonces **`isomorph2(G1,G2)`** debe dar **`false`**.

[Ej. 2] **[flux]** Escribir utilizando la librería Eigen una función
`double flux(shape_t &shape,vector_field_t &vf);` que calcula (2D) el flujo del campo vectorial $\mathbf{V}$ (dado por la clase **`vector_field_t`**) a través de la curva Γ (dada por la clase **`shape_t`**).

$$\Phi = \int_{\Gamma} \mathbf{V}(x) \cdot \hat{\mathbf{n}} \, \mathrm{d}\Gamma$$

```
class vector_field_t {
  public:
  virtual void field(MatrixXd &x,MatrixXd &V)=0;
};

class shape_t {
  public:
  virtual int size()=0;
  virtual void xnode(int j,MatrixXd &x)=0;
};
```

La forma geométrica consiste en una serie de **`N=shape.size()`** segmentos (elementos lineales) en 2D. El segmento **`j`** está definido por las coordenadas de los puntos **`j`** y **`j+1`**, por lo tanto hay **`N+1`** vértices. Para encontrar el vértice **`j`** debe hacerse **`shape.xnode(j,x)`**.

Dado un punto en el espacio **`x`** el campo vectorial en ese punto **`V`** se obtiene haciendo **`vf.field(x,V)`**.

Todos los vectores (posiciones y campo vectorial) son **`MatrixXd`** de tamaño **`(2,1)`**.

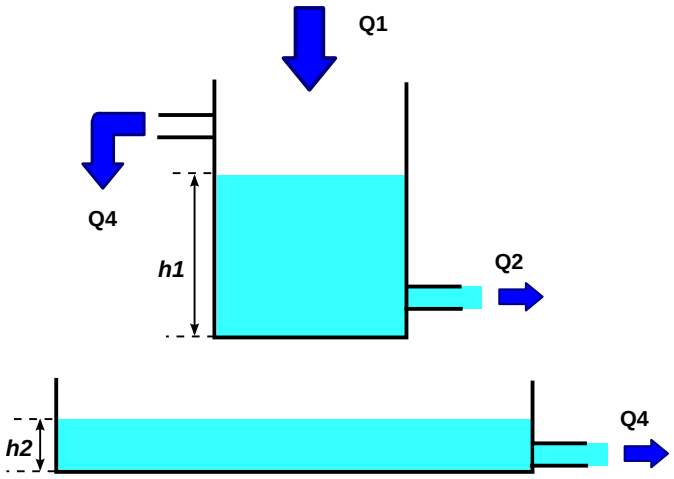
Ayuda:

- Recorrer los segmentos, calcular su punto medio **`xc`** y calcular el campo vectorial **`V`** llamando a **`vf.field(xc,V)`**.
- Calcular el área (con normal) $\hat{\mathbf{n}}\,\mathrm{d}Gamma$ del segmento $[x_j, x_{j+1}]$ haciendo $\Delta x = (x_{j+1} - x_j)$ y rotándolo 90 grados en sentido horario.
- Acumular en el flujo el producto escalar del campo con el área con normal.

La función de chequeo **`check_flux(flux)`** hace una serie de chequeos, con segmentos y círculos, la salida de los cuales debe dar:

```
flux (segment[1,0], uniform(0,1): -1.000000
flux (segment[1,0], uniform(1,0): 0.000000
flux (segment[0,1], uniform(1,0): 1.000000
flux (segment[0,1], uniform(0,1): 0.000000
flux (circle(c=(0,0),R=1), radial(c=(0,0),int=1): 6.279052
flux circle(c=(0,0),R=1), radial(c=(0,0),intens=1), N=1000: 6.283144
flux circle(c=(0,0),R=1), radial(c=(0,0),intens=100): 62.831440
flux circle(c=(0,0),R=1), sole(c=(0,0),intens=100): 0.000000
```

[Ej. 3] **[tanks]** Resolver con la librería ODEINT el sistema de dos ecuaciones que regula la altura de líquido en dos tanques conectados como indica la figura.



