Rozwiązania wybranych zadań z list

Piotr Mazur

26 listopada 2025

Spis treści

1 lista 5, zadanie 2

Treść zadania: Napisz funkcjc `bright _pixel(image)` , ktora dla trójwymiarowej tablicy `image` reprezentującej kolorowy obraz (w sposób opisany na wykładzie) sprawdza, czy w jego lewej górnej ćwiartce znajduje się co najmniej jeden piksel, dla którego co najmniej jedna ze składowych RGB ma wartość większą, niż 200. Cwiartkę tę rozumiemy jako obrazek powstaly z pierwszej połowy wierszy i pierwszej połowy kolumn danego obrazu (odpowiednio zaokrqglajqc liczbę wierszy lub kolumn w dół). Funkcja powinna zwracać `True`, jeżeli taki piksel istnieje i `False` w przeciwnym wypadku.

rozpatrywany obrazek

1.1 Krok 1

importowanie potrzebnych bibliotek, tj.:

1. `imageio` do zaimportowania obrazu jako trójwymiarowej tablicy pikseli.
2. `matplotlib.pyplot` do wyświetlenia zaimportowanego obrazu.
3. `numpy` do działań na tablicy obrazu i przechowywania informacji o tej tablicy.

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

1.2 Krok 2

zdefiniowanie obrazka (gdzie „ ... ” symbolizuje link do obrazka lub ścieżkę na dysku). `print(image.shape` jako pomocnicze wyświetlenie wymiarów tablicy.

```
image=imageio.v2.imread(...)
print(image.shape)

plt.imshow(image)
plt.show()
```

1.3 Krok 3

zdefiniowanie wymiarów tablicy osobno:

1. `h` – ilość wierszy w tablicy (wysokość obrazu w pikselach)
2. `w` – ilość kolumn w tablicy (szerokość obrazu w pikselach)
3. `d` – ilość wymiarów tablicy (ilość kanałów obrazu. Standardowo: R,G,B)

```
h, w, d = image.shape
```

1.4 Krok 4

Separacja konkretnej części tablicy, czyli wycięcie ćwiartki obrazu.

```
quarter = image[0: int(h/2), 0: int(w/2), :]
```

1.5 Krok 5

Warunek sprawdzający czy którykolwiek element tablicy `quarter` ma wartość numeryczną większą od 200.

```
np.max(quarter) > 200
```

1.6 Krok 6

Połączenie fragmentów kodu napisanych w punktach 1.1, 1.2, 1.3, 1.4 1.5 w spójną całość:

```
def bright_pixel(image):
    h, w, d = image.shape
    quarter = image[0: (h//2), 0: int(w/2), :]
    return np.max(quarter) > 200

print(bright_pixel(image))
```

2 lista 2, zadanie 2

Treść zadania: Rozważmy prostokątną planszę o 10 wierszach i 10 kolumnach, której każde pole opisane jest przez parę współrzędnych: numer wiersza oraz numer kolumny (dwie liczby od 1 do 10). Napisz program, który dla liczb i, j (gdzie $1 \leqslant i, j \leqslant 10$) wypisze wszystkie pola (tzn. ich współrzędne) sąsiadujące z polem w i-tym wierszu i j-tej kolumnie, oraz to pole. Pola sąsiadują ze sobą bokami oraz na ukos. Przykładowo, dla pola o współrzędnych $(1, 2)$ poprawny wynik to pola o współrzędnych $(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3)$. Kolejność wypisywania pól może być dowolna, o ile zostaną wypisane odpowiednie pola.

$$\begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} & d_{16} & d_{17} & d_{18} & d_{19} & d_{1,10} \\ d_{21} & d_{22} & d_{23} & d_{24} & d_{25} & d_{26} & d_{27} & d_{28} & d_{29} & d_{2,10} \\ d_{31} & d_{32} & d_{33} & d_{34} & d_{35} & d_{36} & d_{37} & d_{38} & d_{39} & d_{3,10} \\ d_{41} & d_{42} & d_{43} & d_{44} & d_{45} & d_{46} & d_{47} & d_{48} & d_{49} & d_{4,10} \\ d_{51} & d_{52} & d_{53} & d_{54} & d_{55} & d_{56} & d_{57} & d_{58} & d_{59} & d_{5,10} \\ d_{61} & d_{62} & d_{63} & d_{64} & d_{65} & d_{66} & d_{67} & d_{68} & d_{69} & d_{6,10} \\ d_{71} & d_{72} & d_{73} & d_{74} & d_{75} & d_{76} & d_{77} & d_{78} & d_{79} & d_{7,10} \\ d_{81} & d_{82} & d_{83} & d_{84} & d_{85} & d_{86} & d_{87} & d_{88} & d_{89} & d_{8,10} \\ d_{91} & d_{92} & d_{93} & d_{94} & d_{95} & d_{96} & d_{97} & d_{98} & d_{99} & d_{9,10} \\ d_{10,1} & d_{10,2} & d_{10,3} & d_{10,4} & d_{10,5} & d_{10,6} & d_{10,7} & d_{10,8} & d_{10,9} & d_{10,10} \end{bmatrix}$$

Opisana plansza, przedstawiona jako macierz:

d_{11}	d_{12}	d_{13}
d_{21}	d_{22}	d_{23}

sąsiedztwo elementu d_{12}

2.1 Krok 1

Pobranie danych od użytkownika. Oczekujemy numeru i wiersza oraz numeru j kolumny rozpatrywanego elementu:

```
i=int(input("i wiersz:"))
j=int(input("j kolumna:"))
```

2.2 Krok 2

Sprawdzenie sensowności wprowadzonych danych poprzez zdefiniowanie warunku logicznego.

```
if 10 >= i >= 1 and 10 >= j >= 1:

else:
  print("Podano bledne dane")
```

2.3 Krok 3

Zdefiniowanie listy wszystkich indeksów macierzy tj. $(1,1),(1,2),(1,3)\cdots$ Lista powinna zawierać 100 elementów.

```
p=[(m,n) for m in range (1,10+1) for n in range(1,10+1)]
```

2.4 Krok 4

1. Zdefiniowanie pętli, która przebiegnie po wszystkich elementach z listy p.

2. Zdefiniowanie warunku logicznego, który sprawi, że pętla wybierze te elementy d_{mn}, których numer wiersza sąsiaduje z wprowadzonym numerem tj. $m = i-1 \vee m = i+1$

3. Zdefiniowanie warunku logicznego, który sprawi, że pętla wybierze te elementy d_{mn}, których numer kolumny sąsiaduje z wprowadzonym numerem tj. $n = j-1 \vee n = j+1$

4. koniunkcja warunków logicznych z punktów 2 i 3 pozwoli wydrukować każdy element sąsiadujący z tym, który użytkownik wprowadził na początku.

```
for d in p:
    if (d[0] == i-1 or d[0] == i or d[0] == i+1) and
    (d[1] == j-1 or d[1] == j or d[1] == j+1):
        print(f"{p}")
```

2.5 Krok 5

Połączenie fragmentów kodu napisanych w punktach 2.1, 2.2, 2.3, 2.4 w spójną całość:

```
i=int(input("i wiersz:"))
j=int(input("j kolumna:"))

if 10 >= i >= 1 and 10 >= j >= 1:
    p=[(m,n) for m in range (1,10+1) for n in range(1,10+1)]

    for d in p:
        if (d[0] == i-1 or d[0] == i or d[0] == i+1) and
        (d[1] == j-1 or d[1] == j or d[1] == j+1):
            print(f"{p}")
else:
  print("Podano bledne dane")
```