داکیومنت تکلیف شماره ۴

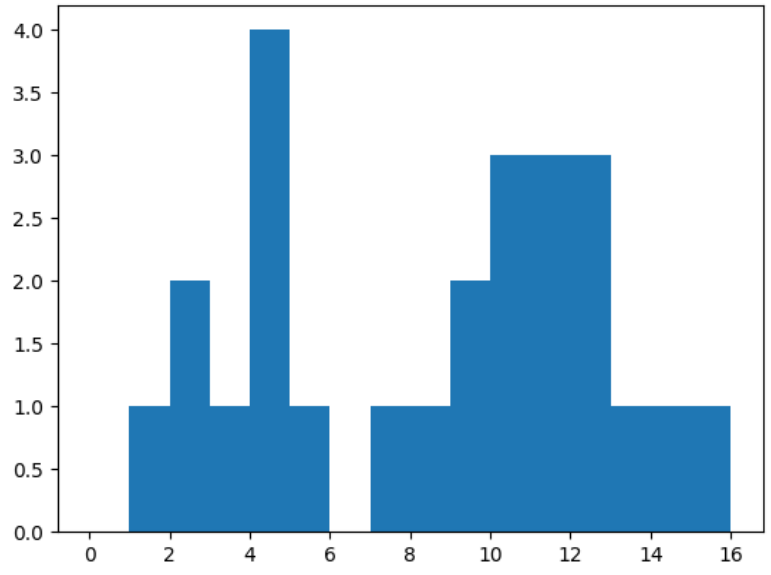
■ سوال ۱

الف)

```python
import numpy as np
import matplotlib.pyplot as plt

matrix = np.array([[12, 13, 5, 4, 9],
                   [11, 7, 10, 10, 1],
                   [8, 11, 3, 2, 2],
                   [9, 12, 4, 4, 4],
                   [10, 11, 12, 15, 14]])

# Flatten the matrix into a 1D array
flat_matrix = matrix.flatten()

# Calculate the histogram using NumPy
hist, bins = np.histogram(flat_matrix, bins=16, range=[0, 16])

# Plot the histogram
plt.hist(flat_matrix, bins=16, range=[0, 16])
plt.show()

# Print the statistics
print("Mean:", np.mean(flat_matrix))
print("Median:", np.median(flat_matrix))
print("Mode:", np.argmax(hist))
print("Variance:", np.mean((flat_matrix - np.mean(flat_matrix)) ** 2))
```

نتیجه محاسبه:

Mean: 8.12
Median: 9.0
Mode: 4
Variance: 16.7456



ب)

```python
import numpy as np

# Define the histogram values
hist = np.array([[12, 13, 5, 4, 9],
                 [11, 7, 10, 10, 1],
                 [8, 11, 3, 2, 2],
                 [9, 12, 4, 4, 4],
                 [10, 11, 12, 15, 14]])

# Define the threshold levels
thresholds = [9.5, 11.5]

best_threshold = None
best_otsu = 0

# Calculate Otsu's algorithm for each threshold level
for threshold in thresholds:
    # Split the histogram based on the threshold
    hist_below = hist[hist <= threshold]
    hist_above = hist[hist > threshold]

    # Calculate weights
    w_below = np.sum(hist_below) / np.sum(hist)
    w_above = np.sum(hist_above) / np.sum(hist)

    # Calculate variances
    if len(hist_below) > 0:
        variance_below = np.sum((hist_below - np.mean(hist_below))**2) / np.sum(hist_below)
    else:
        variance_below = 0

    if len(hist_above) > 0:
        variance_above = np.sum((hist_above - np.mean(hist_above))**2) / np.sum(hist_above)
    else:
        variance_above = 0

    # Calculate Otsu's value
    otsu = w_below * variance_below + w_above * variance_above
    print("Threshold:", threshold, "Otsu Value:", otsu)

    # Check if the current threshold is better
    if otsu > best_otsu:
        best_otsu = otsu
        best_threshold = threshold

# Print the results
print("Best Threshold: ", best_threshold)
print("Best Otsu Value: ", best_otsu)
```

بعد از این که مقادیر داده شده را در کد قرار دهیم، خروجی کد به این صورت خواهد بود:

```
Threshold: 9.5 Otsu Value: 0.5643236074270557
Threshold: 11.5 Otsu Value: 1.1361161524500907
Best Threshold:  11.5
Best Otsu Value:  1.1361161524500907
```

■ سوال ۲

الف)

روش اتسو (Otsu) و گاوسی اتسو (Gaussian Otsu) دو روش برای تشخیص آستانه‌ای تصاویر هستند که بر اساس واریانس بین دسته‌ها و همچنین واریانس داخل دسته‌ها عمل می‌کنند. این دو روش را می‌توان از نظر سرعت و دقت مقایسه کرد.