El tanque superior es alimentado con un caudal $Q_1(t)$ que descarga un caudal variable en el tiempo, de tipo onda cuadrada de período **T**

```
tau = t-floor(t/T)*T;
Q1 = Q10*(tau<T/2.0),
```

es decir durante **T/2** descarga un caudal constante **Q10** y durante la otra mitad del período no descarga. El tanque descarga a la red con un caudal **Q2** que es proporcional a la altura del líquido **h1**

$$Q_2 = C_2 \rho h_1 g$$

Para que no rebalse, cuando el líquido llega a una altura **H** existe una segunda descarga a un tanque inferior.

$$Q_3 = \begin{cases} 0; & \text{si } h_1 \leq H; \\ C_3(h - H); & \text{si } h_1 > H; \end{cases}$$

El tanque inferior descarga con un caudal proporcional a la altura

$$Q_4 = C_4 \rho h_2 g \tag{1}$$

Para cada tanque la ecuación de evolución de su altura es

$$A\dot{h} = \sum Q_j$$

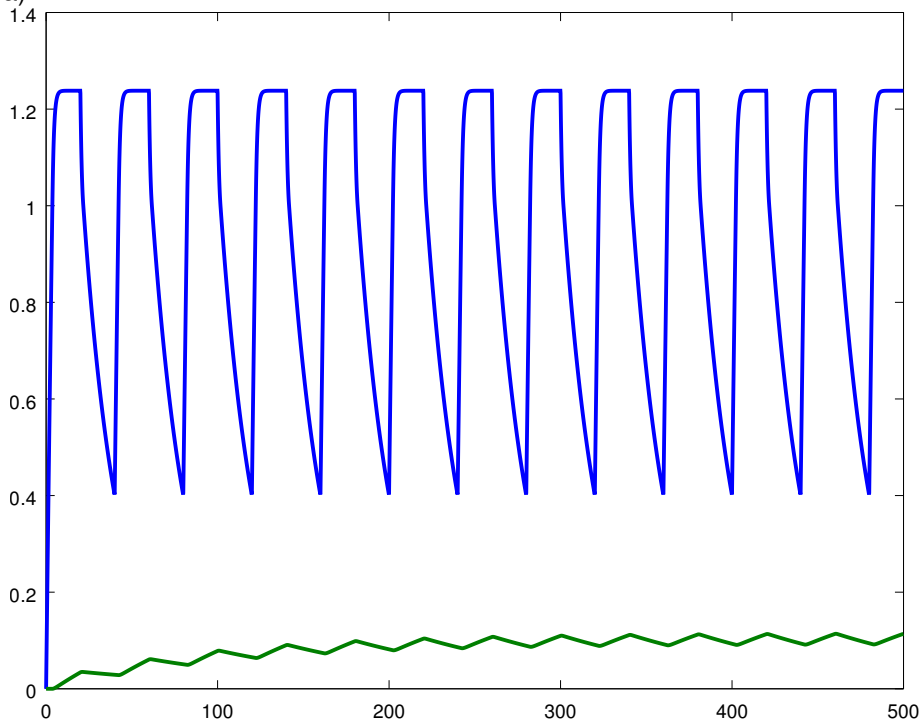
donde **A** es el área del tanque y $\sum Q_j$ la sumatoria (con signo!) de los caudales que ingresan/egresan al tanque.

Consigna: Hallar la altura promedio del tanque inferior luego del transitorio inicial, para los siguientes parámetros:

```
A1 = 1;
C2 = 0.5e-5;
rho = 1000;
g = 10;
C3 = 1;
```

```
H = 1;
Q10 = 0.3;
T = 40;
A2 = 100;
C4 = 1e-4;
```

Ayuda: El ejercicio consiste en escribir la función o clase correspondiente al sistema. El estado del sistema son las dos alturas h_1, h_2. Resolver para un tiempo prudencial y encontrar el promedio de h_2 (en forma aproximada).



Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Hay una función de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

  ```
  ev.eval1(f,vrbs);
  ```

  `ev.eval1(f,vrbs);` toma una serie de casos de prueba de entrada, le aplica la función del usuario **`f`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento

vrbs) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

mstorti@galileo/texstuff-1.2.9-121-g0aa0b78/Wed Aug 12 12:29:05 2015 -0300