# Zadanie Neon – LOGIA 18 (2017/18), etap 3

## Treść zadania

W Turtlandii przygotowują neon do zawieszenia na dwóch słupach. Słupy stoją na kwadratowej siatce, a odległość pomiędzy dwoma sąsiednimi słupami w rzędzie oraz kolumnie wynosi 2. Neon musi być rozwieszony w jednym rzędzie lub kolumnie. Dział marketingu uzależnia wybór słupów od oceny zdefiniowanej jako suma ich wysokości i odległości między nimi. Napisz jednoparametrową funkcję neon, której wynikiem jest najwyższa możliwa ocena. Parametr jest listą list wysokości słupów. Rzędy są opisane kolejnymi podlistami. W każdym rzędzie oraz kolumnie jest od 2 do 500 słupów. Postaraj się, by na wynik funkcji nie trzeba było zbyt długo czekać.

| | | |
|---|---|---|
| 1 | 9 | 2 |
| 3 | 8 | 3 |
| 2 | 1 | 1 |

| | | | |
|---|---|---|---|
| 1 | 2 | 1 | 2 |
| 7 | 1 | 7 | 1 |
| 1 | 1 | 1 | 1 |
| 3 | 3 | 3 | 3 |

*Przykłady:*

wynikiem **neon([[1, 9, 2], [3, 8, 3], [2, 1, 1]])** jest **19** (19=9+8+2),

wynikiem **neon([[1, 2, 1, 2], [7, 1, 7, 1], [1, 1, 1, 1], [3, 3, 3, 3]])** jest **18** (18=7+7+2·2)

## Omówienie rozwiązania

Zadanie sprowadzimy do rozwiązania zadania wcześniej omawianego – neon z II etapu. Jeśli umiemy znaleźć najlepsze rozwiązanie dla jednego wiersza, to tym samym możemy je rozszerzyć na wiele wierszy. Wystarczy dokonać zamiany wierszy z kolumnami i znaleźć wartość maksymalną dla nowych wierszy. Rozwiązaniem zadania będzie maksimum z najlepszego rozwiązania w wierszach i kolumnach.

## Rozwiązanie w języku Python

Implementacja składa się z trzech funkcji – głównej `neon`, która znajduje wartość maksymalną w pionie i poziomie, pomocniczej – `neon_pom` – która wylicza wartość maksymalną w poziomie oraz `trans`, która dokonuje transpozycji wierszy i kolumn.

W funkcji `neon_pom` rozpatrujemy listę począwszy od drugiego elementu, w języku Python zapisujemy to następująco: `lista[1:]`, czyli nazwa zmiennej oraz w nawiasie kwadratowym dwa indeksy oddzielone dwukropkiem: pierwszy wskazujący od jakiej wartości zaczynamy (listy numerujemy od 0), drugi wskazujący do jakiej wartości. Przy czym, jeśli elementy listy mają być brane do końca, to wartość indeksu można pominąć. Warto też zwrócić uwagę, że w rozwiązaniu skorzystano z wbudowanej funkcji `max`, której wynikiem jest maksymalna wartość liczb podanych jako parametr.

W funkcji `trans` i `neon` skorzystano z list składanych, uproszczonego zapisu, w którym generowana jest iteracyjnie lista. Zmienna sterująca i liczba powtórzeń jest zdefiniowana po słowie kluczowym `for`, a przed nim element wyliczany przy każdym obrocie pętli.

```python
#szukanie w jednym wierszu
def neon_pom(lista):
    pom = lista[0]
    naj = 0
    for x in lista[1:]:
        pom = pom + 2
        naj = max(naj, pom + x)
        pom = max(pom, x)
    return naj

#transpozycja kolumn i wierszy
def trans(lista):
    kolumna = []
    for i in range(len(lista[0])):
        wiersz = [lista[j][i] for j in range(len(lista[0]))]
        kolumna.append(wiersz)
    return kolumna

# funkcja główna
def neon(lista):
    x = max([neon_pom(e) for e in lista])
    pom = trans(lista)
    y = max([neon_pom(e) for e in pom])
    return max(x, y)
```