در روش اتسو، واریانس بین دسته‌ها برای تعیین آستانه بهینه استفاده می‌شود. این روش بر اساس پیدا کردن آستانه‌ای است که واریانس بین دسته‌ها در آن بیشینه شود. روش اتسو برای تصاویر با توزیع نمودار دوقلو و روشن‌تاریک کارآمد است و معمولاً به طور سریع عمل می‌کند. با این حال، برای تصاویری با توزیع نمودار پیچیده تر، ممکن است دقت پایینی داشته باشد.

روش گاوسی اتسو یک توسعه از روش اتسو است که واریانس داخل دسته‌ها را نیز در محاسبات آستانه لحاظ می‌کند. در این روش، برای تعیین آستانه بهینه، آستانه‌ای انتخاب می‌شود که واریانس بین دسته‌ها حداکثر شود و در عین حال واریانس داخل دسته‌ها حداقل شود. این روش به طور کلی از روش اتسو دقیق‌تر است و به تصاویر با توزیع نمودار پیچیده‌تر نیز مناسب است. با این حال، به طور معمول، این روش زمان بیشتری نسبت به روش اتسو برای محاسبه آستانه نیاز دارد.

بنابراین، اگر سرعت مهمترین عامل برای ما باشد، روش اتسو پیشنهاد می‌شود زیرا سریعتر از روش گاوسی اتسو است. اما اگر دقت نتایج برای ما مهم تر است و تصاویر شما دارای توزیع پیچیده‌تری هستند، روش گاوسی اتسو می‌تواند بهترین گزینه باشد، اگرچه زمان محاسبه آستانه بیشتری نیاز دارد.

ب) در الگوریتمOtsu ، هدف اصلی کمینه کردن واریانس داخل کلاس‌ها است، نه بیشینه کردن واریانس بین کلاس‌ها. این نکته ممکن است گیج‌کننده به نظر برسد، اما به طور خلاصه می‌توان گفت که کمینه کردن واریانس داخل کلاس‌ها منجر به بیشینه کردن واریانس بین کلاس‌ها می‌شود.

برای توضیح بیشتر، ابتدا باید تعریف واریانس داخل کلاس و واریانس بین کلاس‌ها را بیان کنیم:

- **واریانس داخل کلاس (within-class variance) : این مقدار نشان می‌دهد که چقدر داده‌های هر کلاس به یکدیگر نزدیک هستند و به ازای هر کلاس محاسبه می‌شود.**
- **واریانس بین کلاس‌ها (between-class variance) :این مقدار نشان می‌دهد که چقدر داده‌های دو کلاس مختلف از یکدیگر دور هستند و برای هر زوج از کلاس‌ها محاسبه می‌شود.**

هدف اصلی الگوریتمOtsu ، پیدا کردن آستانه‌ای است که واریانس داخل کلاس‌ها را کمینه کند. یعنی در نقطه آستانه بهینه، واریانس داخل کلاس‌ها بیشترین مقدار ممکن خود را دارد. این امر به طور مستقیم منجر به بیشینه کردن واریانس بین کلاس‌ها می‌شود، زیرا هنگامی که واریانس داخل کلاس‌ها بیشینه می‌شود، واریانس بین کلاس‌ها حداقل مقدار خود را دارد.

به عبارت دیگر، کمینه کردن واریانس داخل کلاس‌ها همانند بیشینه کردن واریانس بین کلاس‌ها است و این دو مفهوم با هم مترادف هستند. بنابراین، در الگوریتمOtsu ، هدف نهایی کمینه کردن واریانس داخل کلاس‌ها است و این هدف به طور مستقیم به بیشینه کردن واریانس بین کلاس‌ها منجر می‌شود.

■ سوال ۳

این تابع برای انجام الگوریتم ناحیه بندی تصویر استفاده می‌شود. الگوریتم ناحیه بندی می‌تواند به شناسایی و تفکیک بخش‌های مختلف تصویر کمک کند. در اینجا، تابع segmentتصویر ورودی را دریافت می‌کند و تصویر ناحیه بندی شده را برمی‌گرداند.

حالت کلی الگوریتم ناحیه بندی در این تابع به این صورت است:

۱. **ساخت نسخه ای از تصویر ورودی برای ناحیه بندی.**

۲. **تبدیل تصویر به float64برای محاسبات ریاضی.**

۳. **تعریف اجزاء مختلفی که باید ناحیه بندی شوند با ویژگی‌ها و پارامترهای خودشان.**

