Informática
Año 2014/2015
Facultad de CC.
Matemáticas

Examen

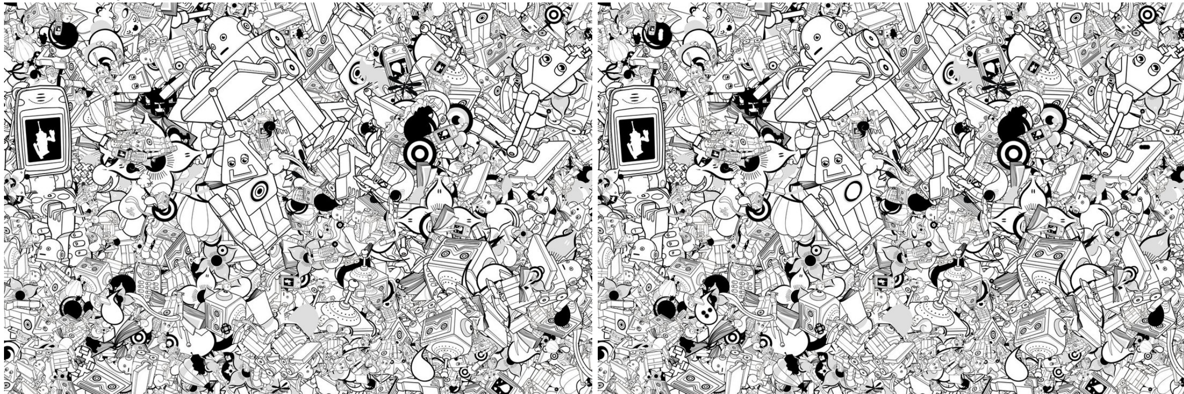
Septiembre

14 de Septiembre 2015

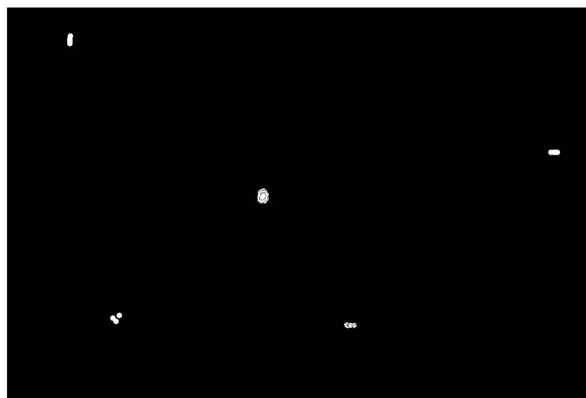
1. Diferencias de imágenes [3.5 puntos]

Escribe una función que reciba dos imágenes y devuelva una nueva imagen que muestre las diferencias entre las imágenes recibidas.

Por ejemplo, la ejecución de la función con las dos imágenes de entrada siguientes



podría devolver como resultado una imagen en la que se mostraran las diferencias así:



La función tiene que comprobar que las imágenes son comparables, es decir, al menos que tienen las mismas dimensiones. Se puede suponer que las imágenes son en blanco y negro.

Ayuda

- Las imágenes se manipulan importando la clase `Image` del módulo `PIL`
- Abrir imágenes con `open`:

  `img = Image.open(fichero_imagen,'r')`
- Tamaño de una imagen `img`:

  `ancho, alto = img.size`

donde `ancho` es el número de píxeles horizontales que tiene la imagen y `alto` es el número de píxeles verticales.

- Crear imagen nueva:

  `resultado = Image.new('L', (w, h), c)`

  donde `w` es el número de píxeles horizontales que tendrá la imagen, `h` es el número de píxeles verticales y `c` el color por defecto. El modo `'L'` es el de las imágenes en blanco y negro.

- Leer el valor de un píxel en las coordenadas `(x,y)` de la imagen `img`:

  `valor = img.getpixel ((x,y))`

- Escribir el valor `v` en un píxel en las coordenadas `(x,y)` de la imagen `img`:

  `img.putpixel ((x,y), v)`

2. Función Cremallera [1.5]

Dadas dos listas *con la misma longitud* $Lista_1$ y $Lista_2$, escribe una función **recursiva** `cremallera` que devuelva una lista de parejas, en el que la pareja $i-$ésima resulta de colocar como primer elemento de la pareja el elemento $i-$ésimo de $Lista_1$ y como segundo el elemento i-ésimo de $Lista_2$. Por ejemplo:

```
>>> cremallera([23345,43,45,145],['a','b','f','h'])
[(23345, 'a'), (43, 'b'), (45, 'f'), (145, 'h')]

>>> cremallera([23345],['a'])
[(23345, 'a')]
```

3. Figuras geométricas: Semiplanos [5 puntos]

Dadas las definiciones de las clases `Point, Vector, Segment` y `Line` (ver Ayuda):

1. **[3 puntos]** Escribe la clase `Semiplane` para representar semiplanos. Para determinar un semiplano basta con considerar una recta y un punto en el semiplano.

   - **[1 punto]** El constructor de la clase debe comprobar la corrección de los datos de entrada, de manera que el semiplano quede determinado de manera precisa.
   - **[2 puntos]** Implementa el método `intersects(self, other)` (que indica si `self` y `other` tienen intersección no vacía). El objeto `other` puede pertenecer a las clases `Point` o `Segment`.

