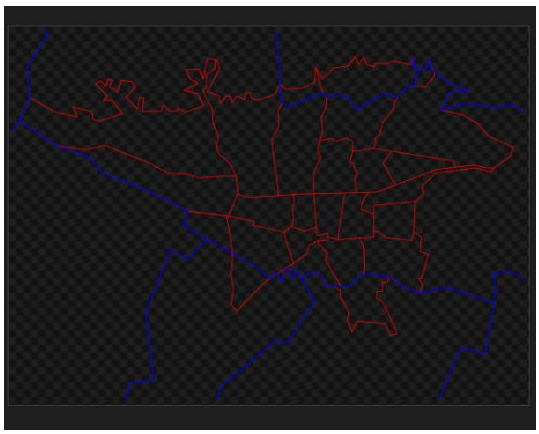
رنگ آمیزی نقشه با استفاده از الگوریتم رنگ آمیزی گراف

در این پروژه با کمک توابع opencv یک عکس با زمینه سیاه و خطوط مرزی مانند نقشه ی تهران از کاربر میگیریم و پس از آن نواحی آن را مشخص میکنیم و به ماتریس مجاورت نواحی تبدیل کنیم و سپس به کمک الگوریتم رنگ آمیزی گراف عکس رنگ شده ناحیه های مختلف را نشان میدهیم.

سه فایل main.py, ImageProcessing.py, Graph.py در این پروژه وجود دارند که :

در main عکس را میگیریم و به نمونه ای از ImageProcessing (IP) میدهیم سپس ماتریس مجاورت بدست آمده را برای مشخص کردن رنگ ناحیه ها به نمونه ای از Graph میدهیم و پس از مشخص شدن رنگ ناحیه ها آنرا به ImageProcessing بر میگردانیم تا عکس را رنگ آمیزی کند.

در فایل main ابتدا یک parser برای گرفتن آرگومان تعریف میکنیم که به صورت پیش فرض مقدار input برابر نام عکس تهران میباشد. (teh.png)



برای بهتر شدن خروجی اعداد و حروف روی نقشه را از عکس ورودی پاک کردی ایم چون دارای نواحی بسته بودند و هرکدام از آنها رنگ می شدند.

پس از آن یک نمونه از کلاس IP می سازیم و آدرس عکس را به آن میدهیم.

ماتریس مجاورت را از متد adjacenct_matrix() آن نمونه میگیریم و یک نمونه از کلاس Graph میسازیم و طول ماتریس مجاورت و خود آن ماتریس را به آن میدهیم.

آرایه ای از رنگ ها که شماره هر ناحیه در آن آرایه نشان دهنده اندیس رنگ آن است را از coloring() میگیریم و در انتها آنرا به متد show() میدهیم تا عکس را رنگ آمویزی کند.

در سازنده کلاس IP آدرس عکس را میگیریم و سپس به وسیله cv2 عکس را میخوانیم و در src_img ذخیره میکنیم.

عکس را به grayscale تبدیل میکنیم که تشخیص زمینه و مرز ها راحت تر بشود

سپس عکس gray_img را به عکس سیاه و سفید تبدیل میکنیم با این شرط که اگر مقدار رنگ خاکستری آن بیشتر از 10 بود به سفید یعنی 255 در غیر این صورت به سیاه 0 تبدیل شود (اعداد در grayscale)

سپس عکس سیاه و سفید حاصل از عملیات بالا را که زمینه سیاه و مرز ها سفید می باشد را برعکس می کنیم و در bw ذخیره میکنم (یعنی مرز ها سیاه و زمینه سفید می شود)

به کمک تابع connectedComponents از opencv عکس bw را به روش component labeling algorithm لیبل بندی میکنیم و تعداد لیبل ها را در num_labels و آرایه ای از شماره لیبل ها متناظر با پیکسل های عکس اصلی را در labels ذخیره میکنیم.