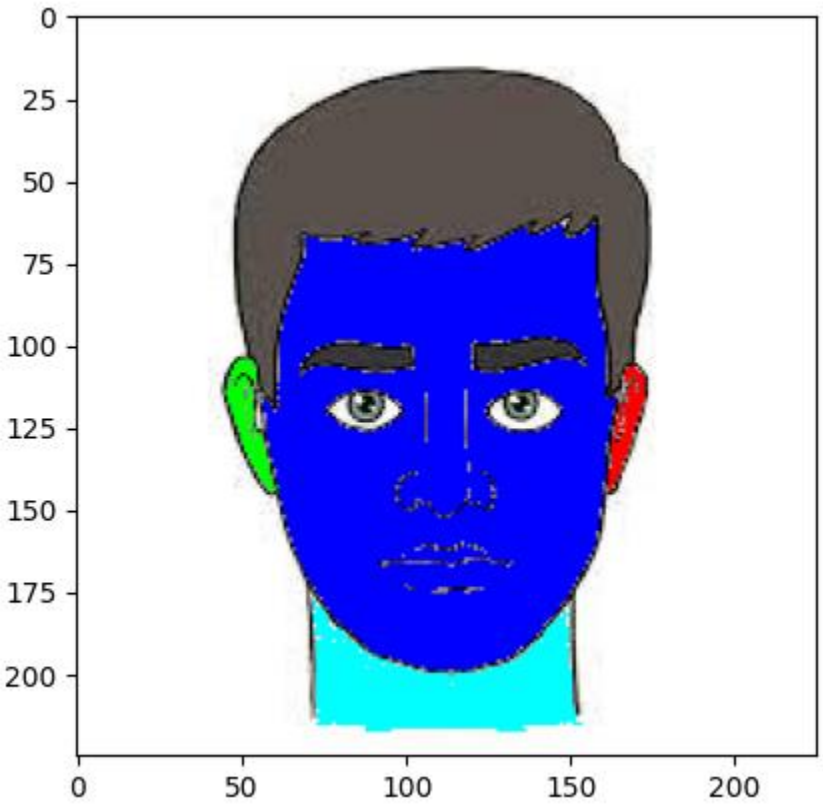
۴. **برای هر یک از اجزاء، مراحل زیر را انجام می‌دهد:**

- **محاسبه تفاوت مطلق بین تصویر و پیکسل مرجع (seed) مربوط به این اجزا و تبدیل آن به نوع داده uint8.**
- **تبدیل تصویر تفاوت به رنگ سیاه و سفید.(grayscale)**
- **ساختن یک ماسک دودویی بر اساس آستانه مشخص شده.**

- **مقداردهی اولیه به صف و آرایه بازدید شده برای الگوریتم رشد منطقه.(region growing)**
- **اضافه کردن پیکسل مرجع به صف.**
- **الگوریتم رشد منطقه:**
  - **استخراج مختصات فعلی پیکسل از صف.**
  - **بررسی وضعیت بازدید شده بودن پیکسل و یا خارج بودن آن از محدوده تصویر یا عدم تطابق با ماسک.**
  - **تنظیم رنگ پیکسل ناحیه بندی شده.**
  - **نشانه‌گذاری پیکسل به عنوان بازدید شده.**
  - **اضافه کردن همسایگان پیکسل فعلی به صف.**

۵. **برگرداندن تصویر ناحیه بندی شده.**

با استفاده از این الگوریتم، تصویر ورودی در بخش‌های مختلفی مانند بینی، گوش چپ، گوش راست و گردن تقسیم می‌شود و هر بخش با یک رنگ مشخص مشخص می شود:



■ سوال ۴

الف)

گسترش:

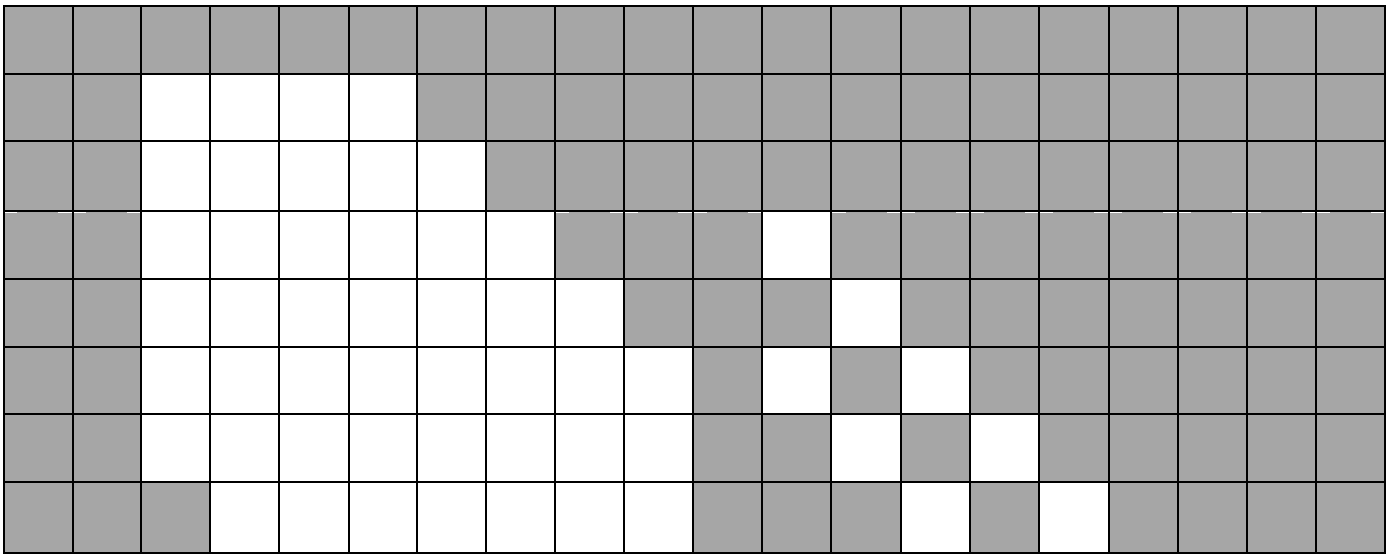
۷۰	۷۰	۷۰	۷۰	۷۰	۷۰	۷۰	۷۰
۷۰	۷۰	۷۰	۷۰	۷۰	۷۰	۷۰	۷۰
۸۰	۸۰	۸۰	۸۰	۸۰	۸۰	۸۰	۷۰
۷۰	۸۰	۷۰	۷۰	۸۰	۷۰	۷۰	۷۰
۷۰	۸۰	۸۰	۸۰	۸۰	۸۰	۷۰	۶۰
۷۰	۸۰	۸۰	۸۰	۸۰	۸۰	۸۰	۶۰
۷۰	۸۰	۸۰	۸۰	۸۰	۸۰	۸۰	۷۰
۷۰	۷۰	۸۰	۸۰	۸۰	۸۰	۸۰	۷۰

سایش:

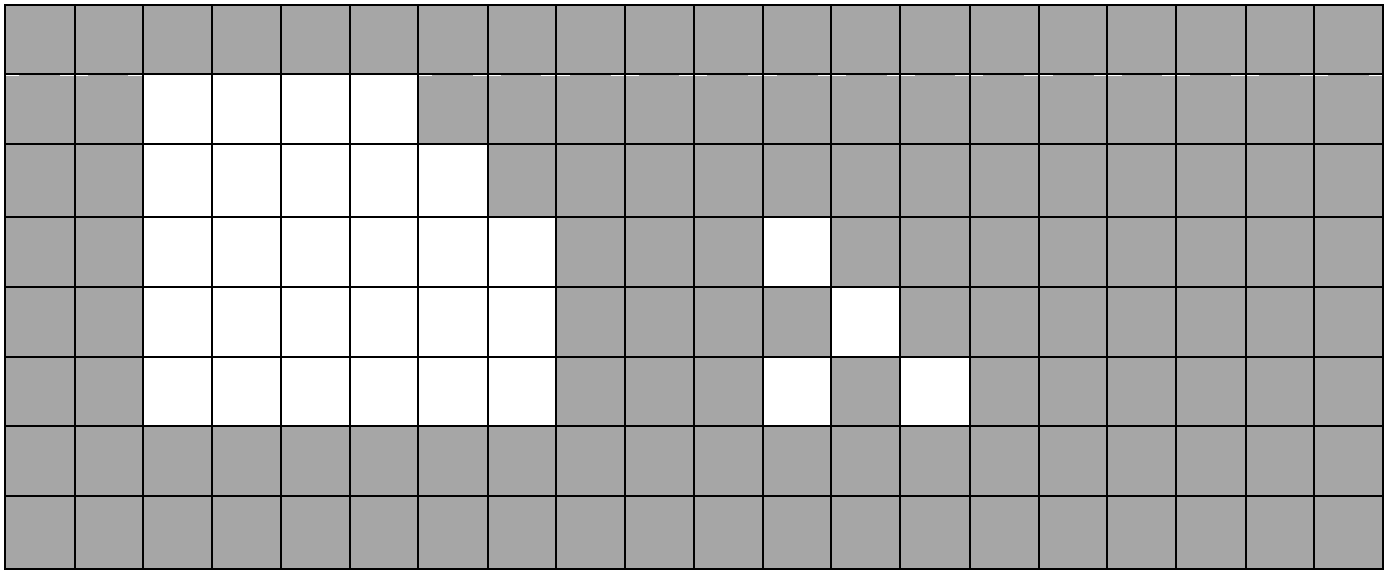
۶۰	۶۰	۶۰	۶۰	۶۰	۶۰	۶۰	۶۰
۶۰	۶۰	۶۰	۶۰	۶۰	۶۰	۶۰	۶۰
۶۰	۶۰	۷۰	۶۰	۷۰	۷۰	۶۰	۶۰
۶۰	۶۰	۶۰	۶۰	۶۰	۷۰	۷۰	۷۰
۶۰	۶۰	۶۰	۶۰	۶۰	۷۰	۶۰	۶۰
۶۰	۶۰	۶۰	۶۰	۶۰	۶۰	۶۰	۶۰
۶۰	۶۰	۶۰	۶۰	۶۰	۶۰	۶۰	۶۰
۶۰	۶۰	۶۰	۶۰	۶۰	۶۰	۶۰	۶۰

ب) برای حالت **close**:

- **Dilate**:

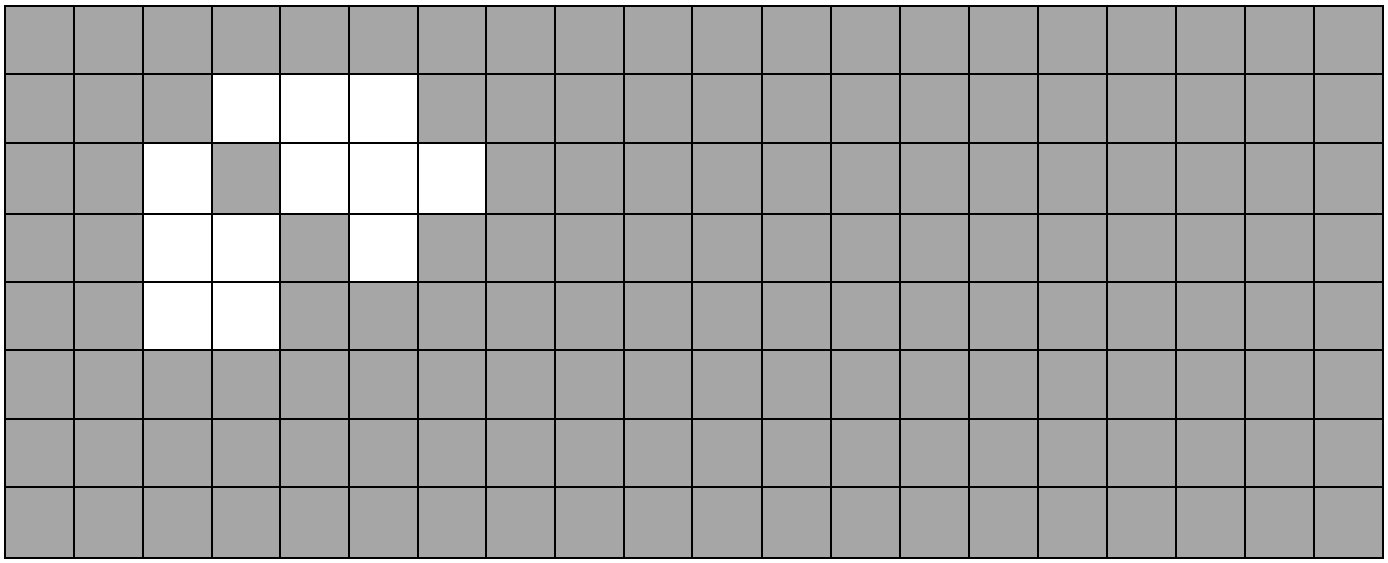


- **Enrode**:

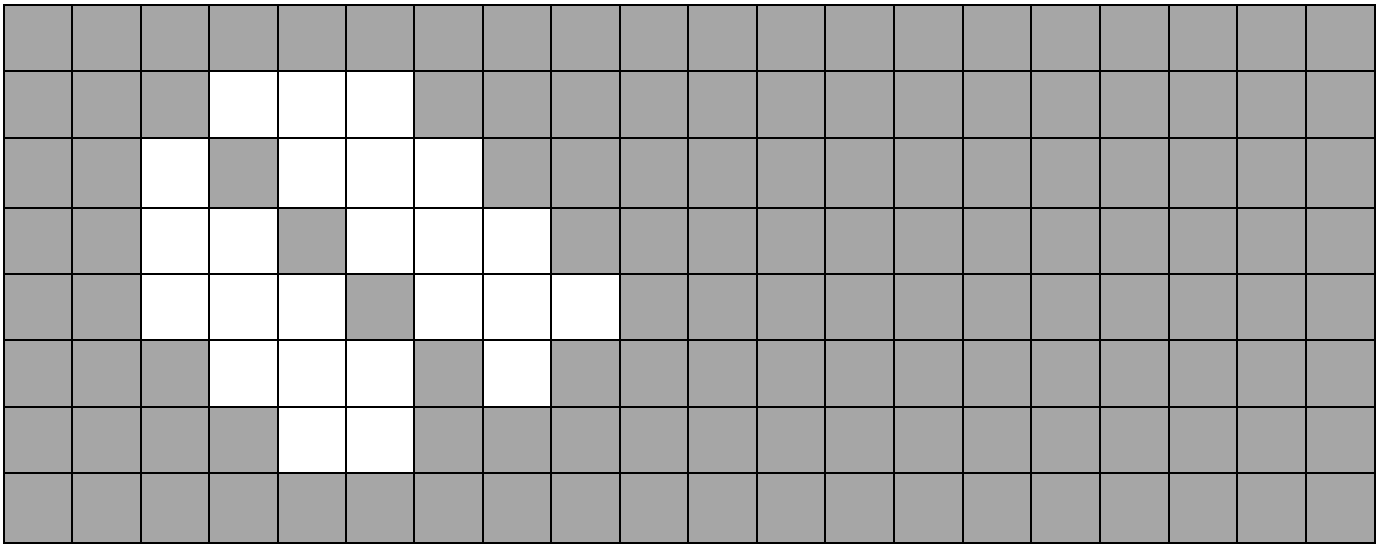


برای حالت open:

- Enrode:

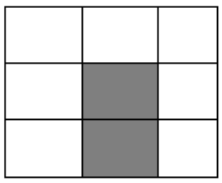


- Dilate:



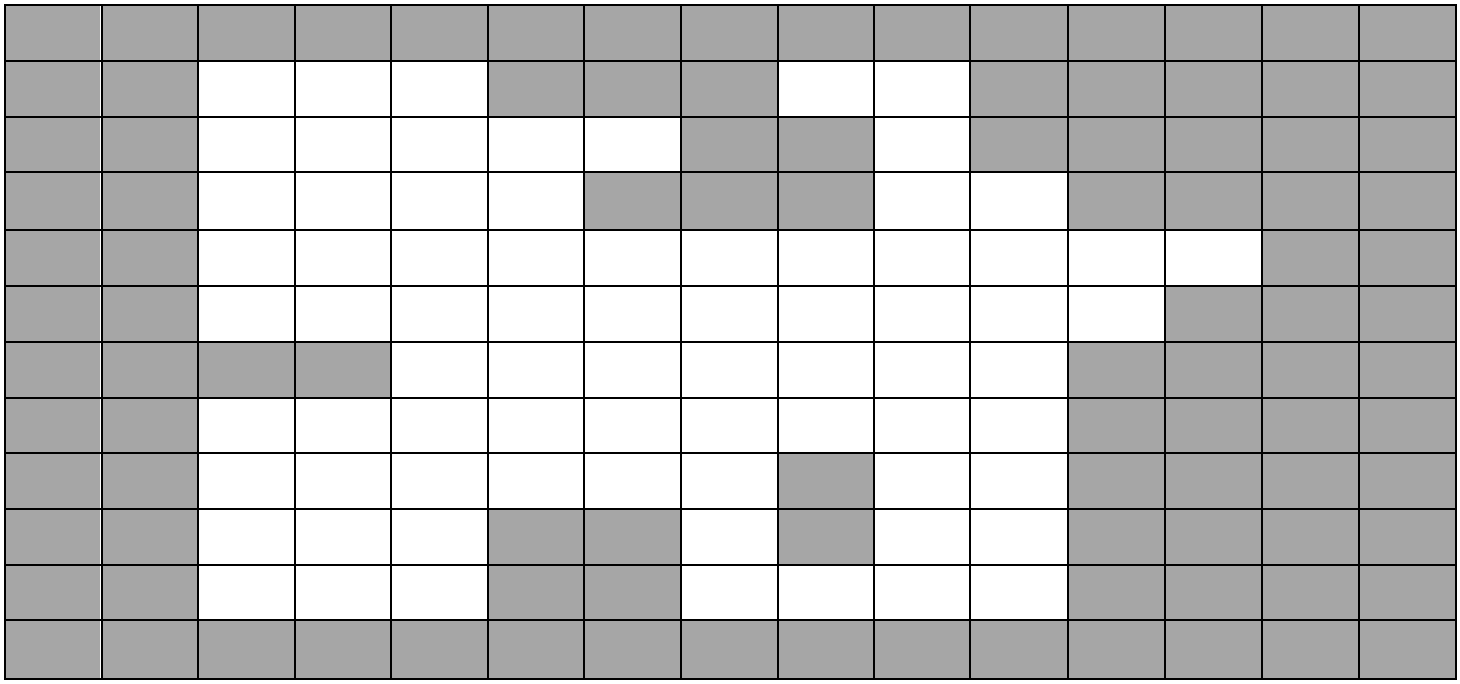
■ **سوال ۵**

الف) برای حذف خط اضافی در وسط تصویر و حفظ عدد صفر در نهایت تصویر، می توان از عملگر opening با استفاده از عنصر ساختاری چهارم استفاده کرد:

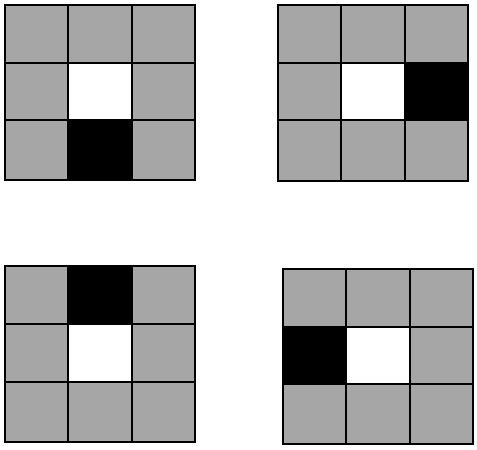


عملگر opening ابتدا عمل dilate (توسعه) را بر روی تصویر انجام می دهد و سپس عمل erode (کاهش) را بر روی نتیجه توسعه اعمال می کند. در نتیجه، خط اضافی در وسط تصویر حذف شده است و عدد صفر نیز حفظ شده است.

ب)

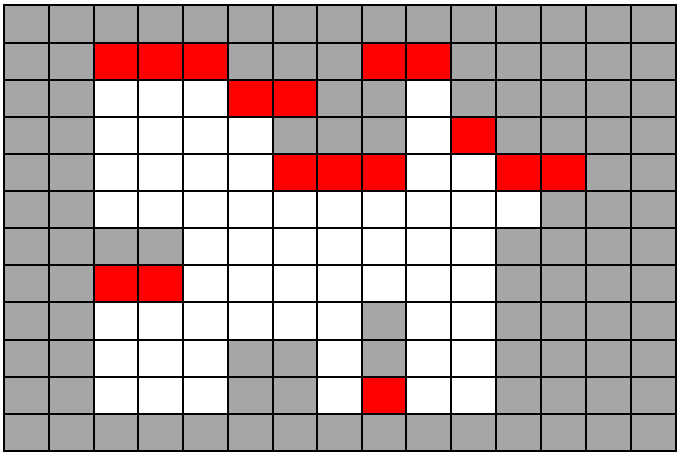


عناصر ساختاری ای که برای مرز ها استفاده می کنیم:

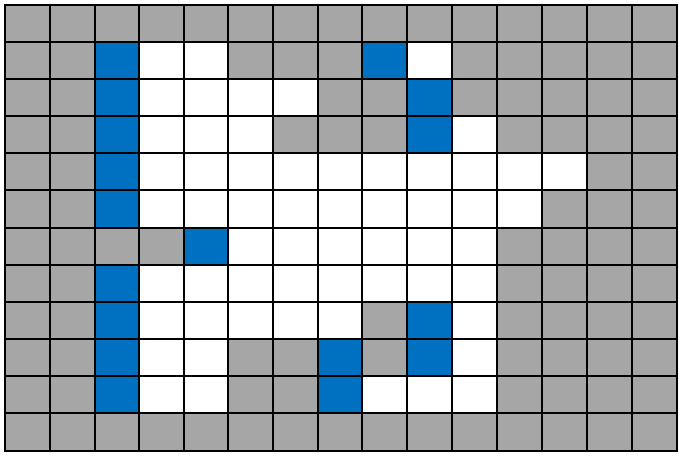


حال برای همه حالات خروجی را رسم میکنیم:

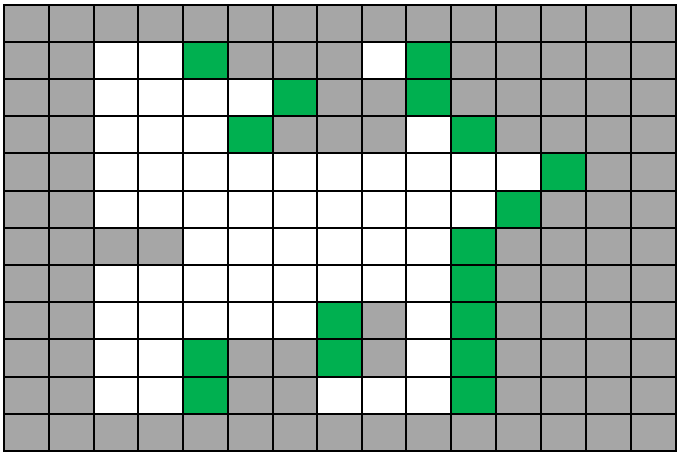
لبه بالایی:



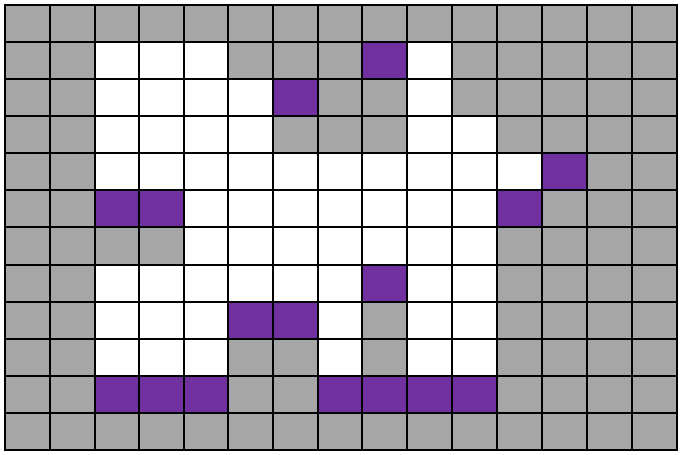
لبه چپ:

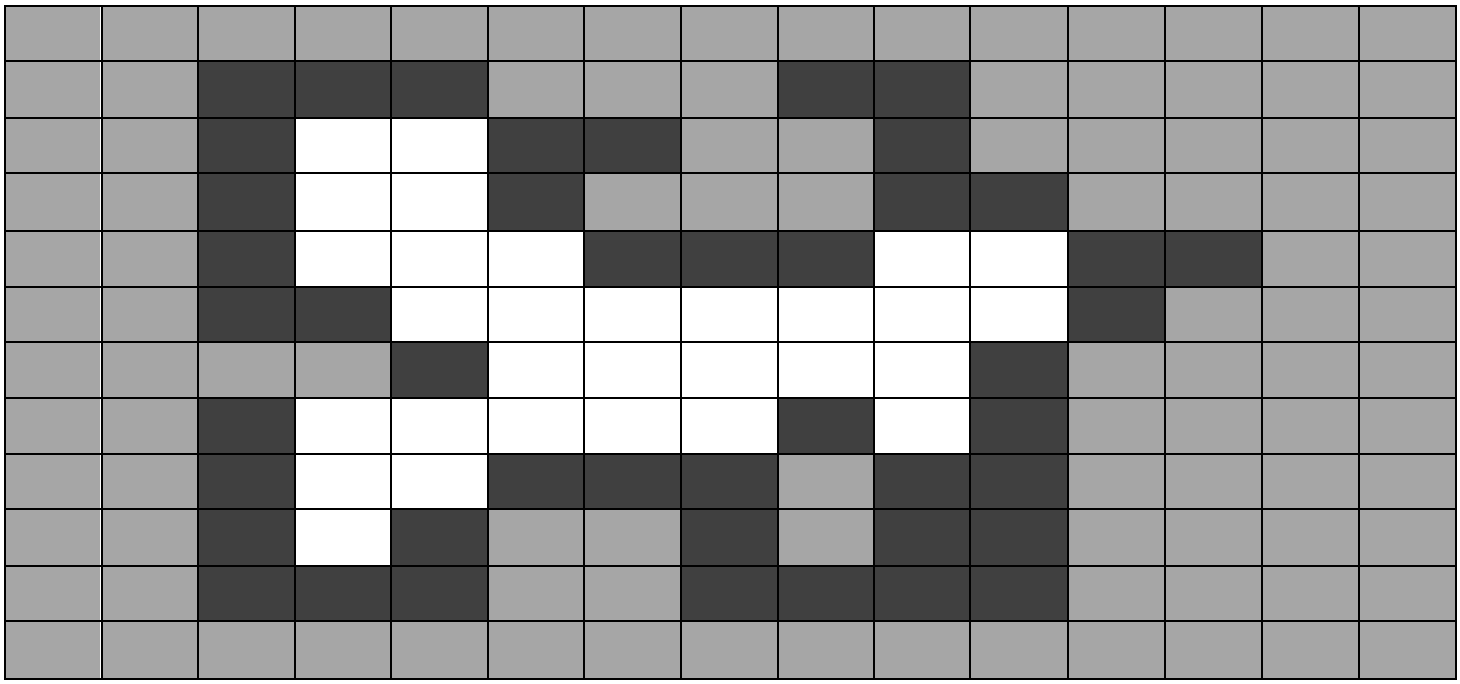


لبه راست:



لبه پایین:





■ **سوال ۶**

الف) کد ها دارای کامنت هستند و توضیحات کامل در آن ها داده شده است.

ب) کد ها دارای کامنت هستند و توضیحات کامل در آن ها داده شده است.

ج) این کد برای پیدا کردن اسکلت یک تصویر ورودی استفاده می شود. اسکلت یک تصویر، ساختار ساده‌تر و نمایش کمتر جزئیات تصویر اصلی است. این عمل برای تحلیل و استخراج ویژگی‌های اصلی تصویر مفید است.

در این کد، ابتدا تصویر ورودی به یک تصویر دودویی تبدیل می‌شود. سپس یک تصویر خالی به نام skeleton ایجاد می‌شود که برای ذخیره اسکلت تصویر استفاده می‌شود. همچنین کرنل مورفولوژی نیز به عنوان ورودی دریافت می‌شود.

سپس با استفاده از عملیات erode و dilate ، تصویر دودویی به آرایه skeleton تبدیل می‌شود. در هر مرحله، تصویر دودویی اول با استفاده از کرنل ارود می‌شود و سپس با استفاده از کرنل توسعه داده می‌شود. سپس تصویر توسعه داده شده از تصویر اولیه کم می‌شود تا اختلاف بین دو تصویر به دست آید. این اختلاف سپس با استفاده از عملیات OR بین تصویر skeleton و اختلاف بدست آمده، به skeleton اضافه می‌شود. عملیات‌های ارود و توسعه و اضافه کردن به skeleton تا زمانی ادامه می‌یابند که تصویر دودویی کاملاً ارود شود (هیچ پیکسل غیر صفری باقی نماند).

در نهایت، skeletonبه صورت معکوس شده (تصویر skeleton با مقادیر سفید و سیاه برعکس شده) برگردانده می‌شود. این تصویر نهایی اسکلت تصویر ورودی را نمایش می‌دهد و می‌تواند برای تحلیل و استخراج ویژگی‌های مربوطه استفاده شود.