Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Examen de Teoría – Soluciones

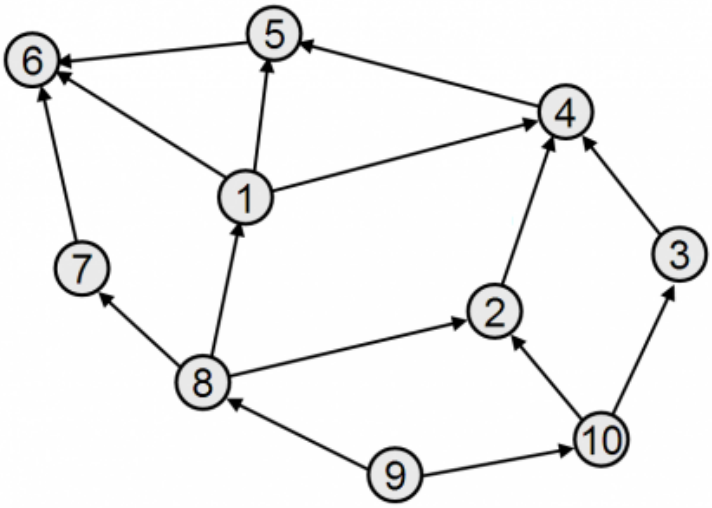
1 de septiembre de 2022

Fundamentos de Informática
Grado en Tecnologías Industriales
Duración: 2h 45m

Ejercicio 1 **[3.0 puntos]**

Un **grafo** es la forma matemática de representar un mapa. Un grafo está formado por una serie de vértices (numerados a partir de 1) situados en una posición del plano, que pueden estar conectados a otros vértices. Las conexiones tienen un sentido determinado, no son bidireccionales: puede ser posible ir de un vértice a otro, pero no a la inversa.

Una forma sencilla de representar un grafo consiste en almacenar el conjunto de sus vértices, guardando para cada uno de ellos su posición y los vértices a los que se puede ir desde él (sus *vecinos*). El número de vecinos de un vértice no tiene porqué ser siempre el mismo, incluso puede haber vértices sin vecinos (se puede llegar a ellos, pero no salir de ellos).



(a) Grafo

```
1   x : ...    y : ...
    vecinos : [4,5,6]
2   x : ...    y : ...
    vecinos : [4]
3   x : ...    y : ...
    vecinos : [4]
...
6   x : ...    y : ...
    vecinos : []
...
8   x : ...    y : ...
    vecinos : [1,2,7]
9   x : ...    y : ...
    vecinos : [8,10]
10  x : ...    y : ...
    vecinos : [2,3]
```

(b) Representación

Una operación típica en un grafo es buscar caminos para ir de un sitio a otro (piensa en Google Maps...). Un camino se representa mediante la secuencia de los vértices que recorremos para ir de un vértice a otro. Por ejemplo, en el grafo de la figura, para ir del vértice 8 al 5 podríamos seguir los caminos `[8,1,5]`, `[8,1,4,5]` o `[8,2,4,5]`.

A partir de esta información, **define** en Pascal los siguientes tipos de datos:

- Un tipo **`tpVertice`**, que almacene la información de un vértice del grafo: sus coordenadas x e y (números reales), y el conjunto de sus vértices vecinos (una secuencia de enteros de longitud variable, con un máximo de 128).
- Un tipo **`tpGrafo`**, que almacene el conjunto de vértices del grafo (de tipo tpVertice), de tamaño variable, con un máximo de 128 vértices.

- Un tipo **`tpCamino`**, que almacene un camino a lo largo del grafo. El camino estará definido mediante los **índices** de los vértices que recorre, y puede ser de tamaño variable, con un máximo de 128.

A partir de esos tipos de dato, se pide **implementar** en lenguaje Pascal los siguientes subprogramas:

- Una función

  ```
  function esVecino(??? gr: tpGrafo; org: integer; dst: integer): boolean;
  ```

  que a partir del grafo `'gr'` (que contiene la información de un grafo), un vértice de partida (`'org'`) y uno de llegada (`'dst'`), devuelva un booleano que indique si se puede ir de forma *directa* de uno a otro (es decir, si `'dst'` es uno de los vecinos de `'org'`).

- Una función

  ```
  function esCaminoValido(??? gr: tpGrafo; ??? cm: tpCamino): boolean;
  ```

  que a partir del grafo `'gr'` (que contiene la información de un grafo), diga si el camino contenido en `'cm'` es un camino válido sobre el grafo, es decir, se puede ir pasando de un vértice a otro del camino desde el primero hasta el último.