(شماره لیبل مرز ها برابر 0 و پس زمینه یک است و بقیه نواحی از 2 شروع می شوند)

Init_colors شامل چند رنگ متنوع برای رنگ آمیزی نهایی نقشه میباشد.

```
7  def __init__(self, img_dir):
8      self.img_dir = img_dir
9      self.src_img = cv2.imread(cv2.samples.findFile(img_dir))
10     gray_img = cv2.cvtColor(self.src_img, cv2.COLOR_RGB2GRAY)
11     bw = cv2.threshold(gray_img, 10, 255, cv2.THRESH_BINARY)[1]
12     self.bw = cv2.bitwise_not(bw)
13     self.num_labels, self.labels = cv2.connectedComponents(self.bw)
14     self.init_colors = {1: [255, 161, 0],
15                         2: [161, 0, 255],
16                         3: [0, 255, 161],
17                         4: [0, 235, 255],
18                         5: [128, 128, 128],
19                         6: [0, 0, 0]}
20     print("Done!")
```

متد adjacency_matrix برای بدست آوردن ماتریس مجاورت لیبل های بدست آمده از سازنده کلاس میباشد.

در ابتدا طول و عرض عکس را بدست می آوریم و ماتریس مجاورت را در matrix ذخیره میکنیم و آنرا مقدار دهی اولیه میکنیم (0)

برای یافتن لیبل های مجاور به این صورت عمل میکنیم که:

یک جهت برای پیمایش و بررسی پیکسل ها در نظر میگیریم و در آن جهت پیش میرویم.

شماره لیبلی که در آن بوده ایم در last ذخیره میکنیم

اگر شماره لیبلی که در آن هستیم صفر است یعنی بر روی مرز قرار داریم پس c را یکی افزایش میدهیم

(مقدار c برابر مقدار خانه های مرزی پشت هم هستند و اگر به خانه غیر از مرز رسیدیم c صفر میشود)

اگر شماره لیبل پیکسلی که در آن هستیم برابر last است هیچ کاری لازم نیس بکنیم فقط c را صفر میکنیم

اگر شرایط بالا بر قرار نباشد یعنی از یک لیبل وارد لیبل دیگری شده ایم پس می توانند همسایه باشند

(به صورت پیشفرض فرض کرده ایم عرض مرز ها از 5 بیشتر نیستند)

پس اگر $c > 5$ بود یعنی مشکلی پیش آمده و نمیتوان گفت که لیبل پیکسلی که در آن هستیم مجاور last می باشد بنابر این تنها مقدار لیبل پیکسل حاضر را در last قرار میدهیم

اگر $c < 6$ بود یعنی از مرز گذشته ایم و از لیبل last به لیبل حاضر رسیده ایم پس این دو لیبل همسایه هستند و در matrix در سطر و ستون و ستون و سطر شماره لیبل ها ذخیره میکنم

الگوریتم بالا را برای تمام پیکسل ها و در دو جهت عمودی و افقی انجام میدهیم تا ماتریس مجاورت کامل شود

```
28        for i in range(height):
29            last = 1
30            c = 0
31            for j in range(width):
32                if self.labels[i, j] == 0:
33                    c = c + 1
34                    continue
35                if self.labels[i, j] == last:
36                    c = 0
37                    continue
38                if c > 5:
39                    last = self.labels[i, j]
40                    c = 0
41                    continue
42                label_index = self.labels[i, j] - 2
43                last_index = last - 2
44                c = 0
45                if last_index != -1 and label_index != -1 and matrix[last_index][label_index] != 1:
46                    matrix[last_index][label_index] = 1
47                    matrix[label_index][last_index] = 1
48                last = self.labels[i, j]
```

```
50        for i in range(width):
51            last = 1
52            c = 0
53            for j in range(height):
54                if self.labels[j, i] == 0:
55                    c = c + 1
56                    continue
57                if self.labels[j, i] == last:
58                    c = 0
59                    continue
60                if c > 5:
61                    last = self.labels[j, i]
62                    c = 0
63                    continue
64                label_index = self.labels[j, i] - 2
65                last_index = last - 2
66                c = 0
67                if last_index != -1 and label_index != -1 and matrix[last_index][label_index] != 1:
68                    matrix[last_index][label_index] = 1
69                    matrix[label_index][last_index] = 1
70                last = self.labels[j, i]
```

در ابتدای متد show() عکس اصلی را چاپ میکنیم src_img .

متد show() رنگ ناحیه هارا میگیرد و دو شماره رنگ 5 و 6 که شماره رنگ های مرز ها و زمینه میباشند (رنگ ها با اندیس شماره یشان در init_colors ذخیره شده اند) را به درون آرایه رنگ های حاصل از حل مسعله رنگ آمیزی گراف اضافه میکنیم.

اندیس صفر یعنی لیبل صفر که مرز می باشد برابر رنگ ششم یعنی 0و0و0 یا همان سیاه می شود

اندیس یک یعنی لیبل یک که زمینه میباشد برابر رنگ پنجم 128و128و128 یا همان خاکستری میشود

بقیه اندیس ها هم همان شماره رنگشان در init_colors می شوند.

چون گراف مسطح است و قضیه چهار رنگ وجود حداکثر چهار رنگ را تضمین میکند پس علاوه بر دو رنگ مرز و زمینه چهار رنگ دیگر برای ناحیه ها در init_colors قرار داده ایم.

سپس آرایه ای میسازیم که رنگ های توضیحات بالا را در یک آرایه به اندازه خود عکس و متناظر با شماره لیبل آن قرار میدهد و آن عکس را چاپ میکنیم.

و در آخر هم شماره هر لیبل را به یکی از اعداد بازه 179 – 0 مپ میکنیم و به وسیله آن یک رنگ hue میسازیم و در انتها هم آن آرایه را به rgb تبدیل میکنیم و نمایش میدهیم در این عکس هر ناحیه دارای رنگی متفاوت از بقیه نواحی میباشد.

```
76      def show(self, colors):
77          cv2.imshow('Binary Image', self.src_img)
78          cv2.waitKey()
79
80          colors.insert(0, 5)
81          colors.insert(0, 6)
82          colored = np.array([[self.init_colors[colors[j]] for j in i] for i in self.labels], dtype=np.uint8)
83
84          cv2.imshow('Colored Image', colored)
85          cv2.waitKey()
86
87          label_hue = np.uint8(179 * self.labels / np.max(self.labels))
88          blank_ch = 255 * np.ones_like(label_hue)
89          labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
90
91          labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)
92          labeled_img[label_hue == 0] = 0
93
94          cv2.imshow("Component Labeling", cv2.cvtColor(labeled_img, cv2.COLOR_BGR2RGB))
95          cv2.waitKey()
```

در سازنده کلاس Graph تعداد راس ها و ماتریس مجاورت را میگیریم و آرایه ای متناظر با اندیس هر راس در نظر میگیریم که شماره رنگش در آن خانه می باشد. در ابتدا همه خانه ها صفر است.

متد coloring() در این متد رنگ آمیزی را انجام میدهیم و شمره رنگ را چاپ میکنیم و آرایه رنگ هارا بر میگردانیم

اگر تعداد راس ها یک بود که با یک رنگ میتوان رنگ آمیزی کرد (رنگ 1)

اگر بیشتر از یک بود به روش عقب گرد شماره رنگ را به متد backtrack میدهیم و اگر با آن تعداد رنگ توانستیم رنگ کنیم آرایه رنگ هارا بر میگردانیم در غیر این صورت False را بر میگردانیم که نشان دهنده اشتباه بودن ماتریس مجاورت است.

در متد backtrack شماره راسی که در آن قرار داریم تعداد رنگ ها و آرایه رنگ ها و شماره ها را میگیریم

اگر راسی که در آن قرار داریم برابر تعداد راس ها بود یعنی با موفقیت به ته درخت رنگ آمیزی رسیده ایم بنابر این True را بر میگردانیم و رنگ ها در همان آرایه ای که گرفته ایم قرار دارند

در غیر این صورت برای شماره راس داده شده تمام رنگ هارا به تعداد رنگ داده شده امتحان میکنیم

رنگ i و راس v :

چک میکنیم ببینیم این رنگ آمیزی امکان پذیر هست یا نه به کمک valid_color

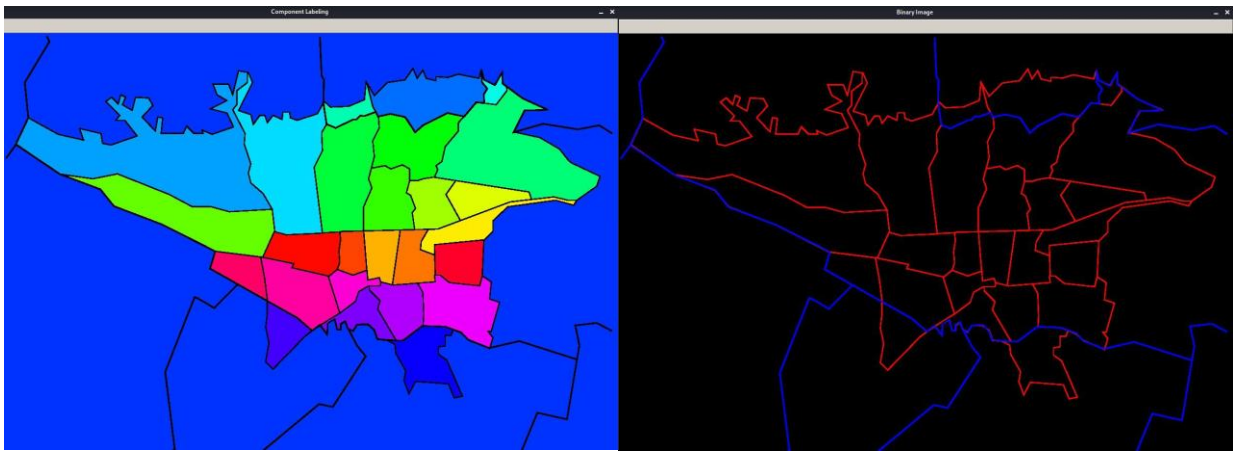
برای تمام راس های مجاور v یعنی تمام اندیس هایی که در سطر v در ماتریس مجاورت دارای مقدار یک می باشند اگر رنگ آن مقدار هم i بود یعنی رنگ کردن v هم به رنگ i هم امکان پذیر نیست و False بر میگردانیم و باید رنگ دیگری امتحان کنیم (رنگ i برای v امیدبخش نیست pormissing!)

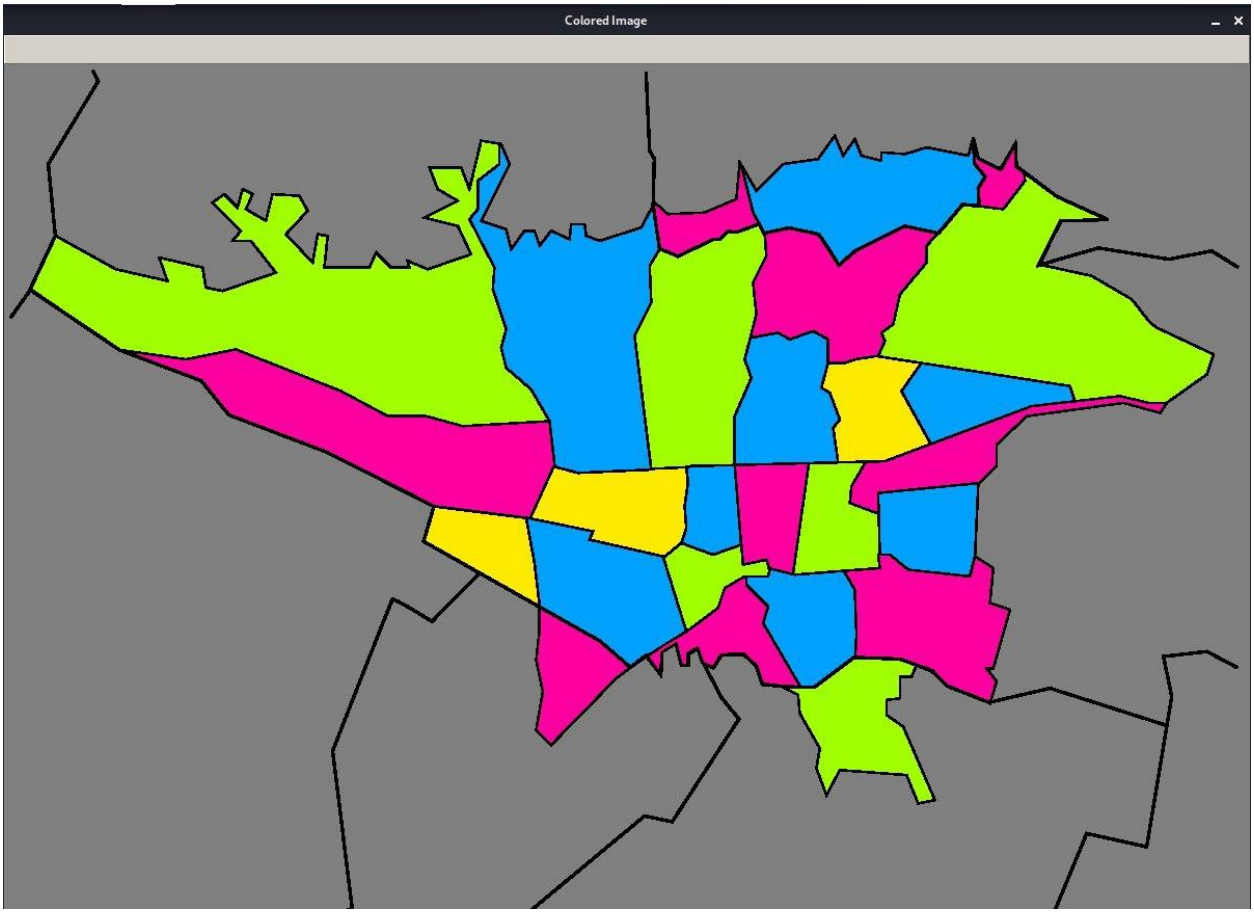
اگر با همه مجاور هایش رنگ متفاوتی داشت آنگاه True را بر میگردانیم و در تابع backtrack با صدا زدن خود تابع برای راس های بعدی زیر درخت هایش را برسی میکنیم ...

```
27  backtrack(self, v, no_colors, colors):
28  if self.no_v == v:
29      return True
30  for i in range(1, no_colors + 1):
31      if self.valid_color(v, i, colors):
32          colors[v] = i
33          if self.backtrack(v + 1, no_colors, colors):
34              return True
35  return False
36
37  valid_color(self, colored_vertex, vertex_color, colors):
38  for i in range(self.no_v):
39      if self.matrix[colored_vertex][i] == 1 and vertex_color == colors[i]:
40          return False
41  return True
```

در ادامه برنامه را برای نقشه ی تهران و ایالات متحده اجرا میکنیم :

```
(venv) tr0dd@tr0dd:~/Documents/github/opencv$ python3.8 main.py
Processing Input Image ... Done!
Calculating Adjacency Matrix ... Done!
26  Regions
Graph Coloring ... 4 Colors
Press Enter To Continue.
```





```
(venv) tr0dd@tr0dd:~/Documents/github/opencv$ python3.8 main.py --input usa.png
Processing Input Image ... Done!
Calculating Adjacency Matrix ... Done!
79  Regions
Graph Coloring ... 4 Colors
Press a Key To Continue.
```