## Testy

Testowanie rozwiązania zaczynamy od przypadków, których wynik można wyznaczyć przy pomocy kartki papieru. Potem przystępujemy do przypadków bardziej skomplikowanych. Należy zadbać o to, by znalazły się przykłady, gdzie optymalne rozwiązanie to ciąg w pionie lub poziomie. Poniżej przedstawiamy krótsze testy, zachęcamy by dłuższe wygenerować samodzielnie.

```
neon([[2, 10, 9], [6, 8, 3], [1, 7, 8]])
```
wynik: 21

```
neon([[1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 12, 12, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 6, 1, 1, 1, 9, 1, 1, 8, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```
wynik: 28

```
neon([[13, 10, 16, 16, 1, 2, 10, 2, 7, 6, 10, 20, 7, 12, 7, 13, 18, 9, 14, 12], [16, 17, 12, 14, 12, 12, 4, 13, 18, 18, 17, 4, 4, 10, 17, 14, 1, 11, 18, 17], [2, 14, 15, 7, 3, 7, 10, 3, 1, 4, 16, 6, 9, 2, 15, 4, 15, 17, 8, 11], [11, 9, 3, 13, 1, 6, 10, 3, 11, 14, 14, 7, 7, 6, 10, 17, 1, 9, 11, 8], [6, 20, 5, 6, 2, 15, 1, 6, 19, 2, 17, 9, 3, 2, 4, 19, 19, 8, 2, 8], [9, 14, 19, 10, 7, 18, 2, 8, 17, 14, 20, 13, 1, 17, 14, 11, 15, 14, 4, 6], [18, 19, 3, 6, 18, 1, 4, 19, 19, 17, 5, 15, 1, 16, 6, 8, 13, 17, 9, 16], [10, 14, 7, 2, 4, 2, 13, 6, 3, 1, 16, 8, 5, 12, 17, 16, 13, 12, 10, 13], [16, 13, 19, 8, 9, 17, 13, 1, 16, 3, 14, 12, 6, 17, 6, 20, 7, 10, 12, 3], [4, 20, 7, 14, 20, 12, 20, 14, 6, 19, 8, 14, 11, 10, 6, 13, 16, 14, 19, 17], [18, 13, 9, 14, 10, 2, 13, 13, 1, 16, 3, 12, 20, 2, 7, 7, 2, 17, 3, 2], [10, 5, 10, 18, 6, 1, 12, 20, 6, 5, 2, 9, 20, 3, 12, 10, 5, 9, 14, 2], [2, 13, 5, 14, 10, 18, 17, 15, 13, 8, 7, 7, 20, 2, 10, 4, 15, 18, 7, 6], [13, 17, 4, 17, 16, 14, 2, 20, 10, 6, 20, 11, 1, 10, 15, 20, 13, 11, 7, 7], [10, 10, 3, 10, 11, 16, 4, 18, 20, 10, 6, 6, 4, 1, 3, 2, 7, 4, 18, 12], [13, 6, 3, 1, 12, 14, 8, 17, 18, 1, 14, 2, 20, 14, 2, 11, 3, 8, 12, 12], [19, 12, 19, 18, 3, 18, 15, 3, 6, 19, 20, 7, 19, 4, 12, 18, 14, 2, 10, 7], [14, 14, 12, 2, 7, 11, 5, 6, 20, 8, 8, 8, 18, 18, 11, 5, 10, 20, 16, 6], [13, 19, 20, 9, 9, 7, 17, 4, 5, 6, 2, 7, 3, 13, 1, 2, 11, 13, 18, 12], [4, 13, 12, 11, 6, 11, 10, 3, 15, 19, 20, 4, 8, 1, 16, 16, 5, 8, 20, 2]])
```
wynik: 74

Przykład samodzielnego generowania testów:

```
1. from random import *
2. x = 20
3. test = [[randint(1,20) for x in range(x)] for y in range(x)]
```