- Una función

  ```
  function longitudCamino(??? gr: tpGrafo; ??? cm: tpCamino): real;
  ```

  que a partir del grafo `'gr'` (que contiene la información de un grafo), y un camino contenido en `'cm'`, devuelva la longitud total del camino, calculada como la suma de las distancias entre los vértices por los que va pasando.

Nota: No hay que implementar un programa completo, sólo los tipos de dato y subprogramas que se piden.

Propuesta de solución:

```
program grafos;

const
  MAXV  = 128;

type
  tpVertice = record
    x,y  : real;
    nvec : integer;
    vec  : array[1..MAXV] of integer;
  end;
  tpGrafo = record
    nver : integer;
    ver  : array[1..MAXV] of tpVertice;
  end;
  tpCamino = record
    nver : integer;
    ver  : array[1..MAXV] of integer;
  end;

function esVecino(const gr: tpGrafo; iv: integer; ic: integer): boolean;
var
  i : integer;
begin
  esVecino := false;
  for i:=1 to gr.ver[iv].nvec do
  begin
    if gr.ver[iv].vec[i] = ic then esVecino := true;
  end;
end;

function esCaminoValido(const gr: tpGrafo; const cm: tpCamino): boolean;
var
  i : integer;
begin
  esCaminoValido := true;
  for i:=2 to cm.nver do
  begin
    if not esVecino(gr,cm.ver[i-1],cm.ver[i]) then esCaminoValido := false;
  end;
end;

function distancia(const gr: tpGrafo; i: integer; j: integer): real;
var
  dx,dy : real;
begin
  dx := gr.ver[i].x - gr.ver[j].x;
  dy := gr.ver[i].y - gr.ver[j].y;
  distancia := sqrt(sqr(dx)+sqr(dy));
end;

function longitudCamino(const gr: tpGrafo; const cm: tpCamino): real;
var
  i : integer;
begin
  longitudCamino := 0;
  for i:=2 to cm.nver do
  begin
    longitudCamino := longitudCamino + distancia(gr,cm.ver[i-1],cm.ver[i]);
  end;
end;

begin
end.
```

Ejercicio 2 **[3.5 puntos]**

Un teléfono móvil va registrando constantemente (una vez por segundo) su posición para cada instante temporal, así como si su pantalla está encendida o apagada (esto es, si el teléfono está en uso o no). Dicha información se guarda en un **fichero de texto** (uno por día), de nombre `'log.txt'`. El formato del fichero es tal que cada línea contiene la información de un instante temporal del día, con los siguientes datos separados por espacios en blanco: un número entero que indica los segundos transcurridos desde el comienzo del día, dos números reales que indican la posición en coordenadas GPS (latitud y longitud, en grados) del teléfono en dicho instante temporal, y una cadena de caracteres indicando si la pantalla está encendida ('on') o apagada ('off'). La Figura 2(a) muestra un fichero de ejemplo. Se desea tener un programa que procese la información de un día contenida en `'log.txt'`, y obtenga a partir de ella una serie de datos.

Para ello, **se pide**, usando el lenguaje Pascal:

(a) Definir un tipo de dato adecuado para representar una posición GPS dada por sus coordenadas de latitud y longitud (**`tpPos`**), y un tipo de dato adecuado para representar una marca temporal dada por sus hora, minutos y segundos (**`tpMarcaTemporal`**).

(b) Implementar una función que, dada una cantidad de tiempo en segundos, la convierta en las horas, minutos y segundos correspondientes:

```
function segundosAMarcaTemporal(seg: integer): tpMarcaTemporal;
```

(c) Implementar una función que indique si el usuario (su teléfono) está en un determinado lugar, esto es, si las coordenadas del teléfono y las de ese lugar coinciden. Se considerará que dos posiciones coinciden si la distancia entre ellas es inferior al margen de error del GPS, valor dado por la constante ERRORPOS.

```
function enLugar(posLugar, posUsuario: tpPos): ???;
```

Nota: Sean dos posiciones $p1$ y $p2$ dadas por longitud y latitud en grados, $p1 = (\theta_1, \phi_1)$ y $p2 = (\theta_2, \phi_2)$, se puede aproximar la distancia d entre ellas como $d(p1, p2) = \sqrt{(\theta_1 - \theta_2)^2 + (\phi_1 - \phi_2)^2}$.

(d) Implementar un programa principal que, haciendo uso de las funciones y tipos de dato anteriores, y dado un fichero de entrada `'log.txt'`, haga lo siguiente:

- Pida al usuario y lea de teclado las coordenadas de su casa y de su trabajo.
- Muestre por pantalla el tiempo total que el usuario ha pasado en casa, y en el trabajo, ese día.
- Escriba en un fichero de texto de salida, de nombre 'uso-telefono.txt', cada instante temporal en que se ha encendido y apagado la pantalla del teléfono.

La Figura 2(b y c) muestra un ejemplo de cómo debe ser el formato de la información de salida.

Observaciones:

1. Se puede asumir que el usuario siempre comienza el día en casa y con la pantalla del teléfono en 'off'. La primera línea corresponde al segundo cero.
2. En el fichero de entrada hay una línea por segundo, se puede asumir que dicho fichero no contiene fallos.
3. Se pueden comparar dos datos de tipo String con los operadores de igualdad (=, <>).
4. *Importante:* No se pueden utilizar datos de tipo `array` (vectores de una o más dimensiones).

```
00000 54.52 40.21 off
00001 54.51 40.22 on
00002 54.50 40.22 on
00003 54.50 40.22 on
00004 54.50 40.22 off
00005 51.00 45.00 off
00006 51.00 45.00 on
00007 51.00 45.00 on
00008 60.00 70.00 off
00009 60.00 70.00 off
00010 60.01 70.02 off
00011 60.01 70.01 on
00012 52.10 53.50 on
00013 52.10 53.50 off
00014 54.50 40.22 on
00015 54.51 40.22 off
00016 54.51 40.22 off
00017 54.52 40.21 off
00018 75.10 30.00 on
```

(a) Ejemplo de fichero de entrada (‘log.txt’). Se muestra un fragmento del fichero.

```
Introduce coordenadas de casa: 54.50 40.21
Introduce coordenadas del trabajo: 60.00 70.00
Tiempo total en casa: 7 segundos
Tiempo total en el trabajo: 3 segundos
```

(b) Ejemplo de interacción. En negrita figura lo que introduce el usuario. Los resultados mostrados corresponden al fragmento de fichero de entrada ‘log.txt’ mostrado en (a).

```
Movil apagado a las   0h0m0s
Movil encendido a las 0h0m1s
Movil apagado a las   0h0m4s
Movil encendido a las 0h0m6s
Movil apagado a las   0h0m8s
Movil encendido a las 0h0m11s
Movil apagado a las   0h0m13s
Movil encendido a las 0h0m14s
Movil apagado a las   0h0m15s
Movil encendido a las 0h0m18s
```

(c) Ejemplo de fichero de salida (‘uso-telefono.txt’). Se muestra un fragmento del fichero de salida, correspondiente al fragmento de fichero de entrada ‘log.txt’ mostrado en (a).

Figura 2: Ejemplo de fichero de entrada (a), interaccion con el usuario (b) y fichero de salida correspondiente (c). Los datos incluidos no son realistas, sino que tienen un propósito meramente ilustrativo, y corresponden sólo a las primeras 19 líneas del fichero de entrada. El usuario está en casa entre el segundo 0 y el 4, y entre el segundo 14 y 17 (7 segundos en total); el usuario está en el trabajo entre el segundo 8 y el 11 (3 segundos en total).

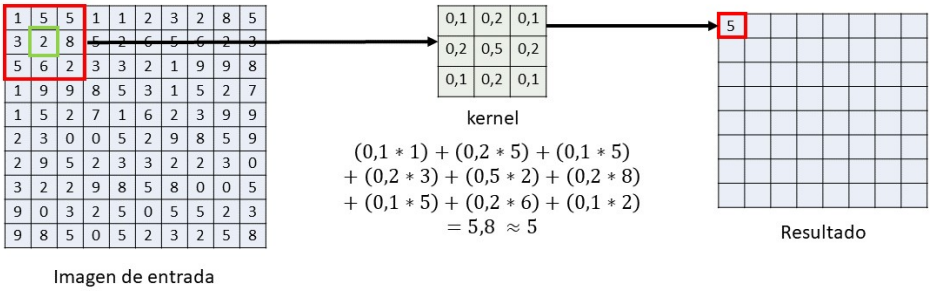
Propuesta de solución:

Ejercicio 3 **[3.5 puntos]**

Una imagen digital no es más que una matriz bidimensional de elementos llamados píxeles, cuyos valores numéricos indican el color de la correspondiente región de la foto. En el caso de una imagen en escala de grises, estos valores numéricos son enteros que van desde 0 (que indica negro) hasta 255 (que indica blanco). Los valores intermedios representan, por tanto, distintas gradaciones de gris.

