به نام خدا

تمرین سری چهارم

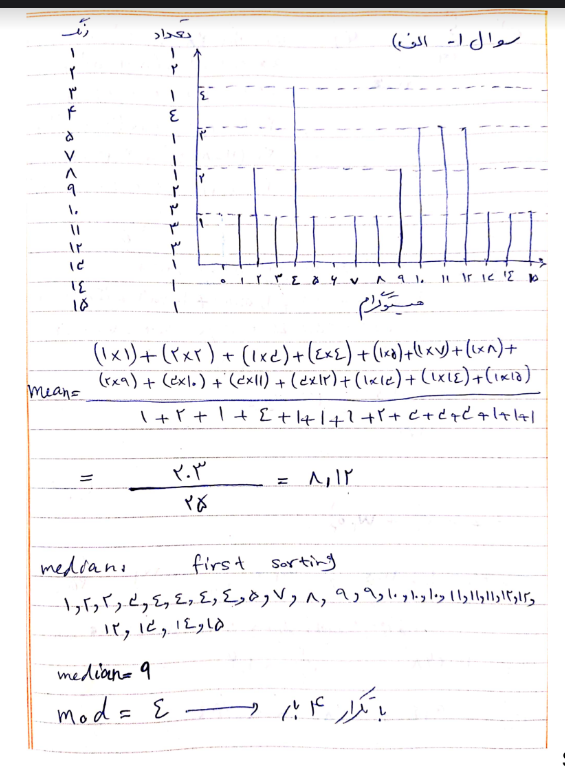
درس بینایی کامپیوتر

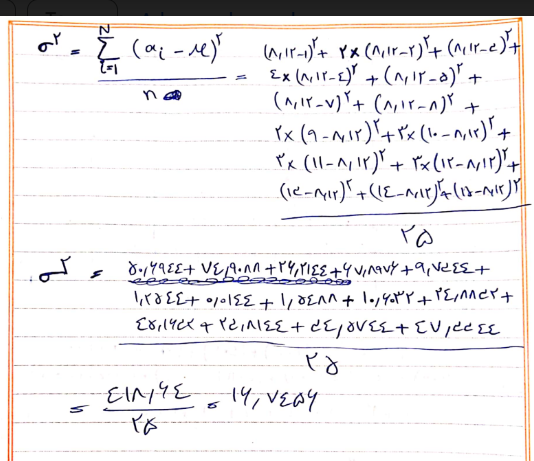
سینا علی‌نژاد

شماره دانشجویی: 99521469

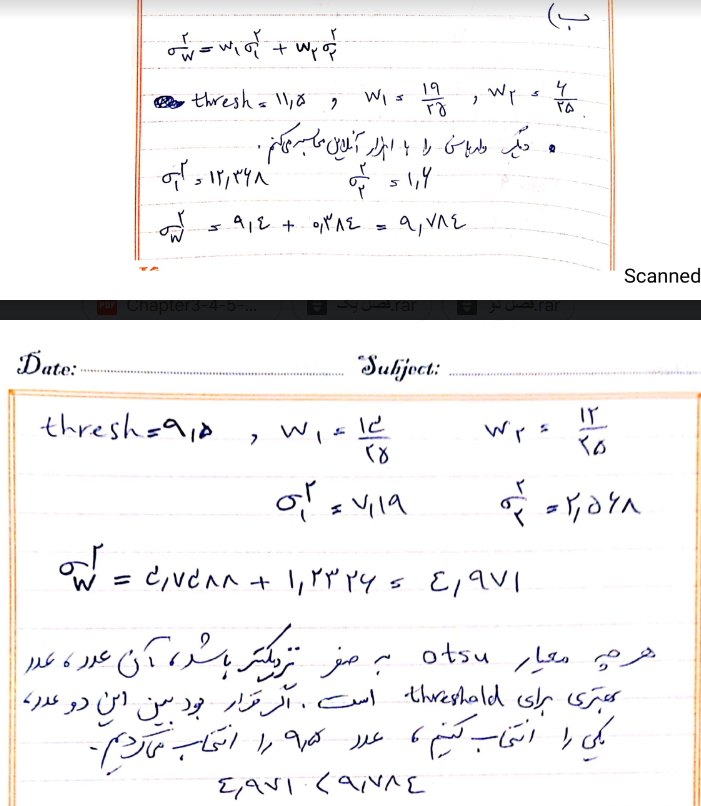
1-

الف)



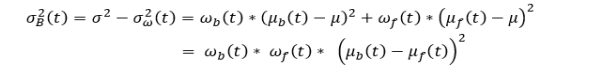


ب)



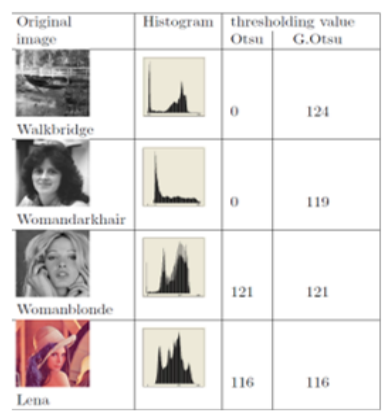
2-

در روش Gaussian otsu میانگین هر دسته را بدست آورده که اختلاف به توان دوی این میانگین‌ها را واریانس بین دسته‌ای میگوییم و هرچه این میانگین‌ها اختلاف بیشتری داشته باشند، نشان‌دهنده این است که عدد threshold ما عدد بهتری بوده و معیار Gaussian otsu عدد بزرگتری خواهد شد. فرمول این روش بدین صورت است که وزنهای دو دسته که تعداد اعضای آن دسته نسبت به کل اعضا است، در هم ضرب شده و حاصل در توان دوی اختلاف میانگین دو دسته ضرب می‌شود. برای حالتی که میانگین هر دو دسته بالا باشند یعنی اختلافشان نزدیک به صفر شود، این معیار نزدیک صفر می‌شود که این یعنی مقدار خوبی برای threshold انتخاب نکردیم.



الف) سرعت Gaussian otsu همواره بیشتر است زیرا اگر threshold را تغییر دهیم، بدست آوردن میانگین جدید دسته‌ها با کم و زیاد کردن مقادیری که از دسته سمت راست کم شد و به دسته چپی اضافه شد، قابل محاسبه است.

دقت: دقت Gaussian otsu در حالتی که دو حالت یا دو mode داشته باشیم بیشتر است، منظور از mode محدوده‌ای است که هیستوگرام مقادیر زیادی دارند. در حالات دیگر دقت یکسان دارند. همچنین اگر عکس ما دارای یک background و یک foreground باشد، الگوریتم otsu درست عمل نمیکند، و مقدار threshold صفر میدهد.



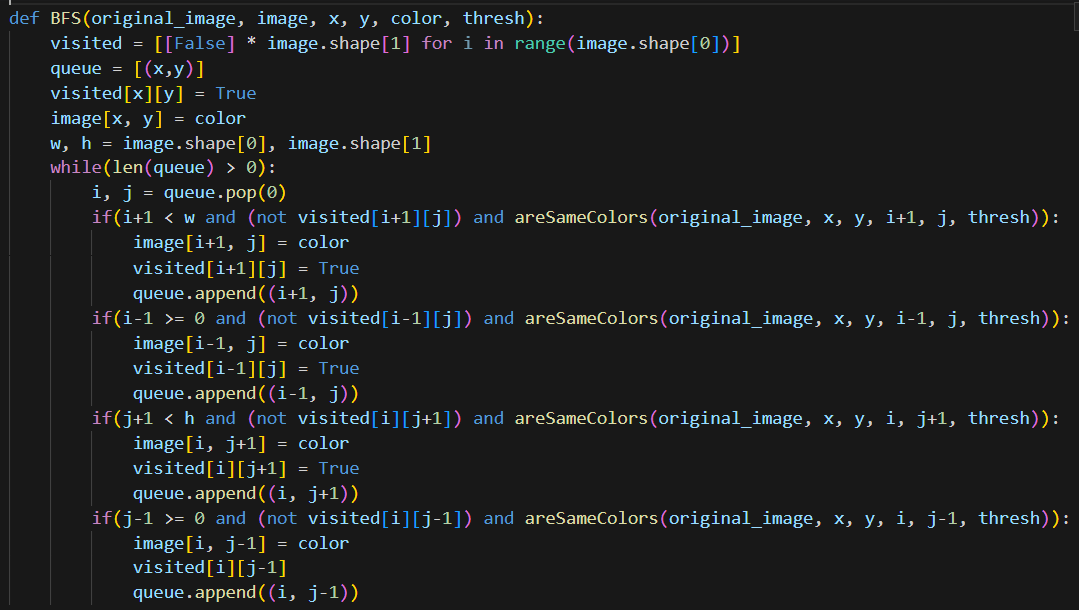
همانطور که مشخص است، برای دو عکس اول که دارای دو mode هستند، الگوریتم otsu مقدار اشتباه صفر میدهد در حالیکه Gaussian otsu درست عمل میکند. برای عکسهای دیگر مثل عکسهای سوم و چهارم، هر دو روش یک عدد را خروجی میدهند.

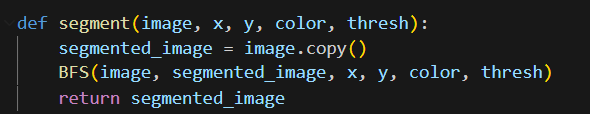
ب) بله، زیرا بیشینه شدن واریانس بین کلاسی بدین معناست که اختلاف میانگین دو دسته بیشتر است و این یعنی دو دسته ما با توجه به این threshold هر کدام در یک سمت افتاده که در سمت چپ مقادیر یک دسته و در سمت راست مقادیر دسته دیگر است، و چون thresholding به درستی انجام شده پس هر دسته داده پرت ندارد و در نتیجه واریانس درون آن کلاس کمینه است.

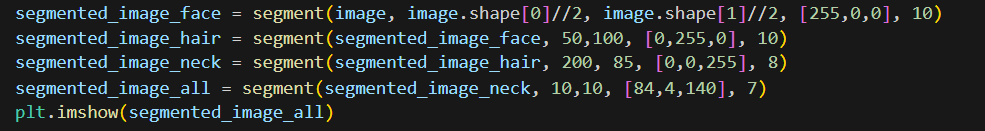
3-

برای این سوال که رشد ناحیه را باید پیاده‌سازی کنیم، من از یک نقطه seed شروع کردم و با الگوریتم BFS اطرافیانش را که با این بذر رنگ تقریبا مشابهی دارند را در صف اضافه کرده و هر بار از این صف یکی را برداشته و برای همسایه های آن این عمل را تکرار میکنم.

کد تابع BFS:

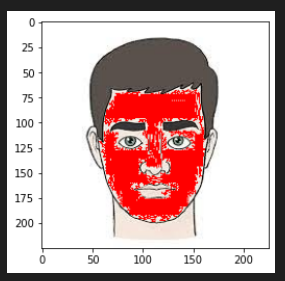






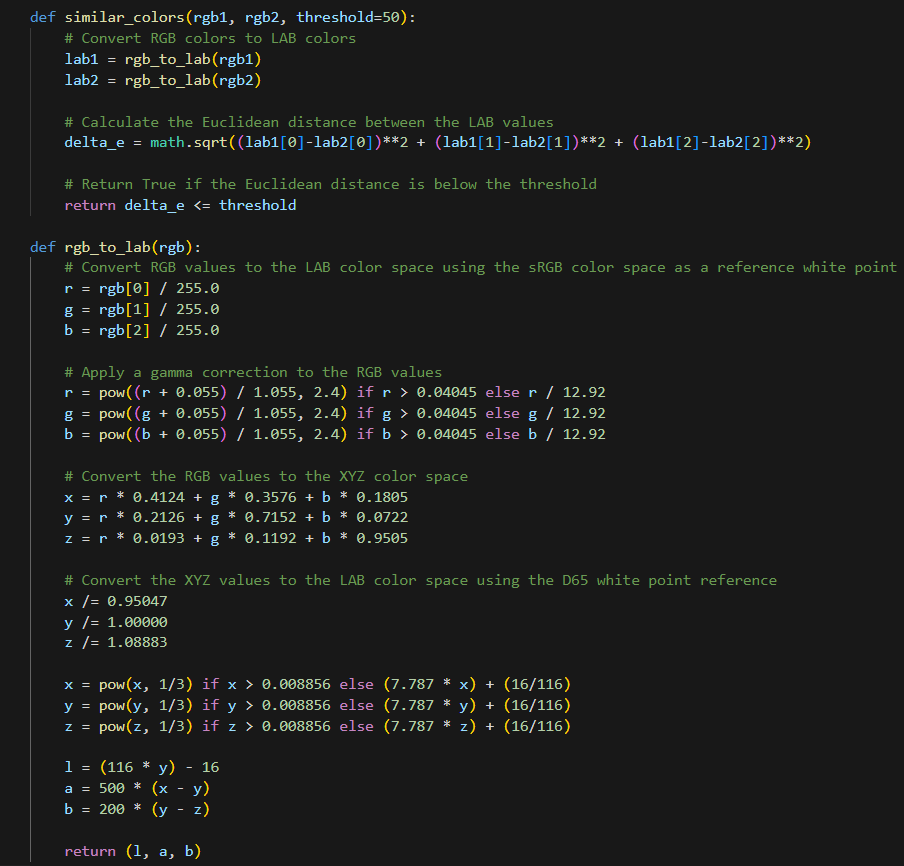
پارامتر اول تابع segment عکس است و پارامتر دوم و سوم، مختصات نقطه بذر هستند. پارامتر چهارم رنگی است که میخواهیم به ناحیه رشد داده شده بدهیم و پارامتر آخر thresholdی است که میخواهیم برای مشابه بودن دو رنگ از آن استفاده کنیم. از آنجا که خواستم همه بخش های انسان و محیط اطراف را مشخص کنم، ممکن بود به مقادیر مختلفی برای threshold نیاز پیدا کنم.

برای تابع مشابهت دو رنگ ابتدا خودم یک تابعی پیاده سازی کردم که از فاصله اقلیدسی دو رنگ استفاده میکرد و برای مثال برای صورت خروجی زیر را دریافت میکردم:

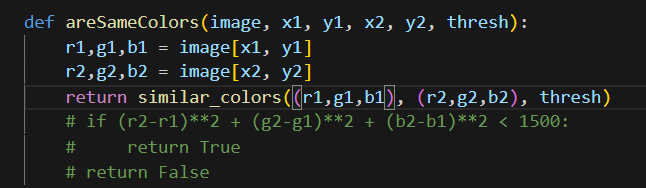


برای نقاط گوشه خوب عمل نمیکرد چون رنگ ها به صورت آهسته در حال تغییر بودند در حالیکه از نظر درک انسان آن رنگ ها نیز جزو صورت بود. برای همین در اینترنت سرچ کردم که چجوری مشابهت دو رنگ را از نظر human perception در بیارم و راهی که وجود داشت تغییر فضای رنگی rgb به lab بود که درباره این فضا چیزی نمیدانم.

و کدی که وجود داشت، به شکل زیر بود:

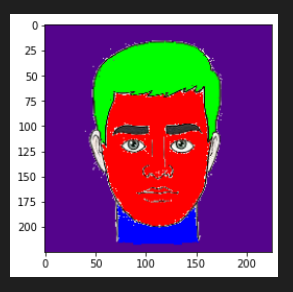


تابع rgb\_to\_lab برای تبدیل یک رنگ و بردن آن به فضای lab است و تابع similar\_colors فاصله اقلیدسی دو رنگ در فضای lab را مقایسه میکند، برعکس روش قبل که فاصله اقلیدسی در فضای rgb را محاسبه کرده بودیم.



این کد هم صرفا از همان دو تابع بالا استفاده میکند. کد کامنت شده مربوط به روش قبلی بود که پیاده کرده بودم.

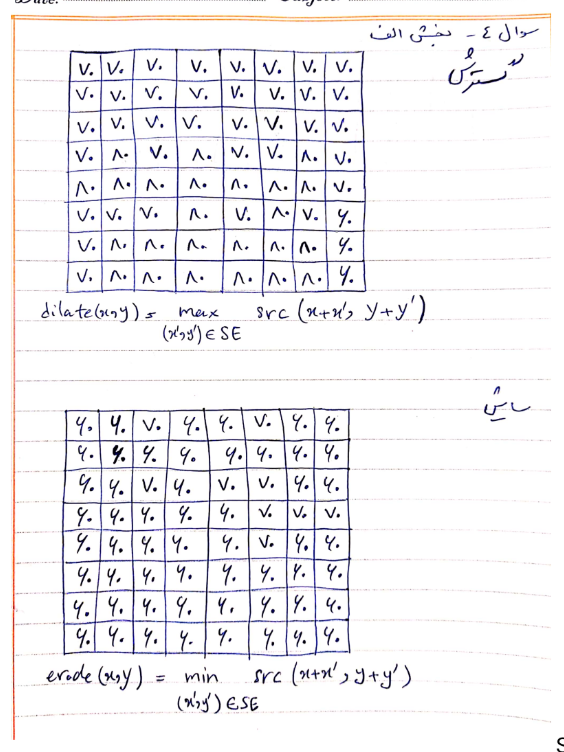
و در نهایت خروجی آخر به شکل زیر است:



روی سیستم من 7 ثانیه طول کشید تا خروجی نهایی ظاهر بشه. همانطور که میبینید، تابع مشابهت دو رنگ خیلی بهتر عمل کرده است.

4-

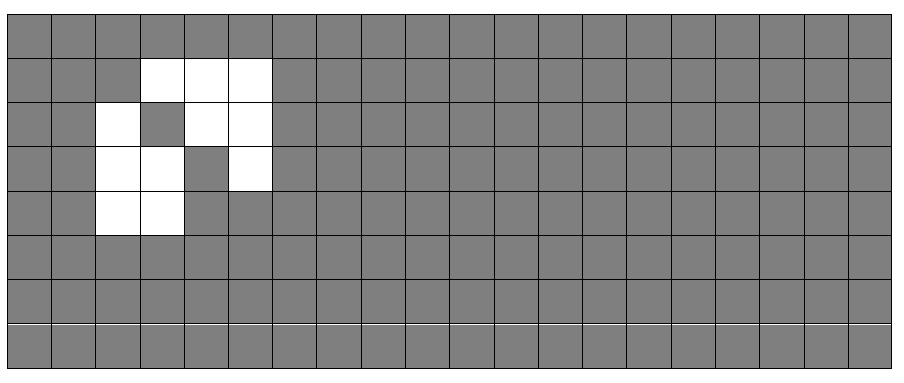
الف)



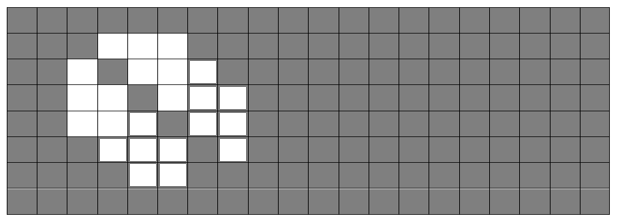
ب)

ابتدا عملگر باز را میزنیم:

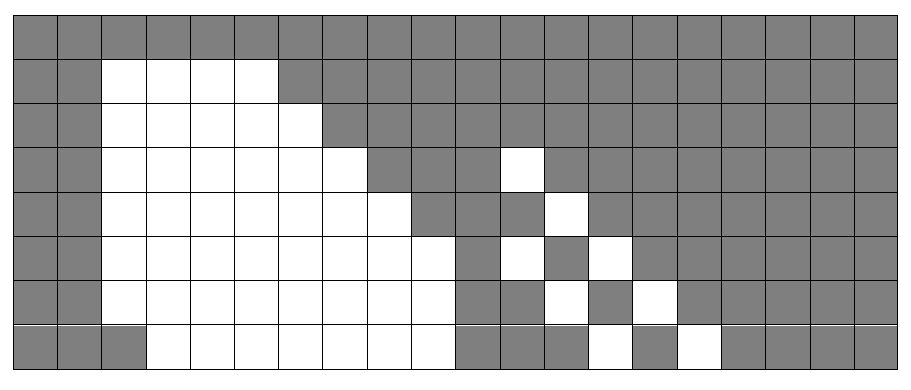
عملگر باز: ابتدا سایش و سپس گسترش را اعمال می‌کنیم. حاصل عملیات سایش:



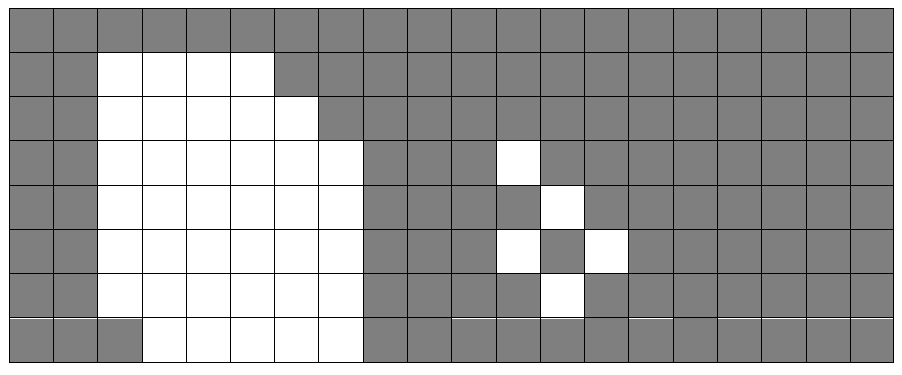
حال روی حاصل عملیات سایش، عملیات گسترش میزنیم که در نتیجه حاصل عملیات باز به شکل زیر بدست می‌آید:



عملگر بسته: ابتدا عملگر گسترش و سپس عملگر سایش را اعمال می‌کنیم. حاصل عملگر گسترش:



اکنون روی تصویر حاصل از گسترش، عملیات سایش را اعمال می‌کنیم.

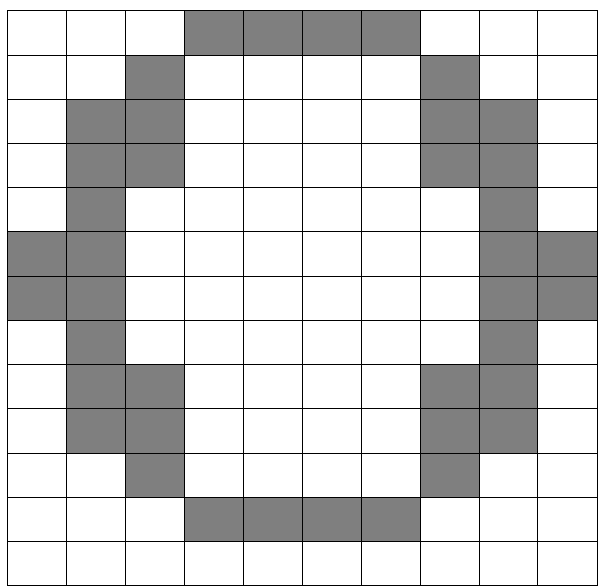


برای این قسمت مجبور بودم از padding استفاده کنم و نوع padding رو reflect انتخاب کردم.

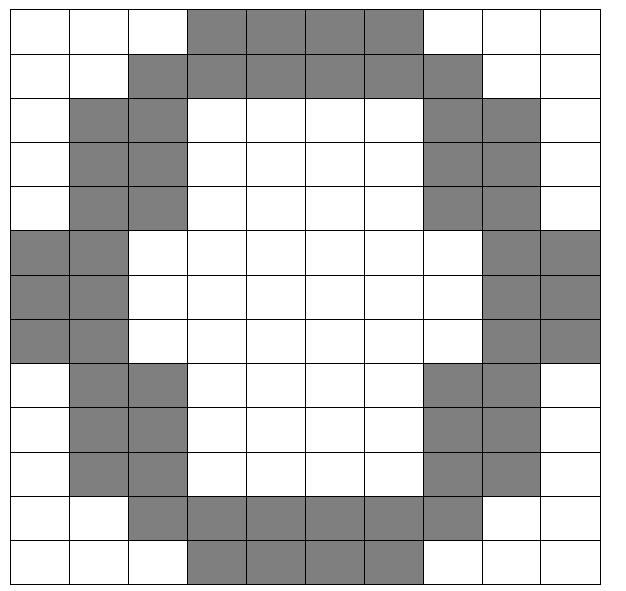
نکته: در این سوال پیکسل های سفید را 1 و پیکسل های سیاه را 0 گرفتم.

5- با کرنل پایین سمت چپ عملیات باز میزنیم. سیاه‌ها را یک و سفید ها را صفر میگیریم.

برای عملیات باز ابتدا عملیات سایش را با کرنل انجام داده که حاصل به صورت زیر میشود.



سپس این تصویر حاصل را با همان کرنل گسترش می‌دهیم. البته باید ابتدا کرنل را 180 درجه بچرخانیم.



تصویر حاصل به گونه‌ای است که خط وسط حذف شده است. اینکه باید با کرنل پایین سمت چپ استفاده کنم و عملیات باز بزنم، تجربی بدست آمد و روش خاصی نبود. برای بقیه کرنل ها نتوانستم عملیاتی که باعث حذف خط وسط و باقی ماندن عدد صفر شود، پیدا کنم.

ب)

برای بدست آوردن مرز از چهار کرنل استفاده کردم:

- حالتی که سمت راست یک سفید، سیاه باشد.

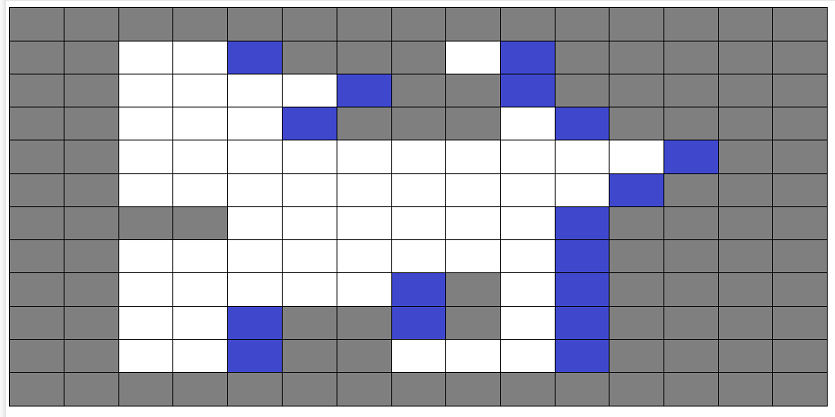
- حالتی که سمت چپ یک سفید، سیاه باشد.

- حالتی که بالای یک سفید، سیاه باشد.

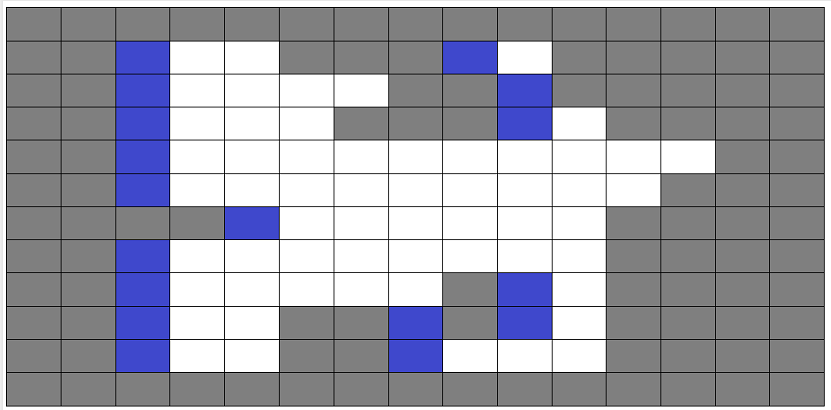
- حالتی که پایین یک سفید، سیاه باشد.

لنگر همه‌ی این کرنل‌ها روی رنگ سفید است. خانه هایی که در مراحل میانی به رنگ آبی هستند، نشان‌دهنده مرزهایی هستند که در آن مرحله بدست آمده‌اند، در انتها همه این خانه‌های آبی را باید اجتماع بگیریم تا جواب نهایی بدست آید. در این سوال خانه های سفید را 1 و خانه های سیاه را 0 گرفتم.

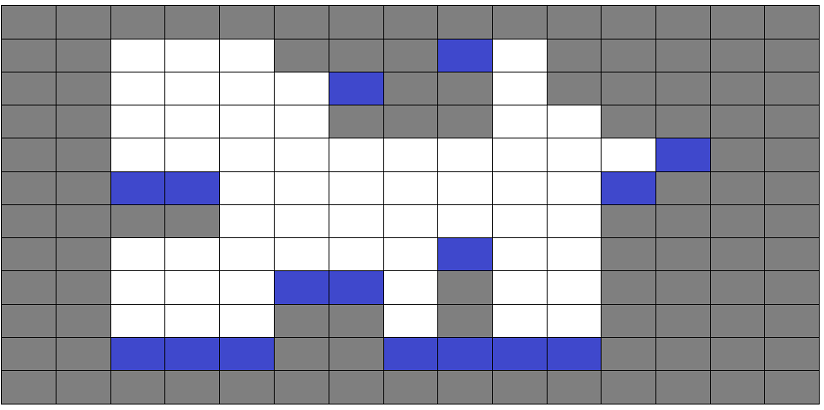




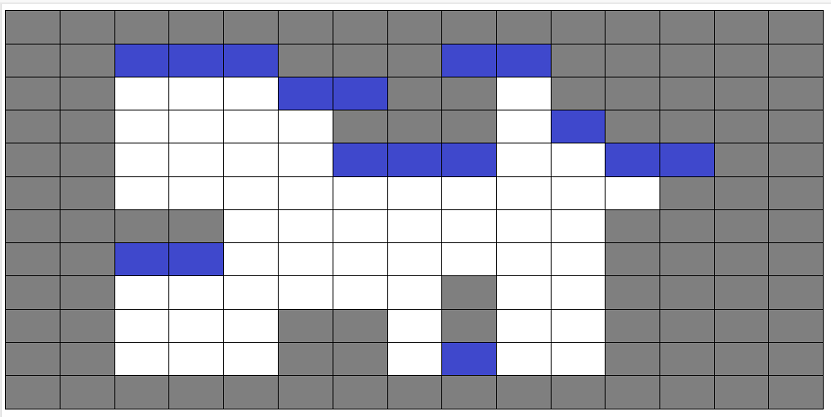




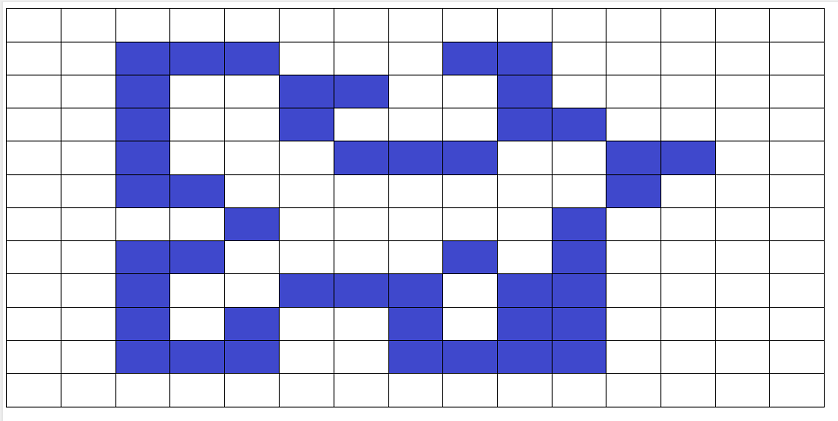






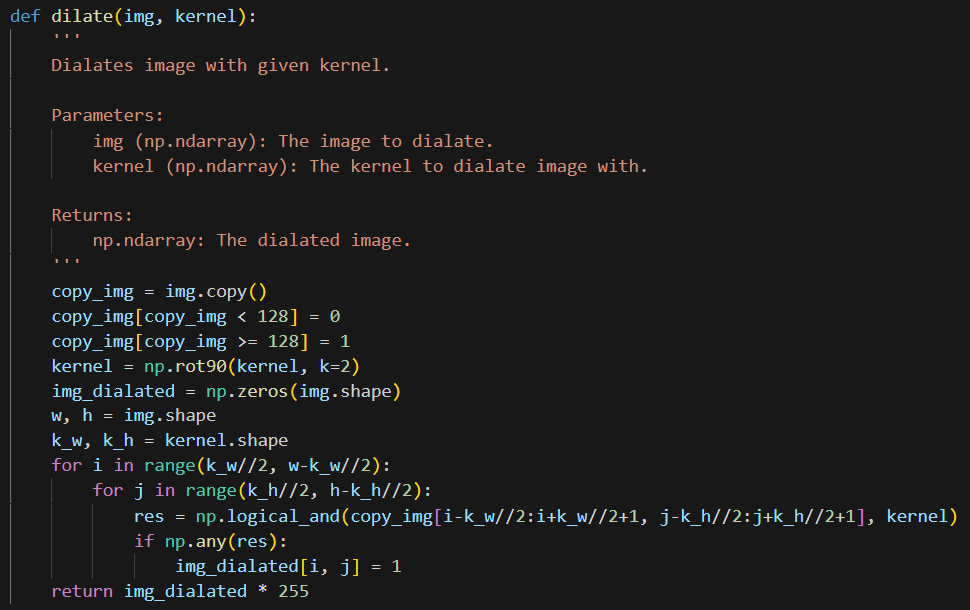


حال اجتماع همه نقاط آبی به شکل زیر خواهد بود:



6-

الف)

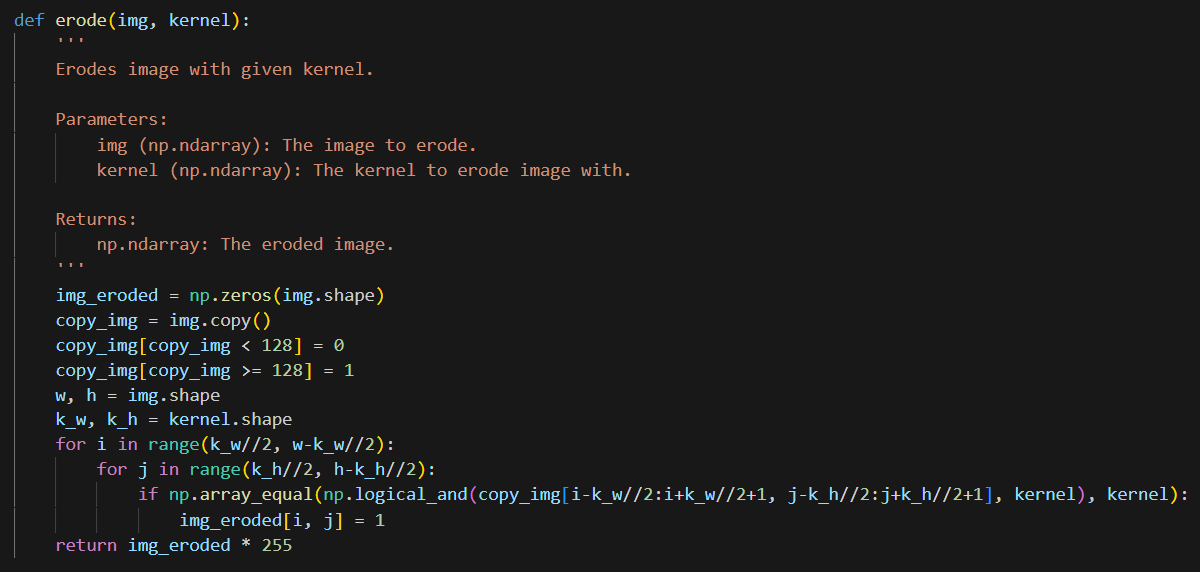


توضیحات تابع dilate:

در این تابع ابتدا رنگ پیکسلهای تصویر را باینری کردم، بیش از 128 را برابر با 1 و کمتر را برابر با 0 گرفتم. سپس کرنل یا عنصر ساختاری را با استفاده از تابع np.rot90 180 درجه چرخاندم، پارامتر دوم این تابع نشان‌دهنده ضریبی است که در 90 ضرب میشود. ابتدا همه مقادیر تصویر گسترش یافته را صفر میگیریم. حال کرنل را روی تصویر لغزانده و با استفاده از تابع np.logical\_and کرنل را با آن قسمت از عکس که روی آن قرار گرفته، and میکنیم، حاصل ماتریسی است که در خانه‌هایی که کرنل مقدار 1 داشت، مقدار خودش را میگیرد و در بقیه نقاط صفر میشود. سپس با استفاده از تابع np.any که بررسی میکند ماتریس ورودی حداقل یک عدد غیر صفر که اینجا 1 است داشته باشد. اگر این اتفاق بیفتد، یعنی آن قسمت از تصویر حداقل یدونه 1 در خانه هایی که کرنل 1 بود دارد پس برای این خانه در تصویر گسترش یافته عدد 1 یعنی سفید را قرار میدهیم. نکته ای که هست تصاویر ما (هواپیما و گاو و ...) سیاه هستند و بکگراند سفید است، در نتیجه عملیات dilate همانطور که انتظار میرود، قسمت های سفید را گسترش داده و در نتیجه قسمتهای سیاه کوچک میشود و در نتیجه عکس شی ما کوچکتر میشود.

خروجی dilate:



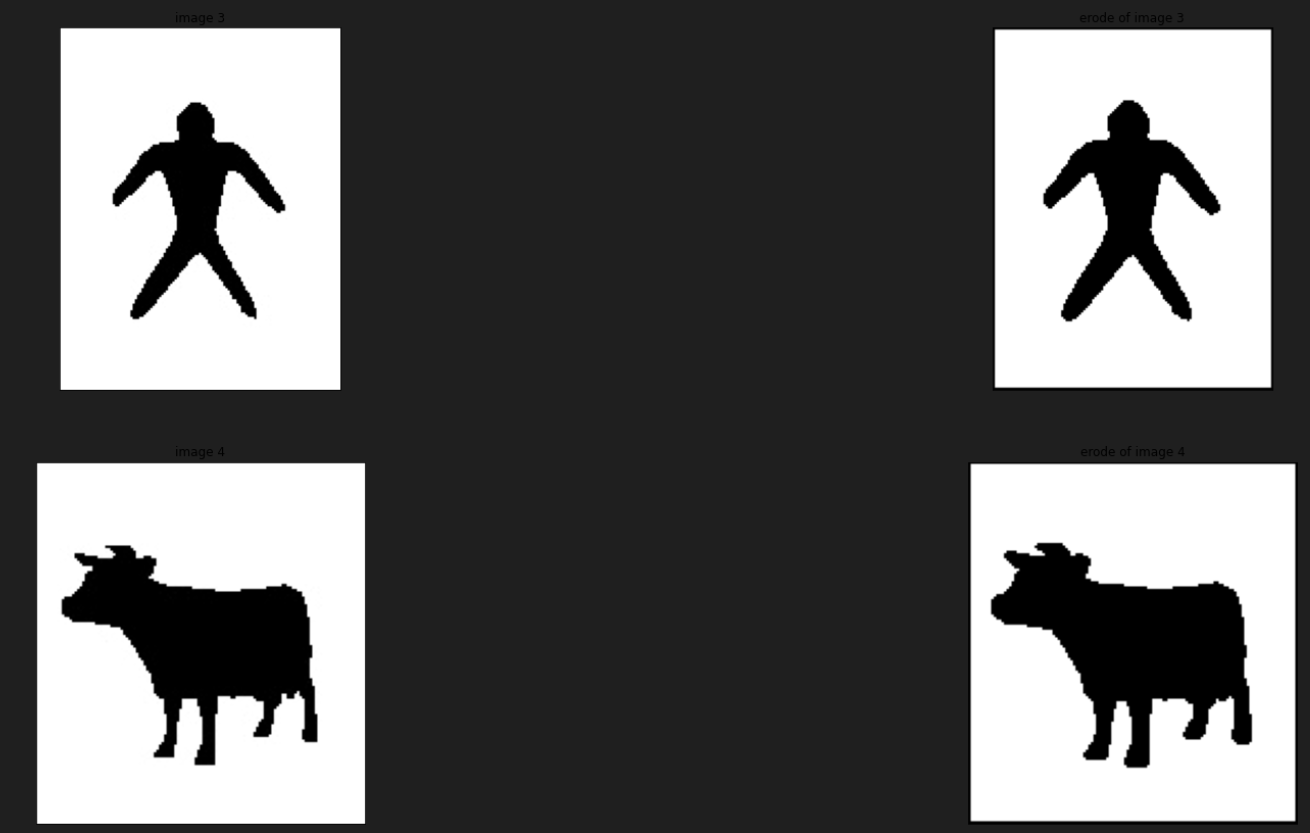


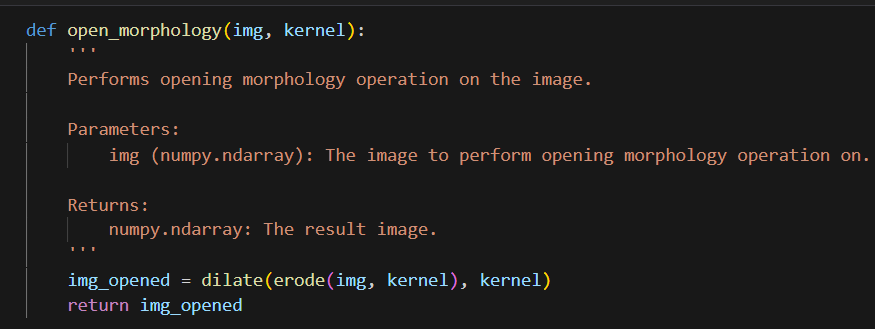
توضیحات تابع erode:

همه چیز همانند تابع dilate به غیر از اینکه کرنل را نمیچرخانیم و اینکه در قسمت بررسی شرط سفید شدن هر خانه به شکل زیر عمل میکنیم:

با استفاده از تابع np.logical\_and کرنل را با آن قسمت از تصویر and کرده و با استفاده از تابع np.array\_equal بررسی میکنیم حاصل این and شدن با کرنل، با خود کرنل برابر شود، زیرا در آن صورت یعنی آن قسمت از تصویر در همه خانه هایی که کرنل 1 بود ، مقدار 1 دارد. در آخر تصویر را سایش یافته را در 255 ضرب کردم که تصویر دارای مقادیر 0 و 255 باشد.

حاصل عملیات سایش:





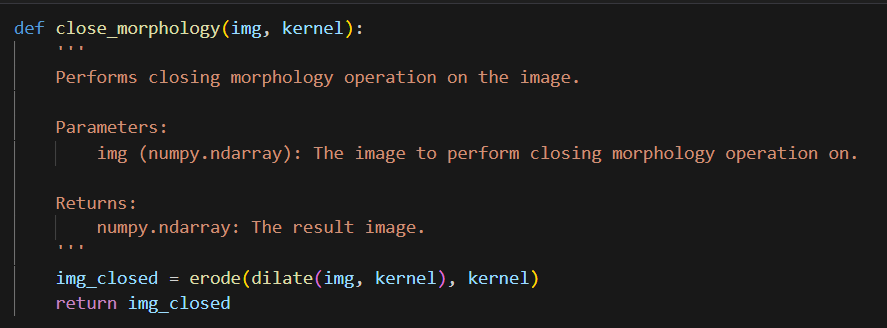
توضیحات تابع open\_morphology:

ابتدا سایش و روی حاصل سایش گسترش میزنیم.

خروجی عملگر باز:



تغییر خاصی مشاهده نشد چون قسمت سیاه و سفید تصویر کامل جدا هستند. اگر کرنل را خیلی بزرگ بگیریم، میتوانیم دست را کلا حذف کنیم.



توضیحات تابع close\_morphology:

ابتدا گسترش و روی تصویر حاصل سایش میزنیم.

خروجی عملگر بسته:



خروجی باقی عکسها هم در پوشه تمرین وجود دارند.

ب)

استفاده از توابع آماده openCv:

تابع dilate:



پارامتر اول عکس و پارامتر دوم کرنل است که باید به تایپ np.uint8 میبود وگرنه خطا میداد. پارامتر iterations تعداد تکرار عملیات را نشان میدهد، ما چون فقط یکبار میخواستیم گسترش را اعمال کنیم، عدد 1 را قرار دادیم.

تابع erode:



همه چیز مشابه dilate

تابع open:



پارامتر اول عکس، پارامتر دوم، نوع عملیات که اینجا عملگر باز است و پارامتر سوم کرنل و پارامتر چهارم همان iterations که در بالا گفتیم.

تابع close:



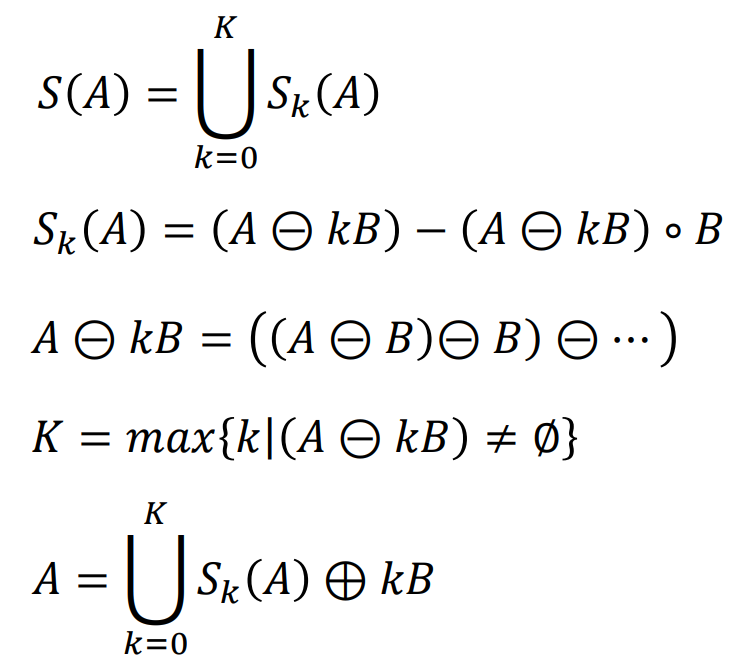
همه پارامترها همانند تابع open

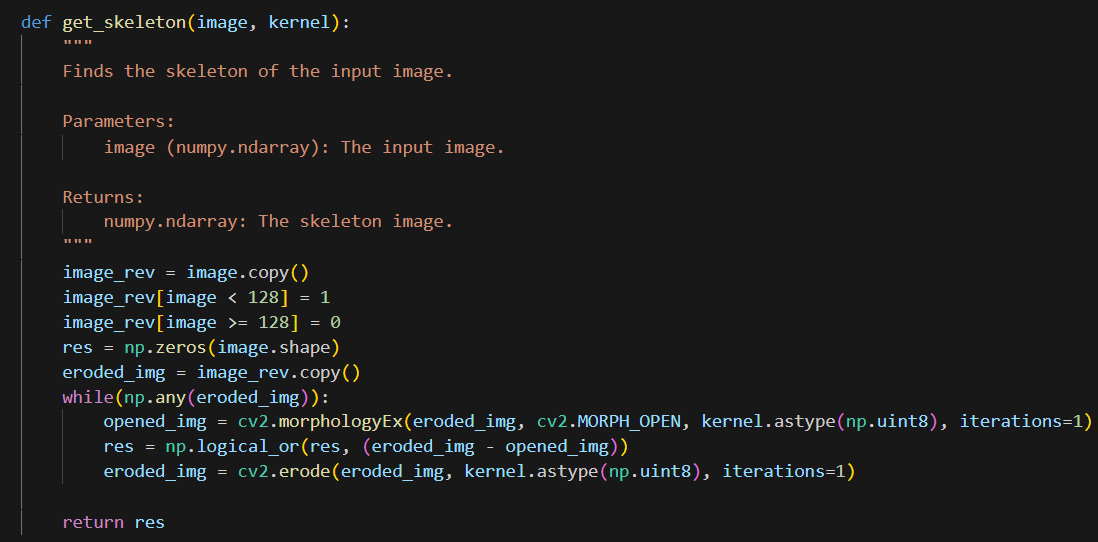
پارامتر دوم یعنی میخواهیم عملگر بسته را اعمال کنیم.

خروجی توابع آماده دقیقا مشابه آنچه پیاده سازی کردم بود و فایل عکسهای حاصل در پوشه تمرین قرار دارد.

ج)

برای بدست آوردن اسکلتون از فرمول زیر استفاده میکنیم:





توضیحات تابع get\_skeleton:

ابتدا مقادیر تصویر را صفر و یکی کردم و این کار را به گونه‌ای انجام دادم که نقاط سیاه، سفید و نقاط سفید، سیاه شوند. این کار را برای این کردم که شی ما قسمت سفید تصویر باشد تا کار با آن و فرمولی که بالا ذکر شد، راحت تر باشد.

سپس تصویر اسکلتون را با مقادیر اولیه صفر مقداردهی کردم و در res ریختم. همچنین یک متغیر eroded\_img تعریف کردم که در هر بار erode کردن از آن استفاده میکنم، تا جایی این سایش پیش میرود که دیگر پیکسل سفیدی در تصویر نماند. برای شرط while هم همین شرط داشتن حداقل یک پیکسل روشن را بررسی کرده و به یک مرحله جلوتر میروم. روی تصویر سایش یافته با استفاده از کرنل عمل باز زده و در متغیر opened\_img میریزم و سپس res را با استفاده از تابع np.logical\_or با حاصل کم شدن تصویر سایش یافته با تصویر سایش یافته منهای تصویر حاصل عملیات باز، or میکنم و در res میریزم، این or کردن همان کار اجتماع را در فرمول انجام میدهد. در آخر حلقه نیز، یک سایش دیگر روی eroded\_img میزنم تا یک مرحله به جلو برویم، هر موقع این سایش‌ها به قدری عکس را کوچک کند که چیزی از شی نماند، الگوریتم یافتن اسکلتون به پایان میرسد.

خروجی حاصل به شکل زیر بود:



