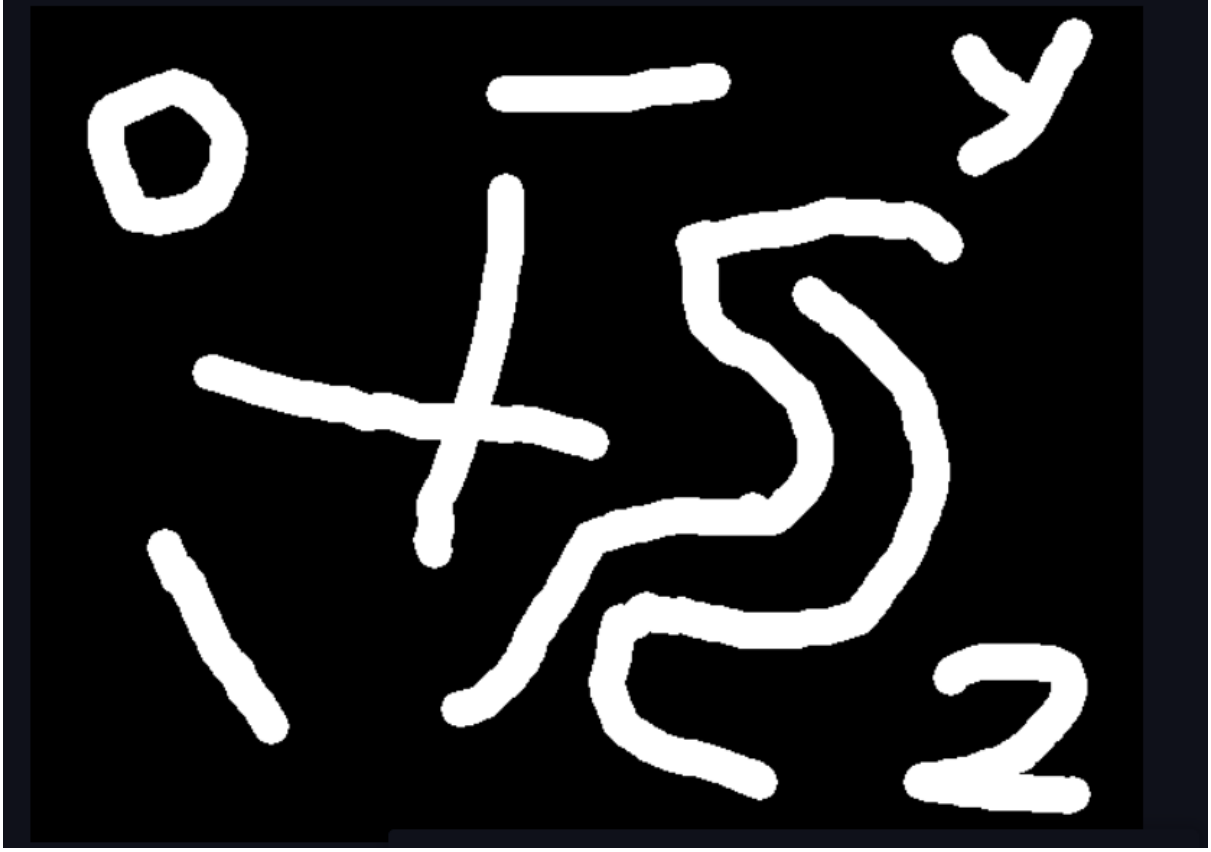
سوال ۱

در ابتدا با استفاده از opencv تصویر را می‌خوانیم و آن را نمایش می‌دهیم.

```python
img1 = cv2.imread('q1.png', cv2.IMREAD_GRAYSCALE) # Read image in grayscale
cv2_imshow(img1) # Showing image using opencv. It was not showing correct image u
```

Python



سپس در گام بعدی، جهت یافتن اجزای متصل موجود در تصویر از قطعه کد زیر استفاده می‌کنیم:

```python
count_labels, labels = cv2.connectedComponents(img1) # We get connected component
rows, cols = img1.shape
```

Python

با استفاده از این کد، یک ماتریس هم‌اندازه با تصویر اولیه را دریافت می‌کنیم. به طوری که به ازای هر پیکسل، یک عدد بر روی ماتریس قرار گرفته است، که هر عدد نشان‌دهندهٔ شمارهٔ گروه پیکسلی است. همچنین تعداد ردیف‌ها و ستون‌ها را نیز می‌یابیم چرا که بعداً به کار می‌آیند.

برای آنکه بخواهیم تصویر را به همراه لیبل‌هایش نمایش دهیم در ابتدا تصویر را به صورت BGR در می‌آوریم.

```
# We need to convert image to bgr to show each label on it.
img1_color = cv2.cvtColor(img1, cv2.COLOR_GRAY2RGB)
img1_color = cv2.cvtColor(img1_color, cv2.COLOR_RGB2BGR)
```

سپس یک تابع ایجاد می‌کنیم که یک رنگ تصادفی ایجاد کند.

```
# A function to return colors
def get_random_color(count):
    ret_list = []
    for i in range(count):
        ret_list.append(np.random.randint(0, 256, size=3).tolist())
    return ret_list
```

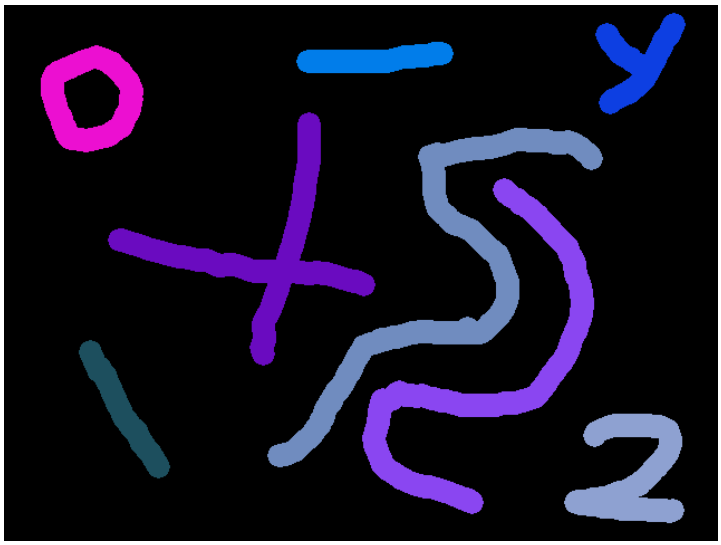
در گام بعدی در ابتدا رنگ‌ها را به اندازهٔ تعداد لیبل‌ها ایجاد می‌کنیم و سپس هر پیکسل را با توجه به ماتریس لیبل‌ها رنگ‌آمیزی می‌کنیم.

```
colors = get_random_color(count_labels)

# replace each pixle based on its label
for i in range(rows):
    for j in range(cols):
        if labels[i, j] != 0:
            img1_color[i, j] = colors[labels[i, j] - 1]

cv2_imshow(img1_color)
```

نتیجهٔ نهایی رنگ‌آمیزی به صورت زیر خواهد شد:



در گام بعدی، تعداد اجزای متصل موجود در شکل را بر روی آن چاپ می‌کنیم. برای این منظور از تابع cv2.putText استفاده می‌کنیم.

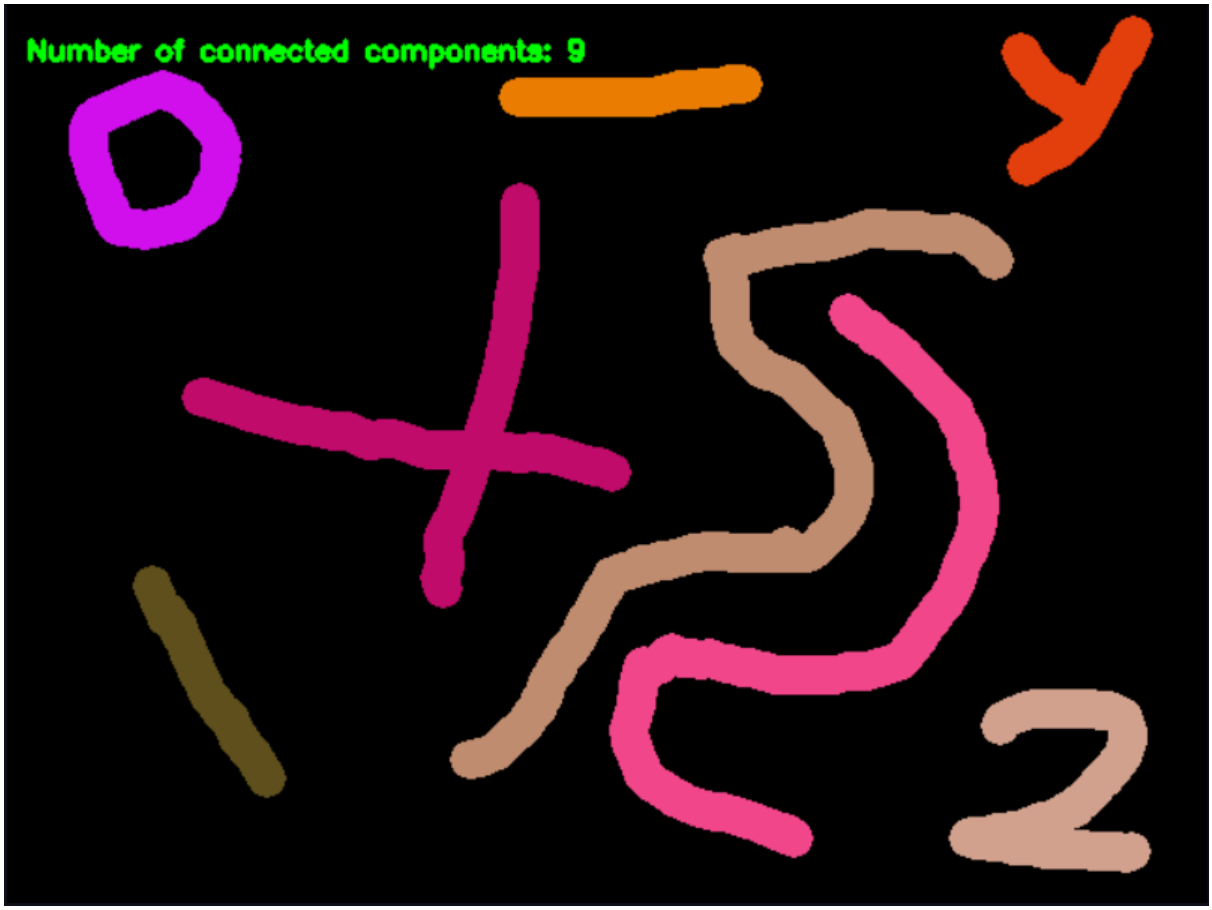
```
# Define the text to draw
text = 'Number of connected components: ' + str(count_labels)

# Set the font properties
font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 0.5
font_color = (0, 255, 0)  # Green color
thickness = 2

# Set the text position
text_x = 10
text_y = 30

# Draw the text on the image
cv2.putText(img1_color, text, (text_x, text_y), font, font_scale, font_color, thickness)
```

نتیجهٔ نهایی به دست آمده به صورت زیر است:



عدد ۹ به دلیل ۸ شکل موجود در تصویر و نیز رنگ سیاه پیش‌زمینه است.

سوال ۲

برای حل این مسئله، پس از مشخص کردن نقطهٔ اولیه، با استفاده از الگوریتم DFS، و نیز مقایسهٔ هر نقطهٔ همسایهٔ یک نقطه با آن، عملیات Region Growing را انجام می‌دهیم. برای مشخص کردن همسایه‌های هر نقطه، از دو حالت ۴تایی و نیز ۸تایی استفاده شده است. همچنین به ازای ۴ آستانه نیز در پایان تست صورت گرفته است و نتیجهٔ نهایی به نمایش در آمده است.

در زیر نتیجه به ازای آستانه‌ها و نیز حالت‌های مختلف در نظر گرفتن همسایه‌ها آورده شده است. تصاویر ستون راست مربوط به حالت ۸تایی و سمت چپ مربوط به حالت ۴تایی هستند.

- آستانهٔ ۱:

- آستانهٔ ۲:

- آستانهٔ ۳.۵:

- آستانهٔ ۵:

همانطور که مشخص است به ازای آستانهٔ ۳.۵ بهترین پاسخ بدست آمده است.

در حالت همسایگی ۸تایی، نسبت به حالت همسایگی ۴تایی، ناحیه‌هایی که به عنوان نقاط مشابه با نقطهٔ اولیه در نظر گرفته شده‌اند بیشتر بوده و ناحیه‌های بزرگتری را به عنوان نقاط مشابه در نظر گرفته است. همین موضوع باعث می‌شود که احتمال نشت کردن الگوریتم بالاتر رفته و نواحی‌ای که شباهتی به حالت اولیه ندارد نیز به عنوان نقاط مشابه شناخته شوند و رشد نواحی در این نقاط بیشتر شود.

در رابطه با آستانه نیز همین موضوع صدق می‌کند؛ به طوری که هر چقدر آستانه بالاتر رود ناحیهٔ در نظر گرفته شده به عنوان نقاط مشابه با نقطهٔ سید، بیشتر می‌شود و در صورت تنظیم نادرست آن احتمال نشت الگوریتم بالاتر می‌رود. کما اینکه با قرار دادن آستانه برابر با مقدار ۵، می‌بینیم که تمام چهرهٔ فرد، چه موهای سر و چه چهرهٔ فرد به عنوان ناحیهٔ انتخاب شده تشخیص داده می‌شوند.

سوال ۳

در ابتدا با استفاده از کد پایتون یک ماتریس دوبعدی از اعداد را تشکیل می‌دهیم. نتیجهٔ بدست آمده به صورت زیر است:

```
np.random.randint(1, 16, size=25).reshape(5, 5)

array([[ 8, 11,  6,  6, 12],
       [ 6, 10, 15,  5, 12],
       [ 8,  5, 13,  5,  6],
       [15,  7,  1, 13,  8],
       [15,  6,  1, 15, 13]])
```

در ادامه ماتریس را برای هر بخش به دو گروه تقسیم می‌کنیم و الگوریتم Otsu را روی آن اعمال می‌کنیم. به طوری که برای هر بخش طبق الگوریتم، یک امتیاز را به صورت زیر محاسبه می‌کنیم:

$$score = w_1\sigma_1^2 + w_2\sigma_2^2$$

تقسیم‌بندی به این صورت است که مقادیر بزرگتر مساوی آستانه در یک گروه و مقادیر کوچکتر از آن در گروه دوم قرار می‌گیرند.

- با استفاده از آستانهٔ ۶:

۸	۱۱	۶	۶	۱۲
۶	۱۰	۱۵	۵	۱۲
۸	۵	۱۳	۵	۶
۱۵	۷	۱	۱۳	۸
۱۵	۶	۱	۱۵	۱۳

داریم:

گروه «بزرگتر مساوی»:

$$w1 = 20$$

$$\mu_1 = \frac{8+11+6+6+12+6+10+15+12+8+13+6+15+7+13+8+15+6+15+13}{20}$$

$$\rightarrow \mu_1 = 10.25$$

حال با استفاده از رابطهٔ زیر واریانس این گروه را محاسبه می‌کنیم.

$$S^2 = \frac{\sum (x_i - \bar{x})^2}{n-1}$$

$$\sigma_1^2 = \frac{(8-10.25)^2 + (11-10.25)^2 + (6-10.25)^2 + \cdots + (13-10.25)^2}{19}$$
$$\rightarrow \sigma_1^2 = 12.4$$

گروه «کوچکتر»:

$$w_2 = 5$$
$$\mu_2 = \frac{5+5+5+1+1}{5} = \frac{17}{5} = 3.4$$
$$\sigma_2^2 = \frac{(5-3.4)^2 + (5-3.4)^2 + (5-3.4)^2 + (1-3.4)^2 + (1-3.4)^2}{4}$$
$$\rightarrow \sigma_2^2 = 4.8$$

حال داریم:

$$Score = 4.8 \times 5 + 12.4 \times 20 = 272$$

- با استفاده از آستانهٔ ۱۰:

۸	۱۱	۶	۶	۱۲
۶	۱۰	۱۵	۵	۱۲
۸	۵	۱۳	۵	۶
۱۵	۷	۱	۱۳	۸
۱۵	۶	۱	۱۵	۱۳

گروه «بزرگتر مساوی»:

$$w_1 = 11$$
$$\mu_1 = \frac{11+12+10+15+12+13+15+15+13+15+13}{11}$$
$$\rightarrow \mu_1 = 13.09$$
$$\sigma_1^2 = \frac{(11-13.09)^2 + (10-13.09)^2 + (12-13.09)^2 + \cdots + (13-13.09)^2}{10}$$
$$\rightarrow \sigma_1^2 = 3.09$$

گروه «کوچکتر»:

$$w_2 = 14$$
$$\mu_2 = \frac{8+6+6+6+5+8+5+5+6+7+1+8+6+1}{14} = \frac{78}{14} = 5.57$$

$$\sigma_2^2 = \frac{(8-5.57)^2 + (6-5.57)^2 + (6-5.57)^2 + \cdots + (1-5.57)^2}{13}$$
$$\rightarrow \sigma_2^2 = 4.87$$

حال داریم:

$$Score = 4.87 \times 14 + 3.09 \times 11 = 102.17$$

با توجه به امتیازات بدست آمده، امتیازی بهینه است که کمینه باشد. به عبارتی نیاز است گروه‌بندی‌ای را انتخاب کرد که تفاوت میزان واریانس مقادیر موجود در آن کمینه باشد. با این تفاسیر گروه دوم را انتخاب می‌کنیم و خط آستانه را بر روی مقدار ۱۰ قرار می‌دهیم.

سوال ۴

در ابتدا تاثیر هر یک از پارامترهای موجود در تابع cv.adaptiveThreshold را شرح می‌دهیم.

- تاثیر c: این متغیر میزان تاکید بر روی پارامتر c را مشخص می‌کند. به طوری که با افزایش آن، تصویر روشن‌تر شده و با کاهش آن تصویر بدست آمده تیره‌تر می‌شود. به طور کلی این پارامتر تاثیرگذارتر است در نتیجهٔ نهایی‌ای که بدست می‌آوریم. همچنین هر چه مقدار c بزرگتر باشد، در تصویر نهایی، پیکسل‌هایی که در سایه بوده‌اند بیشتر تحت تاثیر قرار می‌گیرند و یک مقداری fade می‌شوند.
- تاثیر block_size: هر چه این پارامتر بزرگ‌تر باشد، میزان هموارسازی تصویر بیشتر می‌شود و تصویر نهایی بدست آمده دارای خطوط ضخیم‌تر خواهد شد و نویزهای موجود در تصویر تقویت می‌شوند.
- تاثیر thresholdType: این پارامتر در صورتی که بر روی THRESH_BINARY_INV تنظیم شده باشد، نتیجهٔ نهایی را به صورتی تغییر می‌دهد که سفیدها سیاه شده و سیاه‌ها سفید شوند.

فرض می‌کنیم که تصویر اولیهٔ داده شده دارای متن مشکی و نیز اشیاه موجود در تصویر نیز به رنگ سیاه هستند. با توجه به این فرض داریم:

- پارامتر thresholdType تنها در تصویر نهایی برابر با THRESH_BINARY_INV است؛ چرا که تصویر در این حالت به صورت پیش‌زمینهٔ سیاه و پس‌زمینهٔ سفید درآمده است.
- در صورتی که در یک تصویر c بزرگتر باشد، پیکسل‌های موجود در سایه fade می‌شوند. با توجه به این نکته، تصاویر q4_1 و q4_3 و نیز q4_5 تصاویری هستند که در آن‌ها c بزرگتر بوده، چرا که در پایین سمت چپ تصویر fade به وجود آمده است. پس داریم:
  - c=5: در تصاویر q4_2 و q4_4
  - c=30: در تصاویر q4_1، q4_3، q4_5
- برای مشخص کردن سایز بلاک، تصاویری که c در آن‌ها برابر بوده‌اند را با همدیگر مقایسه می‌کنیم. با مقایسهٔ تصاویر q4_2 و q4_4 و پررنگ‌تر شدن جزئیات در تصویر q4_2 می‌توان متوجه شد که در این تصویر سایز بلاک بزرگ‌تر است. همچنین با مقایسهٔ q4_1 و q4_3 نیز می‌توان متوجه شد، که در تصویر q4_3 مقدار سایز بلاک بزرگتر بوده است. همچنین تصویر q4_5 از نظر ساختاری شبیه به q4_3 بوده، پس سایز بلاک در این تصویر نیز بزرگ است.
  - bs=21: در تصاویر q4_4 و q4_1
  - bs=41: در تصاویر q4_2، q4_3 و نیز q4_5

سوال ۵

با توجه به سایز عنصر ساختاری داده شده، نیاز است که از *padding* با سایز یک واحد از هر طرف استفاده کرد. با توجه به خواستهٔ مسئله مبنی بر استفاده از *padding reflect*، پس از اعمال عملیات مربوطه به تصویر زیر می‌رسیم.

22	*22*	*22*	*22*	*33*	*22*	*22*	*33*	*22*	*22*
22	*22*	*22*	*22*	*33*	*22*	*22*	*33*	*22*	*22*
22	*22*	*33*	*33*	*33*	*33*	*33*	*33*	*22*	*22*
22	*22*	*22*	*22*	*33*	*22*	*33*	*44*	*22*	*22*
22	*22*	*22*	*33*	*44*	*22*	*33*	*22*	*22*	*22*
22	*22*	*22*	*44*	*22*	*22*	*44*	*33*	*22*	*22*
33	*33*	*22*	*44*	*22*	*44*	*33*	*33*	*22*	*22*
33	*33*	*33*	*33*	*33*	*33*	*22*	*33*	*22*	*22*
33	*33*	*33*	*44*	*33*	*22*	*44*	*22*	*44*	*44*
33	*33*	*33*	*44*	*33*	*22*	*44*	*22*	*44*	*44*

- گسترش: برای گسترش عملیات بیشینه‌گیری را در ناحیهٔ گفته شده و به ازای مقادیر ۱ موجود در عنصر ساختاری انجام می‌دهیم.

33	*33*	*33*	*33*	*33*	*33*	*33*	*33*
33	*33*	*33*	*33*	*33*	*44*	*44*	*44*
33	*33*	*44*	*44*	*44*	*44*	*33*	*22*
22	*44*	*44*	*44*	*44*	*44*	*44*	*33*
33	*44*	*44*	*44*	*44*	*44*	*33*	*33*
33	*44*	*33*	*44*	*44*	*33*	*33*	*33*
33	*44*	*44*	*44*	*44*	*44*	*44*	*44*
33	*44*	*44*	*44*	*44*	*44*	*44*	*44*

- سایش: برای سایش عملیات کمینه‌گیری را در ناحیهٔ گفته شده و به ازای مقادیر ۱ موجود در عنصر ساختاری انجام می‌دهیم.

22	22	22	22	22	22	22	22
22	22	22	22	22	22	22	22
22	22	22	22	33	22	22	22
22	22	22	22	22	22	22	22
22	22	22	22	22	22	22	22
22	22	22	22	22	22	22	22
22	22	22	22	22	22	22	22
33	33	33	33	22	22	22	22

سوال ۶

با توجه به مطالب مطرح شده در کلاس درس و نیز اسلایدها، عملیات داده شده مربوط به تشخیص نقاط گوشهٔ بیرونی موجود در یک شکل است که با استفاده از عملگر *Hit or Miss* صورت می‌گیرد.

همچنین ساختار گوشه‌ای که توسط این فرآیند به دست می‌آید بایستی به صورت عنصر ساختاری شمارهٔ ۱ باشد. و اینکه سه پیکسل مربوط به عنصر ساختار شمارهٔ ۲ نیز بایستی ۰ باشد. به عبارتی با ترکیب این دو عنصر به عنصر زیر می‌رسیم:

0	-1	-1
1	1	-1
0	1	0

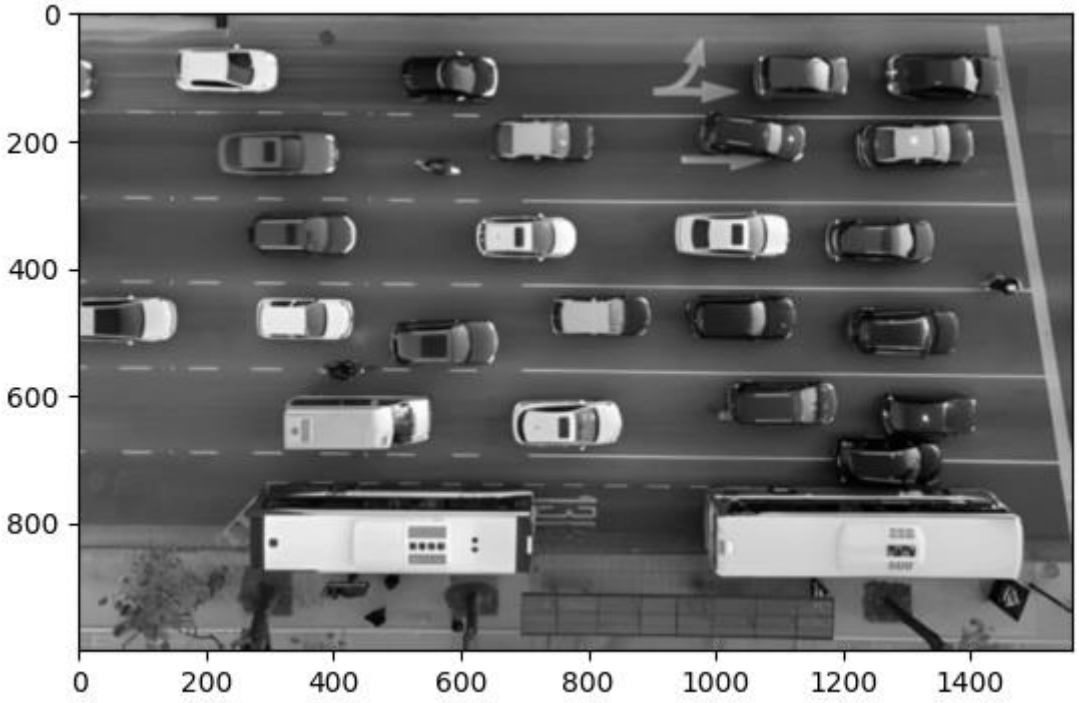
پس از اعمال این عملگر با استفاده از دو عنصر ساختاری داده شده به شکل زیر خواهیم رسید که با توجه به آن، نقاط گوشهٔ بیرونی که ساختار بالا را داشتند، مقدار ۱ را خواهند داشت و باقی پیکسل‌ها صفر می‌شوند.

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

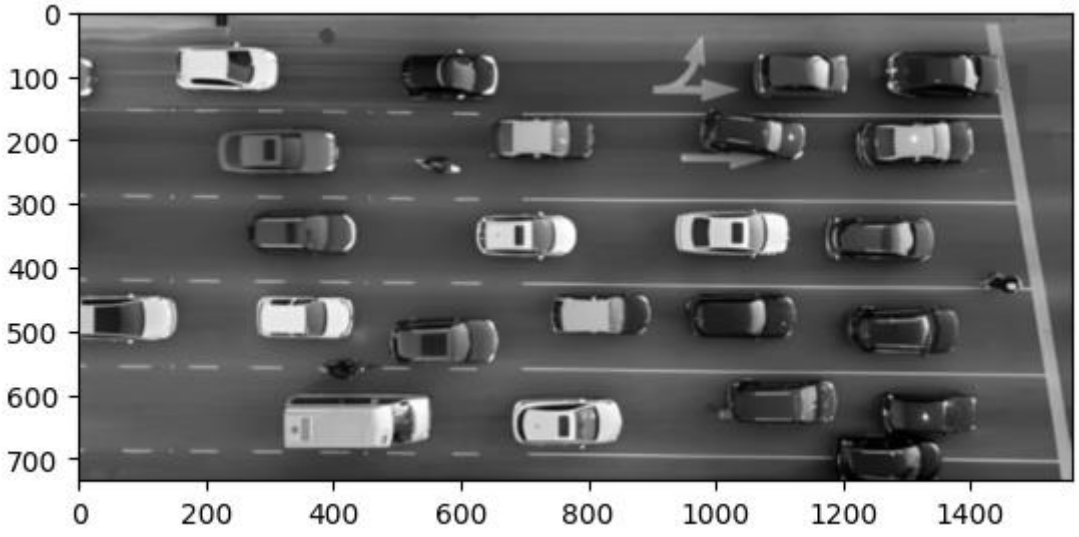
سوال ۷

بخش الف

در ابتدا تصویر را خوانده و آن را به تصویر خاکستری تبدیل می‌کنیم. در گام بعدی بر روی تصویر خاکستری بدست آمده میانگین‌گیری گاوسی می‌زنیم و آن را هموار می‌کنیم.



سپس با توجه به اینکه بخش پایینی تصویر داده شده تصویر باینری را دچار نویز می‌کند، نیاز است که بخش پایینی تصویر را کراپ کنیم.

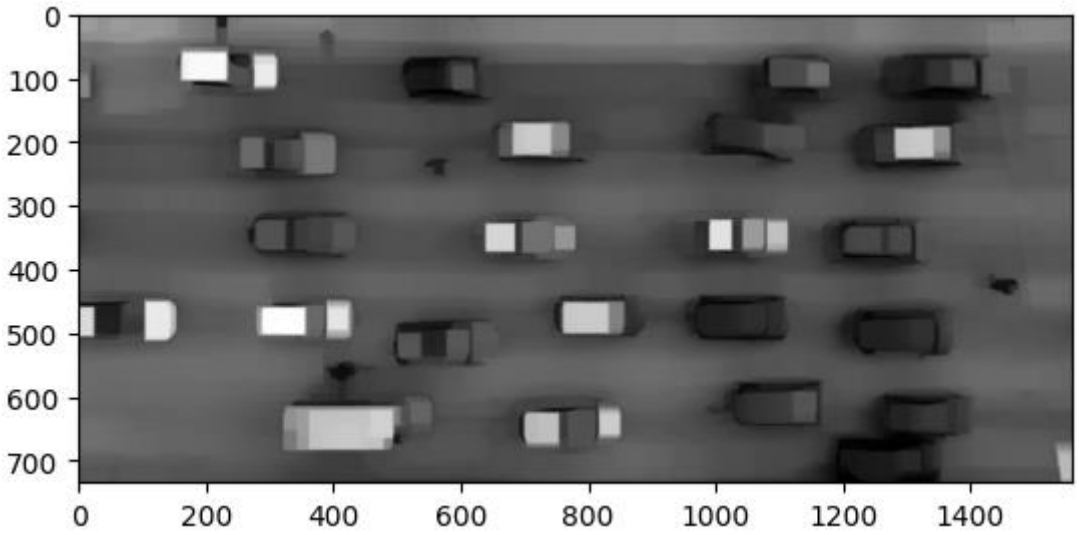


سپس خطوط عمودی و افقی تصویر را بر روی تصویر خاکستری حذف می‌کنیم. برای این منظور از قطعه کد زیر استفاده می‌کنیم:

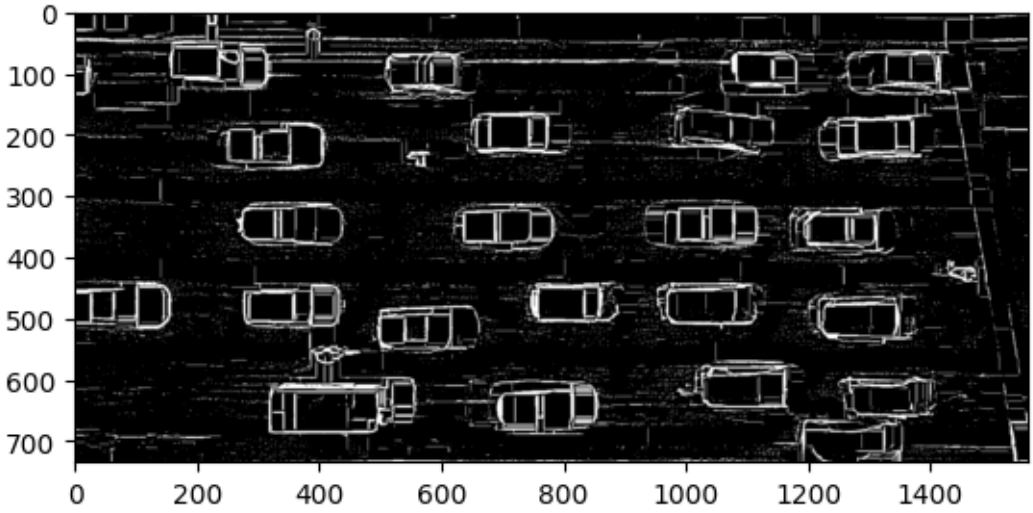
```
# Removing vertical and horizontal lines
opening = cv2.morphologyEx(crop_img, cv2.MORPH_OPEN, np.ones((31, 1), np.uint8)) # Vertical lines
opening = cv2.morphologyEx(opening, cv2.MORPH_OPEN, np.ones((1, 31), np.uint8)) # Horizontal lines
plt.imshow(opening, cmap='gray')
```

خط اول برای حذف خطوط عمودی بوده و خط دوم برای حذف خطوط افقی می‌باشد.

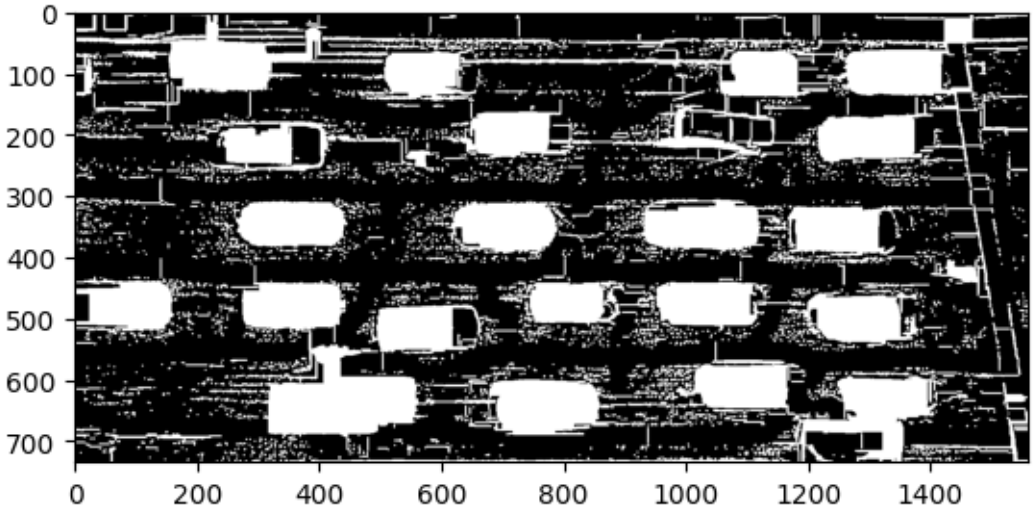
نتیجهٔ بدست آمده به صورت زیر خواهد شد:



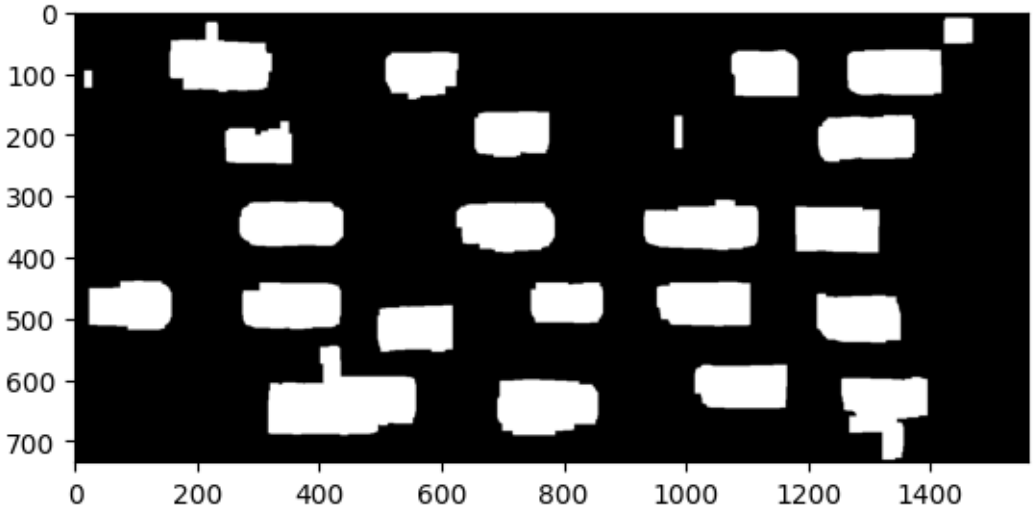
سپس در ابتدا *Threshold* را بر روی تصویر اعمال می‌کنیم تا یک تصویر باینری تحویل بگیریم. و در گام بعدی یک میانگین‌گیری گاوسی اعمال می‌کنیم تا نویز موجود در تصویر از بین برود.



پس از این خطوط کنتورهای موجود در تصویر را پیدا می‌کنیم. دلیل یافتن کنتورها پیدا کردن، پر کردن مستطیل‌های مربوط به ماشین‌های موجود در تصویر است.



بعد از این که خطوط بالا را اجرا کردیم، دو عمل *opening* و *dilation* پشت سر هم انجام می‌دهیم تا نویزهایی که در تصویر بالا وجود دارند از بین بروند و همچنین مستطیل‌های موجود در تصویر بزرگتر شوند.



در پایان با استفاده از *Contour* مستطیل‌های موجود در تصویر پیدا می‌کنیم و بر روی تصویر اولیه قرار می‌دهیم.

```
cnts, _ = cv2.findContours(cars.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

print("Number of cars found:", len(cnts))
```

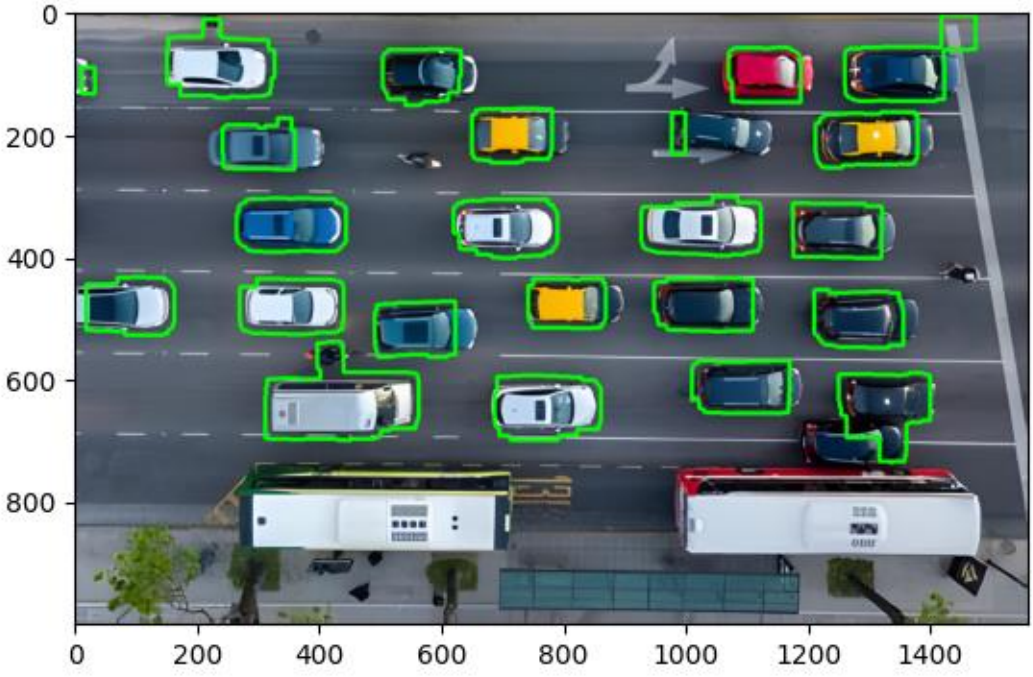
```
Number of cars found: 23
```

```
for c in cnts:
    # compute the center of the contour, then detect the name of the
    # shape using only the contour
    M = cv2.moments(c)
    cX = int((M["m10"] / M["m00"]))
    cY = int((M["m01"] / M["m00"]))

    c = c.astype("float")
    c = c.astype("int")
    cv2.drawContours(img, [c], -1, (0, 255, 0), 6)

# show the output image
plt.imshow(img)
```

نتیجهٔ نهایی به صورت زیر خواهد شد:



همانطور که مشاهده می‌شود، الگوریتم نوشته شده و توابع *Morphology* استفاده شده به خوبی توانسته است که ماشین‌ها را تشخیص دهد. تنها مشکل موجود در الگوریتم، ماشین سیاه رنگ در پایین سمت راست تصویر است که با توجه به رنگ مشکی و نیز چسبندگی آن به ماشین بالایی‌اش امکان تشخیص مستقل آن سخت بوده است. بنابراین همانطور که در تصاویر بالا مشهود است، تعداد ماشین‌های نهایی گزارش شده، ۲۳ مورد هستند.

سوال ۸

در ابتدا توابع مورد نیاز برای اعمال عملیات‌های مورفولوژی را تعریف می‌کنیم.

- تابع *Erosion*:

```python
def erosion(image, kernel):
    """
    Performs erosion on a binary image using a given kernel.
    """
    rows, cols = image.shape
    kernel_rows, kernel_cols = kernel.shape

    # Create an output image with same dimensions
    output = np.zeros_like(image)

    # Iterate over the image
    for i in range(rows - kernel_rows + 1):
        for j in range(cols - kernel_cols + 1):
            # Check if the kernel fits entirely within the image region
            if np.all(image[i:i+kernel_rows, j:j+kernel_cols] >= kernel):
                output[i+kernel_rows//2, j+kernel_cols//2] = 1

    return output
```

  - در این تابع عملیات *Erosion* صورت می‌گیرد. به این صورت که کرنل بر روی تصویر می‌لغزد و در هر ناحیه بررسی می‌شود که آیا کرنل به طور کامل در ناحیه قرار می‌گیرد یا خیر.

- تابع *Dilation*:

```python
def dilation(image, kernel):
    """
    Performs dilation on a binary image using a given kernel.
    """
    image = image.copy()
    output = np.zeros_like(image)

    kernel = np.array(kernel)
    kernel_size = kernel.shape[0]
    img_padded = np.pad(image, kernel_size//2, mode='constant')

    for i in range(kernel_size//2, img_padded.shape[0] - kernel_size//2):
        for j in range(kernel_size//2, img_padded.shape[1] - kernel_size//2):
            output[i-kernel_size//2, j-kernel_size//2] = np.max(img_padded[i-kernel_size//2:i+kernel_size//2 + 1, j-kernel_size//2:j+kernel_size//2 + 1] * kernel)
    return output
```

  - در این تابع، کرنل را بر روی تصویر می‌لغزانیم، و در هر ناحیه بزرگترین مقدار موجود در تصویر متناظر با مقادیر ۱ موجود در کرنل را استخراج می‌کنیم. با توجه به اینکه از کرنل دایره‌ای استفاده می‌شود نیازی به چرخاندن تصویر و قرینه کردن آن نیست.

- تابع *Opening*:

```python
def opening(image, kernel):
    image = image.copy()
    eroded_img = erosion(image, kernel)
    opened_img = dilation(eroded_img, kernel)
    return opened_img
```

  - در این تابع طبق تعریف در ابتدا تصویر را سایش می‌کنیم و سپس آن را گسترش می‌دهیم.

- تابع *Dilation* به اندازهٔ k مرتبه:

```python
def _dilate_k(image, kernel, k):
    eroded_img = image.copy()
    for _ in range(k):
        eroded_img = dilation(eroded_img, kernel)
    return eroded_img
```

  - در این تابع، تصویر را به تعداد k مرتبه *dilate* می‌کنیم. از این تابع در هنگام تشکیل تصویر اولیه با استفاده از ساختمان‌های بدست آمده، استفاده می‌کنیم.

- تابع یافتن ساختمان یک تصویر

```python
def find_structure(image, kernel):
    fst_term = image.copy()
    snd_term = opening(fst_term, kernel)
    structures = [fst_term - snd_term]
    while np.any(snd_term == 1):
        fst_term = erosion(fst_term, kernel)
        snd_term = opening(fst_term, kernel)
        structures.append(np.array(fst_term - snd_term))
        if len(structures) % 10 == 0:
            print(f'{len(structures)}th iteration')
    return structures
```

  - در این تابع، طبق تعریف موجود در اسلایدها، در ابتدا تصویر تحت عملگر *opening* باز می‌کنیم. و سپس اختلاف تصویر بدست آمده با تصویر اولیه را به عنوان اولین بخش ساختمان‌های یافت شده برای تصویر در آرایه‌ای قرار می‌دهیم. سپس تا زمانی تصویر حاصل از بازشدگی، مقدار یک را در خود دارد، ادامه می‌دهیم.
  - در هر گام از حلقهٔ نوشته شده، در ابتدا تصویر را دچار سایش می‌کنیم و سپس تصویر سایش یافته را تحت عملگر *opening* باز می‌کنیم. سپس اختلاف بین دو تصویر بدست آمده را در آرایهٔ تعریف شده قرار می‌دهیم.
  - همچنین یک لاگ کردن هم در پایان حلقه قرار داده شده است.
  - در پایان آرایهٔ شامل تمام ساختمان‌ها را بازمی‌گردانیم.

- تابع ساختن تصویر از روی ساختمان‌های بدست آمده

```python
def find_image(structures, kernel):
    image = np.zeros_like(structures[0])
    for i, s in tqdm(enumerate(structures)):
        s_k = _dilate_k(s, kernel, i)
        image = np.logical_or(image, s_k)
        if i % 10 == 0:
            print(f'{i}th iteration')
    return np.array(image, np.uint8)
```

  - در هر گام، بر روی مجموعه ساختمان‌های بدست آمده پیش می‌رویم. به این صورت که در هر گام به اندازهٔ اندیس ساختمان در آرایه، عملیات گسترش را انجام می‌دهیم. با استفاده از همان تابع *dialte_k_* که پیشتر

تعریف شده است. در هر گام تصویر تازه بدست آمده را با تصویری که تاکنون ساخته‌ایم *OR* می‌کنیم تا تصویر تازه‌ای ساخته شود.

- تابع ساختن تصویر مربوط به ساختمان‌ها با استفاده از *OR* کردن همهٔ آن‌ها:

```
def get_structure_image(structures):
    image = np.zeros_like(structures[0])
    for s in tqdm(structures):
        image = np.logical_or(image, s)
    image = np.array(image, np.uint8)
    return image
```

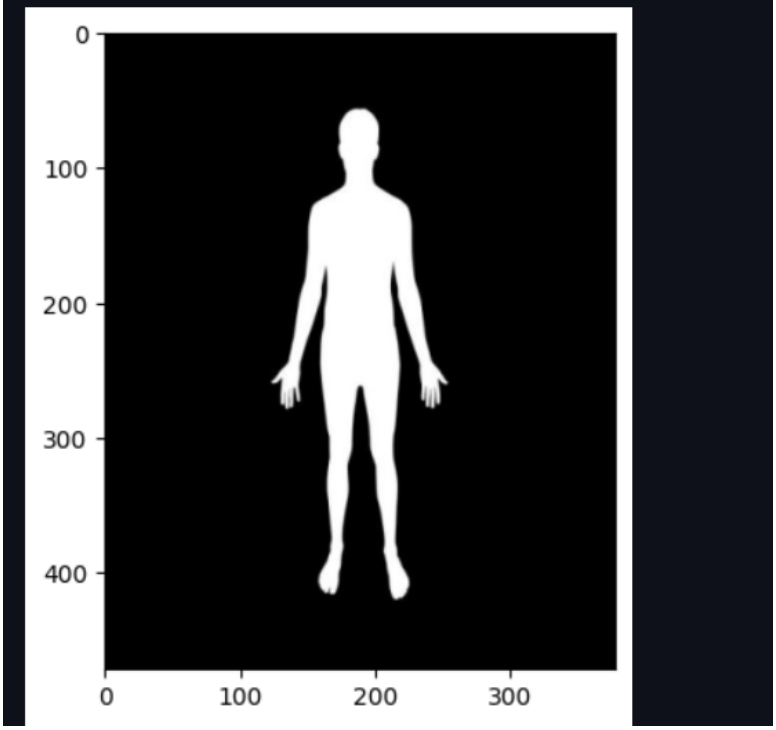
  - در این تابع، تصویر مربوط به مجموعهٔ ساختمان‌های ساخته شده را ایجاد می‌کنیم. به این صورت که هر عضو موجود در آرایهٔ بدست آمده را با همدیگر *OR* می‌کنیم تا تصویر نهایی ساخته شود.

تصاویر داده شده به طور کامل باینری نیستند. برای همین نیاز است که در ابتدا هر یک را باینری کنیم. برای این منظور به صورت دستی یک *Threshold* تعریف می‌کنیم و مقادیر بالاتر از آن را یک و مقادیر کمتر از آن را صفر می‌کنیم. همچنین جهت تسریع فرآیند پردازش، تصویر را در ابتدا *resize* می‌کنیم. همچنین برای نمایش هر یک از تصاویر، هر یک را در ۲۵۵ ضرب می‌کنیم تا بتوان آن را نمایش داد.

تصویر *8.5*:

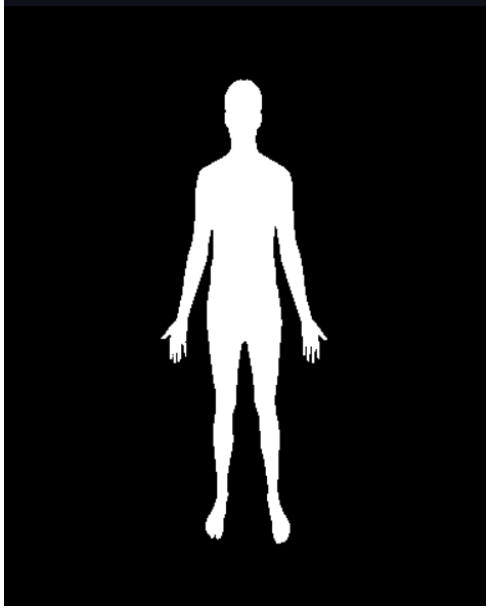
```
image1 = cv2.imread('q8_5.jpg', cv2.IMREAD_GRAYSCALE)
# image1 = cv2.resize(image1, (image1.shape[0]//2, ima
plt.imshow(image1, 'gray')
```

```
<matplotlib.image.AxesImage at 0x78102d2e9e10>
```



```
image1[image1<=160] = 0
image1[image1>160] = 1
show_binary_image(image1)
```

```
ndarray (472, 378)  show data
```

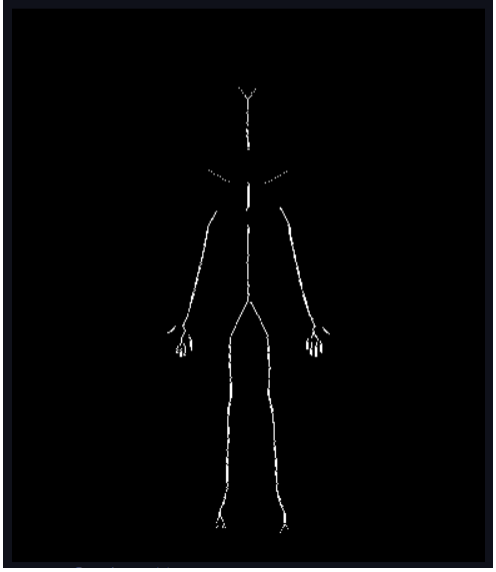


با استفاده از تابع تعریف شده، ساختمان تصویر را تولید می‌کنیم.

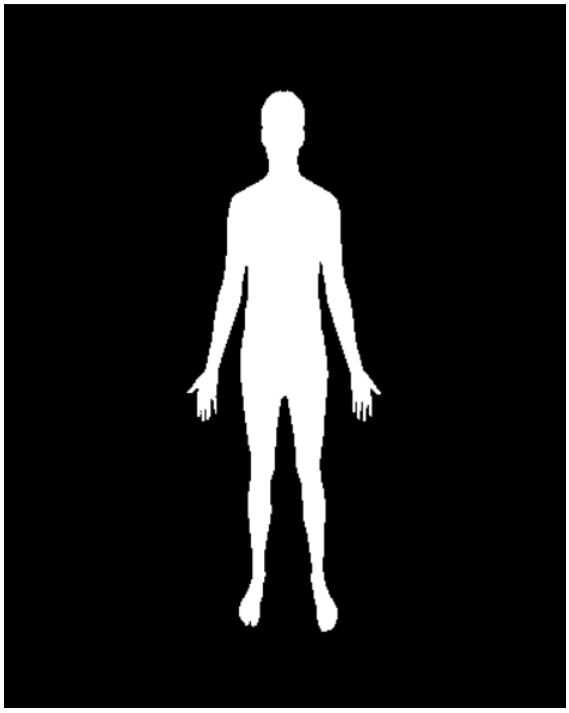
```
structures1 = find_structure(image1, kernel)
get_structure_image(structures1) * 255
```

```
10th iteration
20th iteration
30th iteration
100%|██████████| 38/38 [00:00<00:00, 10901.75it/s]

ndarray (472, 378)  show data
```



در پایان تصویر اولیه را دوباره از ساختمان‌های ایجاد شده، بدست می‌آوریم.

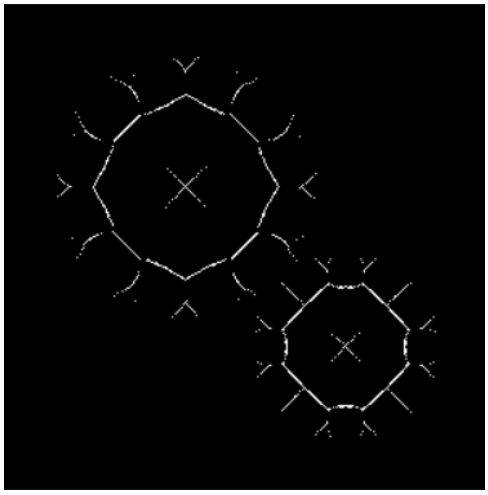


تصویر *8.6*:

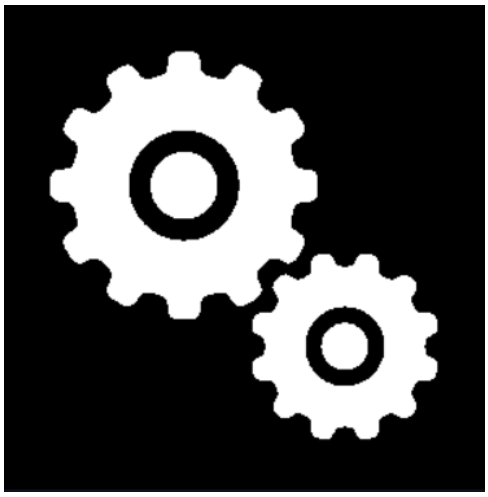
```
blurred2 = cv2.GaussianBlur(image2, (15, 15), 0)
blurred2[blurred2<=150] = 0
blurred2[blurred2>150] = 1
show_binary_image(blurred2)
```

ndarray (300, 300) show data

ساختمان تصویر به صورت زیر بدست می‌آید:



سپس تصویر اولیه را از ساختمان‌های بدست آمده تولید می‌کنیم.



تصویر *8.7*:

در ابتدا، تصویر را باینری می‌کنیم.

سپس ساختمان‌های تصویر را بدست می‌آوریم.

و در مرحلهٔ نهایی تصویر اولیه را ایجاد می‌کنیم.

سوال ۹

در ابتدا عناصر ساختاری را به صورت زیر تعریف می‌کنیم. علت تعریف هر یک را در کنار آن قرار داده‌ایم.

مرز چپ	0	0	0
	-1	1	0
	0	0	0

مرز راست	0	0	0
	0	1	-1
	0	0	0

مرز بالا	0	-1	0
	0	1	0
	0	0	0

مرز پایین	0	0	0
	0	1	0
	0	-1	0

با اعمال این عناصر بر روی تصویر، به تصویر نهایی زیر می‌رسیم. در این تصویر، پیکسل‌ها به رنگ آبی مربوط به اولین عنصر، رنگ قرمز مربوط به دومین عنصر، رنگ سبز مربوط به سومین عنصر و رنگ زرد مربوط به آخرین عنصر است. همچنین اولین پیکسلی که در یک عنصر ساختاری قرار می‌گیرد را به آن عصنر نسبت می‌دهیم.