Existen numerosas técnicas de tratamiento digital que se basan en alterar los datos de una imagen. Una de esas herramientas es el filtro gaussiano, muy usado en los editores de imágenes. Este filtro puede aplicarse a una imagen original para realizar un difuminado de la misma, obteniendo una imagen filtrada.

La imagen filtrada se obtiene de la siguiente forma: cada píxel de la imagen filtrada es el resultado de realizar una combinación lineal (una suma con pesos) de los valores de los vecinos[1] del píxel correspondiente en la imagen original. Los pesos de esa combinación lineal vienen dados por el *kernel*, una matriz de tamaño reducido que sirve para dar diferente importancia a cada uno los píxeles vecinos. Por ejemplo, para el primer píxel de la imagen filtrada (resultado):



Esta combinación lineal se aplica a cada píxel de la imagen original (exceptuando los de los bordes, ya que no tienen 8 vecinos), para obtener los valores de píxel de la imagen filtrada.
Dada la siguiente definición de tipos de dato:

```
const
   MAX = ...
type
   tpImagen = record
      dat: array[1..MAX, 1..MAX] of integer;
      nFil, nCol: integer;
   end;
   tpKernel = array[1..3, 1..3] of real;
```

se pide implementar en Pascal el siguiente procedimiento, que calculará la imagen filtrada a partir de la imagen original y el kernel:

```
procedure filtrar(const original: tpImagen; kernel: tpKernel; var filtrada: tpImagen);
```

Observaciones:

1. La imagen filtrada tendrá dos filas y dos columnas menos que la imagen original (como en la figura). Esto es porque no se puede aplicar el kernel a pixeles que no tengan suficientes vecinos.
2. Pese a que los pesos del kernel son valores de tipo `real`, los valores de la imagen resultante tienen que ser de tipo `entero`. Para convertir un dato de tipo real en un entero se puede usar la función predefinida en Pascal `trunc`, que toma como parámetro un número real y devuelve su parte entera:

   ```
   function trunc(x: real): integer;   { devuelve la parte entera del real x }
   ```

   Por ejemplo:

   ```
   trunc(5.8) = 5 // donde 5 es un dato de tipo integer
   ```

[1] Los vecinos de un píxel son los píxeles de arriba, abajo, izquierda, derecha, y los ubicados contiguamente en las diagonales.

Propuesta de solución:

```
program Convolucion;
const
    MAX = 1024;
type
    tpImagen= record
        datos: array[1..MAX,1..MAX] of integer;
        ancho: integer;
        alto: integer;
    end;
    tpKernel= array[1..3,1..3] of real;

{ Opcion 1 }

function aplicar_kernel(const imagen:tpImagen;const kernel:tpKernel):tpImagen;
var
   salida: tpImagen;
   i,j : integer;
   valor: real;
   x,y : integer;
begin
   salida.ancho:= imagen.ancho - 2;
   salida.alto:= imagen.alto - 2;
   for i:= 2 to imagen.alto-1 do
    for j:=2 to imagen.ancho-1 do
        begin
        valor:=0.0;
        for x:= 1 to 3 do
            for y:= 1 to 3 do
            begin
                valor:= valor + kernel.datos[x,y]*imagen.datos[(i-1+x),(j-1+y)];
            end;
        end;
    salida[x,y]:=trunc(valor);
end;

{ Opcion 2 }

function aplicar_kernel(const imagen:tpImagen;const kernel:tpKernel):tpImagen;
var
   salida: tpImagen;
   i,j : integer;
   valor: real;
   x,y : integer;
begin
   salida.ancho:= imagen.ancho - 2;
   salida.alto:= imagen.alto - 2;
   Offset := 1;
   for i:=Offset+1 to imagen.alto-Offset do
    for j:=Offset+1 to imagen.ancho-Offset do
        begin
        valor:=0.0;
        for x:= -Offset to Offset do
            for y:= -Offset to Offset do
            begin
                valor:= valor + kernel.datos[Offset+x,Offset+y]*imagen.datos[(i+x),(j+y)];
            end;
        end;
    salida[x,y]:=trunc(valor);
end;
```