2. **[2 puntos]** Se debe implementar un método `contains(self, other)` que indica si `other`, que puede ser un punto `Point` o un segmento `Segment`, está totalmente contenido en el semiplano.

Ayuda

Listing 1: Código de las clases `Vector`, `Point`, `Line`, `Segment`

```
from sympy import *
def is_number_type(n):
    return isinstance(n,int) or isinstance(n,float) or isinstance(n,Expr)

class Vector(object):
    def __init__(self,vx,vy):
        if is_number_type(vx) and is_number_type(vy):
            self.x=vx
            self.y=vy
        else:
            raise Exception('Bad data for Vector construction')

    def __repr__(self):
        return 'Vector('+str(self.x)+','+str(self.y)+')'

    def dot(self,otro):
        return self.x*otro.x+self.y*otro.y

    def __add__(self,other):
        return Vector(self.x+other.x,self.y+other.y)

    def norm(self):
        return sqrt(self.dot(self))

    def unit(self):
        l=self.norm()
        return Vector(self.x/l,self.y/l)

    def ortogonal(self):
        return Vector(-self.y,self.x)

    def is_parallel(self,other):
        return self.ortogonal().dot(other)==0

class Point(object):
    def __init__(self,px,py):
        if is_number_type(px) and is_number_type(py):
            self.x=px
            self.y=py
        else:
            raise Exception('Bad data for Point construction')

    def __repr__(self):
        return 'Point('+str(self.x)+','+str(self.y)+')'

    def intersects(self,other):
        if isinstance(other,Point):
            return self.x==other.x and self.y==other.y
        else:
            return other.intersects(self)

    def distance(self,other):
        if isinstance(other,Point):
            return self.vector_to(other).norm()
        else:
            return other.distance(self)

    def __add__(self,v):
        return Point(self.x+v.x,self.y+v.y)

    def vector_to(self,other):
        return Vector(other.x-self.x,other.y-self.y)
```

```
class Line(object):
    def __init__(self,point,point1):
        if isinstance(point,Point) and isinstance(point1,Point)\
          and point.distance(point1)>0:
            self.p=point
            self.p1=point1
            self.v=point.vector_to(point1)
            self.normal=self.v.ortogonal().unit()
        else:
            raise Exception('Bad data for Line construction')

    def __repr__(self):
        return 'Line('+str(self.p)+','+str(self.p1)+')'

    def intersects(self,other):
        if isinstance(other,Point):
            v=self.p.vector_to(other)
            return v.is_parallel(self.v)
        elif isinstance(other,Line):
            if self.v.is_parallel(other.v):
                return self.intersects(other.p)
            else:
                return True
        else:
            return other.intersects(self)

    def distance(self,other):
        if isinstance(other,Point):
            v_orto=self.v.ortogonal().unit()
            return abs(v_orto.dot(self.p.vector_to(other)))
        elif isinstance(other,Line):
            if self.v.is_parallel(other.v):
                return self.distance(other.p)
            else:
                return 0
        else:
            return other.distance(self)
```

```
class Segment(object):
    def __init__(self,p1,p2):
        if isinstance(p1,Point) and isinstance(p2,Point) and p1.distance(p2)>0:
            self.end1=p1
            self.end2=p2
            self.v=p1.vector_to(p2)
            self.length=self.v.norm()
            self.unit=self.v.unit()
            self.l=Line(self.end1,self.end2)
        else:
            raise Exception('Bad data for Segment construction')

    def __repr__(self):
        return 'Segment('+str(self.end1)+','+str(self.end2)+')'

    def support(self):
        return self.l

    def intersects(self,other):
        if isinstance(other,Point):
            on_line=self.l.intersects(other)
            proy=self.unit.dot(self.end1.vector_to(other))
            return on_line and (0<=proy and proy<=self.v.norm())
        elif isinstance(other,Line):
            v_ort=other.v.ortogonal()
            proy1=other.p.vector_to(self.end1).dot(v_ort)
            proy2=other.p.vector_to(self.end2).dot(v_ort)
            return proy1*proy2<=0
        elif isinstance(other,Segment):
            if self.v.is_parallel(other.v):
                return self.intersects(other.end1) or self.intersects(other.end2) or\
                       other.intersects(self.end1)
            else:
                return other.intersects(self.l) and self.intersects(other.l)
        else:
            return other.intersects(self)

    def distance(self,other):
        if isinstance(other,Point):
            proyection=self.unit.dot(self.end1.vector_to(other))
            if proyection>self.length:
                return self.end2.distance(other)
            elif proyection>=0:
                return Line(self.end1,self.end2).distance(other)
            else:
                return self.end1.distance(other)
        elif isinstance(other,Line):
            if self.intersects(other):
                return 0
            else:
                return min(self.end1.distance(other),self.end2.distance(other))
        elif isinstance(other,Segment):
            if self.intersects(other):
                return 0
            else:
                return min(self.end1.distance(other),self.end2.distance(other),\
                           other.end1.distance(self),other.end2.distance(self))
        else:
            return other.distance(self)
```