Problem 1

1. $h(S_1, S_2)$:

$D_{min}(A \to S_2) = \sqrt{2}$

$D_{min}(B \to S_2) = \sqrt{2}$

$D_{min}(C \to S_2) = 2\sqrt{2}$

$h(S_1, S_2) = Max(\sqrt{2}, \sqrt{2}, 2\sqrt{2}) = 2\sqrt{2}$

$h(S_2, S_1)$:

$D_{min}(D \to S_1) = \sqrt{2}$

$D_{min}(E \to S_1) = \sqrt{2}$

$D_{min}(F \to S_1) = \sqrt{5}$

$D_{min}(G \to S_1) = \sqrt{13}$

$h(S_2, S_1) = max(\sqrt{2}, \sqrt{2}, \sqrt{5}, \sqrt{13}) = \sqrt{13}$

$H(S_1, S_2) = max(h(S_1, S_2), h(S_2, S_1)) = \sqrt{13}$

2. As to $h(S_1, S_2)$

For each point in S_1, make a vertical line to the rectangle, the distance is $D_{min}(P - \text{Rectangle})$

Therefore, $h(\triangle \to \square)$

$=$ the distance between $C(0, -3)$ to the rectangle

$= 2$

As to $h(\square \to \triangle)$, Obviously, $G(-3, -1)$

AC: $y = -3x - 3$

Assume, GM is vertical to AC.

GM: $y = \frac{1}{3}x$

$-3x - 3 = \frac{1}{3}x$, $\frac{10}{3}x = -3$, $x = -\frac{9}{10}$, $y = -\frac{3}{10}$

$M(-\frac{9}{10}, -\frac{3}{10})$

$|GM| = \sqrt{(-3+\frac{9}{10})^2 + (-1+\frac{3}{10})^2} = \sqrt{\frac{441}{100} + \frac{49}{100}} = \sqrt{\frac{490}{100}} = \sqrt{\frac{49}{10}}$

$H(\triangle, \square) = max(h(\triangle \to \square), h(\square \to \triangle))$

$= \frac{7\sqrt{10}}{10}$

2. the two peaks ~~$g_i = 1750$, $g_j = 13$.~~

$g_i = 103$ $g_j = 231$

$H(g_i) = 1750$ $H(g_j) = 1300$.

$g_k = 157$ $H(g_k) = 190$.

$$P(i, j, k) = [\min\{1750, 1300\}] / H(g_k)$$

$$= 1300 / 190 \approx 6.84.$$

the peakness is 6.84. threshold g_k is 157

In [9]:

```
from PIL import Image
import numpy as np
import cv2, os
from scipy.io import loadmat
from scipy import signal
import evaluate_boundaries
from scipy import ndimage

N_THRESHOLDS = 99

def detect_edges(imlist, fn):
  images, edges = [], []
  for imname in imlist:
    I = cv2.imread(os.path.join('data', str(imname)+'.jpg'))
    images.append(I)

    I = cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)
    I = I.astype(np.float32)/255.
    mag = fn(I)
    edges.append(mag)
  return images, edges

def evaluate(imlist, all_predictions):
  count_r_overall = np.zeros((N_THRESHOLDS,))
  sum_r_overall = np.zeros((N_THRESHOLDS,))
  count_p_overall = np.zeros((N_THRESHOLDS,))
  sum_p_overall = np.zeros((N_THRESHOLDS,))
  for imname, predictions in zip(imlist, all_predictions):
    gt = loadmat(os.path.join('data', str(imname)+'.mat'))['groundTruth']
    num_gts = gt.shape[1]
    gt = [gt[0,i]['Boundaries'][0,0] for i in range(num_gts)]
    count_r, sum_r, count_p, sum_p, used_thresholds = \
              evaluate_boundaries.evaluate_boundaries_fast(predictions, gt,
                                                             thresholds=N_THRESHOLDS,
                                                             apply_thinning=True)
    count_r_overall += count_r
    sum_r_overall += sum_r
    count_p_overall += count_p
    sum_p_overall += sum_p

  rec_overall, prec_overall, f1_overall = evaluate_boundaries.compute_rec_prec_f1(
        count_r_overall, sum_r_overall, count_p_overall, sum_p_overall)

  return max(f1_overall)

def compute_edges_dxdy(I):
  """Returns the norm of dx and dy as the edge response function."""

  dx = signal.convolve2d(I, np.array([[-1, 0, 1]]), mode='same')
  dy = signal.convolve2d(I, np.array([[-1, 0, 1]]).T, mode='same')
  mag = np.sqrt(dx**2 + dy**2)
  mag = normalize(mag)
  return mag

def normalize(mag):
  mag = mag / 1.5
  mag = mag * 255.
  mag = np.clip(mag, 0, 255)
  mag = mag.astype(np.uint8)
```

```
  return mag


imlist = [12084, 24077, 38092]
fn = compute_edges_dxdy
images, edges = detect_edges(imlist, fn)
display(Image.fromarray(np.hstack(images)))
display(Image.fromarray(np.hstack(edges)))
f1 = evaluate(imlist, edges)
print('Overall F1 score:', f1)
```

```
Overall F1 score: 0.4163603293750655
```

1. **[2 pts] Warm-up.** As you visualize the produced edges, you will notice artifacts at image boundaries. Modify how the convolution is being done to minimize these artifacts.

In [10]:

```
def compute_edges_dxdy_warmup(I):
  """Hint: Look at arguments for scipy.signal.convolve2d"""
  # ADD CODE HERE

  dx = signal.convolve2d(I, np.array([[-1, 0, 1]]), mode='same', boundary="symm")
  dy = signal.convolve2d(I, np.array([[-1, 0, 1]]).T, mode='same', boundary="symm")
  mag = np.sqrt(dx**2 + dy**2)
  mag = normalize(mag)
  return mag


imlist = [12084, 24077, 38092]
fn = compute_edges_dxdy_warmup
images, edges = detect_edges(imlist, fn)
display(Image.fromarray(np.hstack(images)))
display(Image.fromarray(np.hstack(edges)))
f1 = evaluate(imlist, edges)
print('Overall F1 score:', f1)
```

```
Overall F1 score: 0.4175307435125027
```

2. **[5 pts] Smoothing.** Next, notice that we are using [−1, 0, 1] filters for computing the gradients, and they are susceptible to noise. Use derivative of Gaussian filters to obtain more robust estimates of the gradient. Experiment with different sigma for this Gaussian filtering and pick the one that works the best.

In [11]:

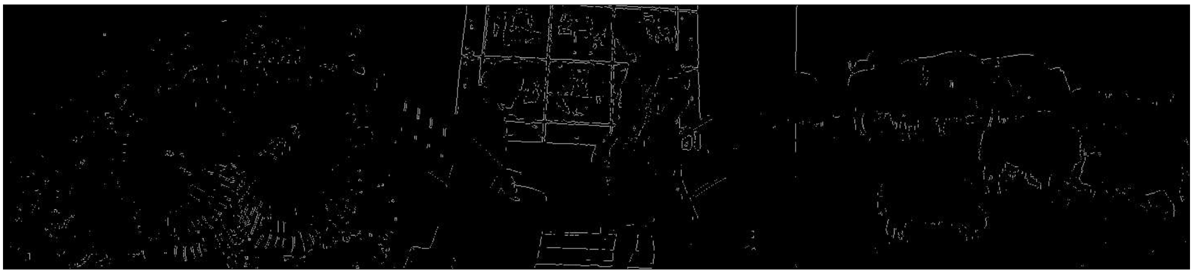
```
def compute_edges_dxdy_smoothing(I):
  """ Copy over your response from part 3.1 and alter it
  to include this answer. See cv2.GaussianBlur"""
  # ADD CODE HERE

  I = cv2.GaussianBlur(I, (3,3), 0.51)

  dx = signal.convolve2d(I, np.array([[-1, 0, 1]]), mode='same', boundary="symm")
  dy = signal.convolve2d(I, np.array([[-1, 0, 1]]).T, mode='same', boundary="symm")
  mag = np.sqrt(dx**2 + dy**2)
  mag = normalize(mag)
  return mag


imlist = [12084, 24077, 38092]
fn = compute_edges_dxdy_smoothing
images, edges = detect_edges(imlist, fn)
display(Image.fromarray(np.hstack(images)))
display(Image.fromarray(np.hstack(edges)))
f1 = evaluate(imlist, edges)
print('Overall F1 score:', f1)
```

```
Overall F1 score: 0.4195793230186947
```

3. **[8 pts] Non-maximum Suppression.** The current code does not produce thin edges. Implement non-maximum suppression, where we look at the gradient magnitude at the two neighbours in the direction perpendicular to the edge. We suppress the output at the current pixel if the output at the current pixel is not more than at the neighbors. You will have to compute the orientation of the contour (using the X and Y gradients), and then lookup values at the neighbouring pixels.

In [56]:

```
def compute_edges_dxdy_nonmax(I):
  """ Copy over your response from part 3.2 and alter it
  to include this response"""
  # ADD CODE  HERE

  I = cv2.GaussianBlur(I, (3,3), 0.5)

  dx = signal.convolve2d(I, np.array([[-1, 0, 1]]), mode='same', boundary="symm")
  dy = signal.convolve2d(I, np.array([[-1, 0, 1]]).T, mode='same', boundary="symm")

  Grad = np.hypot(dx, dy)
  Grad = Grad / Grad.max() * 2.4
  theta = np.arctan2(dy, dx)

  mag = NMS(Grad, theta)
  mag = normalize(mag)

  return mag

def NMS(grad, theta):
    X, Y = grad.shape
    output = np.zeros((X,Y), dtype=np.int32)
    theta = theta * 180. / np.pi
    theta[theta < 0] += 180
    for i in range(1,X-1):
        for j in range(1,Y-1):
            try:
                m = n = 255
                if (0 <= theta[i,j] < 22.5):
                    m = grad[i, j+1]
                    n = grad[i, j-1]
                elif (157.5 <= theta[i,j] <= 180):
                    m = grad[i, j+1]
                    n = grad[i, j-1]
                elif (22.5 <= theta[i,j] < 67.5):
                    m = grad[i+1, j-1]
                    n = grad[i-1, j+1]
                elif (67.5 <= theta[i,j] < 112.5):
                    m = grad[i+1, j]
                    n = grad[i-1, j]
                elif (112.5 <= theta[i,j] < 157.5):
                    m = grad[i-1, j-1]
                    n = grad[i+1, j+1]
                if (grad[i,j] >= m) and (grad[i,j] >= n):
                    output[i,j] = grad[i,j]
                else:
                    output[i,j] = 0

            except IndexError as e:
                    pass
    return output

imlist = [12084, 24077, 38092]
fn = compute_edges_dxdy_nonmax
images, edges = detect_edges(imlist, fn)
display(Image.fromarray(np.hstack(images)))
display(Image.fromarray(np.hstack(edges)))
f1 = evaluate(imlist, edges)
print('Overall F1 score:', f1)
```



Overall F1 score: 0.5976593536089015

In [ ]: