

Julia Grzegorzewska, Wiktoria Fimińska

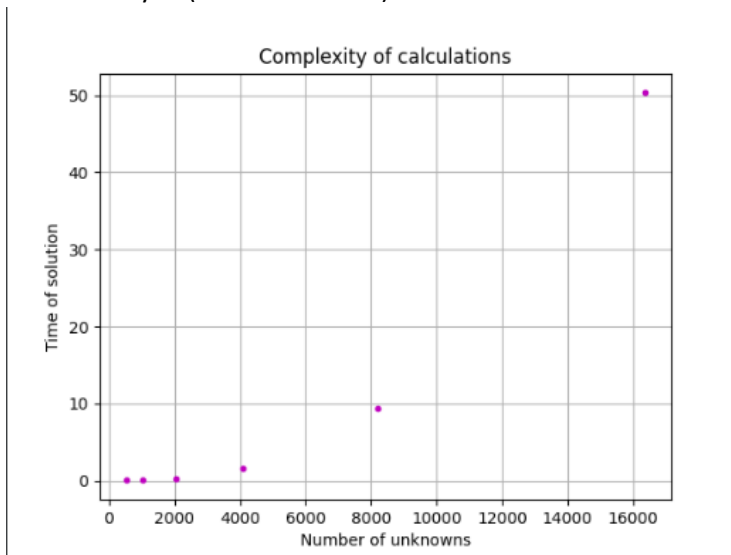
Link do repozytorium: [GitHub - grzesiaaa/Algorytmy_lista5](https://github.com/grzesiaaa/Algorytmy_lista5)

RAPORT LISTA 5

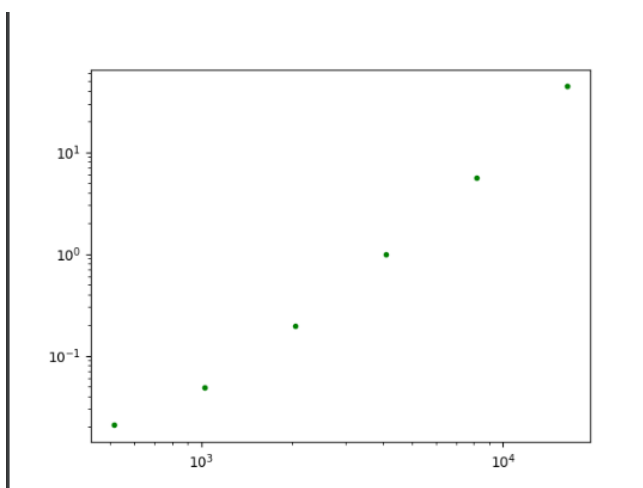
Zadanie 1

Funkcja *check_time* sprawdza czas rozwiązania układu równań dla podanej liczby niewiadomych.

Funkcja *plot_times* tworzy wykres czasów rozwiązania układu równań dla podwojonej liczby niewiadomych (od 2^9 do 2^{14}). Oto rezultat:



Patrząc na wykres wnioskujemy, że mamy złożoność wykładniczą. Sprawdźmy to nakładając logarytm na obie osie wykresu (jeśli wyjdzie funkcja liniowa to znaczy, że teza jest prawdziwa). Zrobimy to za pomocą funkcji *plot_log*. Tak wygląda ten wykres:



Czyli widzimy, że teza poprawna, a więc wzór naszej funkcji ma postać ax^b .

Znajdźmy teraz współczynniki a i b . Z pomocą przyszedł nam internet, a konkretnie ta strona:

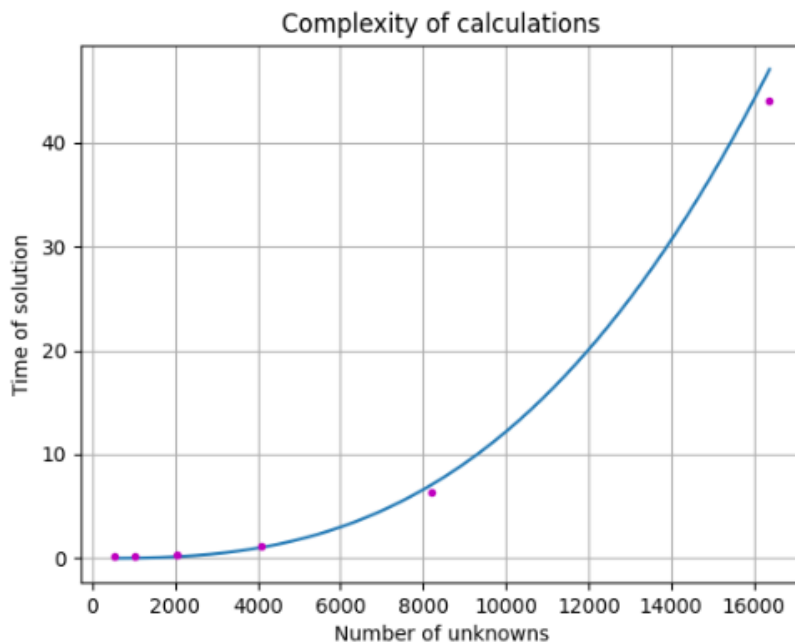
https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html?fbclid=IwAR0ieVgtRhugAaWYWAQTDKlh9qusazAVxcxp3P_ojiozXlCjDs-H_A8bmEo

W funkcjach *func* i *find_factors* posługujemy się opisanymi tam metodami i wyliczamy, że nasze a i b mają odpowiednio takie wartości:

[1.3563650748562964e-10, 2.7387256394573027]

a więc złożoność to $O(n^{2,74})$.

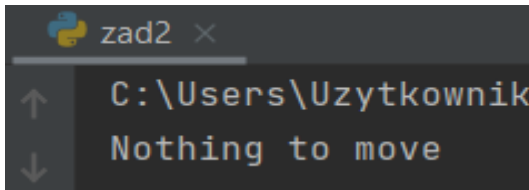
Sprawdźmy, czy te dwa wykresy (czasów rozwiązania i ax^b dla znalezionych współczynników) pokrywają się. Posłuży nam do tego funkcja *check*.



Widzimy, że wykresy te w miarę ładnie się pokrywają, a więc znalezione współczynniki i złożoność są poprawne.

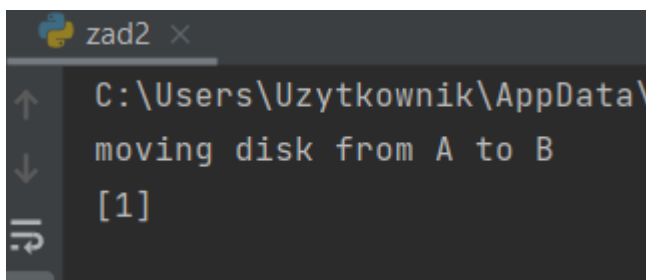
Zadanie 2

W tym zadaniu korzystamy ze stosu. Funkcja `move_disc` sprawdza, czy są do przesunięcia krążki i jeśli tak to przesuwa je. Jeśli lista jest pusta to program wyrzuca „Nothing to move”,



```
zad2 x
C:\Users\Uzytkownik
Nothing to move
```

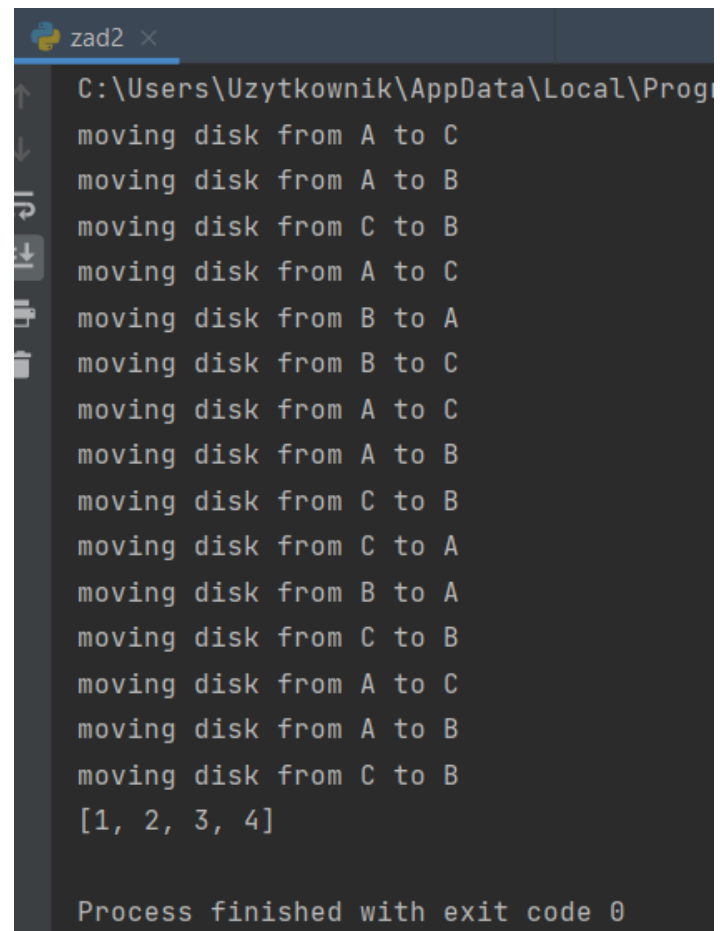
jeśli jest jeden to przesuwa tylko jego i wyrzuca listę jednoelementową. Poniżej przykład:



```
zad2 x
C:\Users\Uzytkownik\AppData\Local\Programs\Python\Python39-64\Scripts\python.exe
moving disk from A to B
[1]
```

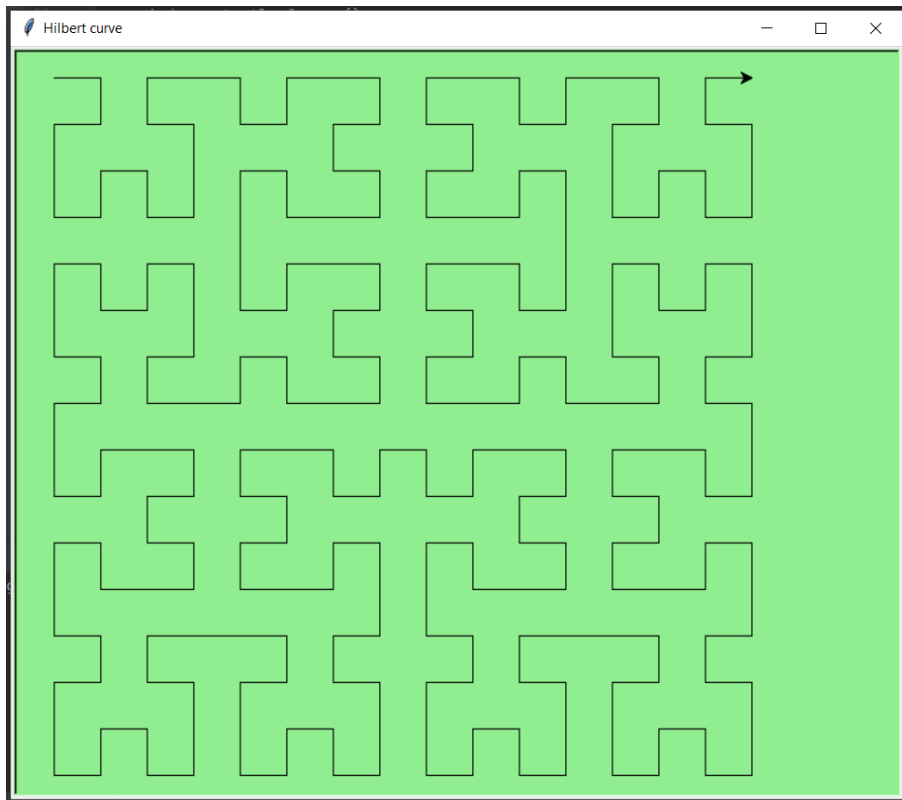
Gdy zaś jest ich więcej to każdego przesuwa tak, aby działała zasada wieży Hanoi. Czyli przesuujemy n krążków (w naszym kodzie n to `amount`) z A(init) do B(final), przy pomocy C(temp). Aby to zrobić musimy najpierw przenieść $n-1$ krążków z A na C przy pomocy B. Potem ten ostatni pozostały krążek przenosimy na B i $n-1$ krążków z C na B przy pomocy A.

Każdy ruch jest printowany, tzn. widzimy, z którego stosu na który krążek jest przenoszony. Obok na zdjęciu przykład dla 4 krążków.



```
zad2 x
C:\Users\Uzytkownik\AppData\Local\Programs\Python\Python39-64\Scripts\python.exe
moving disk from A to C
moving disk from A to B
moving disk from C to B
moving disk from A to C
moving disk from B to A
moving disk from B to C
moving disk from A to C
moving disk from A to B
moving disk from C to B
moving disk from C to A
moving disk from B to A
moving disk from C to B
moving disk from A to C
moving disk from A to B
moving disk from C to B
[1, 2, 3, 4]
Process finished with exit code 0
```

Zadanie 3



Na zdjęciu wywołanie dla $degree = 4$ i $length = 40$.

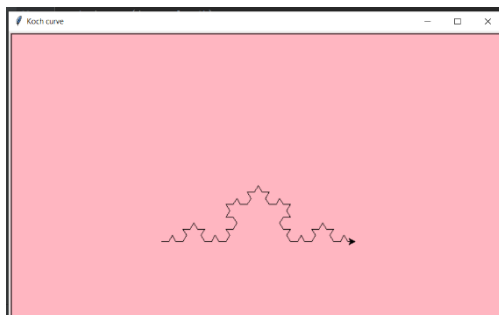
Najpierw stworzyliśmy osobną funkcję *hilbert_curve* do rysowania toru ruchu krzywej Hilberta.

Funkcja *main* tworzy okno, na którym możemy zobaczyć wizualizację przy podanych wartościach *degree* oraz *length*. Najpierw obracamy żółwia o 90 stopni, wykonujemy krzywą dla $degree - 1$ i kąta -90 stopni. Następnie kierujemy się prosto i znów się obracamy ale w drugą stronę i znowu wykonujemy krzywą dla $degree - 1$. Jesteśmy dobrze ustawieni, więc nie musimy się teraz odwracać i znowu krzywa dla $degree - 1$. Następnie znowu obrót i idziemy do przodu i algorytm dla $degree-1$. Na koniec jeszcze raz się obracamy.

Zadanie 4

W tym zadaniu miałyśmy zwizualizować krzywą Kocha. W pierwszej funkcji *koch* opisujemy kolejne ruchy „turtle’a”.

Dla stopnia zerowego żółwik idzie prosto (tworzy odcinek). Dzielimy podaną długość na trzy części i wykonujemy funkcję dla *degree - 1*, obracamy żółwia o 60 stopni w lewo i znowu wykonujemy funkcję dla *degree-1*. Następnie żółw o 120 stopni w prawo i funkcja dla *degree-1*. Na koniec żółw znowu 60 stopni w lewo i znowu funkcja dla *degree-1*.



```
4 def koch(degree, length):
5     if degree == 0:
6         turtle.forward(length)
7     elif degree > 0:
8         length /= 3
9         koch(degree - 1, length)
10        turtle.left(60)
11        koch(degree - 1, length)
12        turtle.right(120)
13        koch(degree - 1, length)
14        turtle.left(60)
15        koch(degree - 1, length)
```

W funkcji *curve* możemy narysować określony odcinek krzywej, w zależności od *degree* oraz długości *length*.

Naszym ostatnim zadaniem było stworzenie z krzywych kocha kształt płatka śniegu. Zrobiliśmy to w funkcji *snowflake*. Aby to wykonać należało połączyć pod odpowiednim kątem trzy krzywe kocha ze sobą.

