Wrocław, 02.11.2021r.

Lista zadań nr 2

Julia Mazur, 262296

Przetestuję kolejne metody pod względem ilości obliczeń.
(wyniki działania funkcji są przedstawione wg wzoru
$[< potwierdzenie\ istnienia\ trójki\ pitagorejskiej >, a, b, c, < ilość\ wykonanych\ operacji >]$)

1. Zacznijmy od najbardziej podstawowej, przedstawionej na zajęciach, czyli takiej, w której sprawdzamy podane w zadaniu warunki dla każdego a, b, c z zakresu $[0, n)$.

   ```
   find_abc1(1000)
   ```
   ```
   [True, 200, 375, 425, 2805456541]
   ```

   Widzimy, że ilość wykonanych operacji jest w miliardach, czyli metoda zdecydowanie nie jest optymalna. Dodatkowo do policzenia najprostszej trójki pitagorejskiej, czyli takiej której $n = 12$ funkcja działa w następujący sposób:

   ```
   find_abc1(12)
   ```
   ```
   [True, 3, 4, 5, 6849]
   ```

   Nawet dla najmniejszego n ilość operacji jest dość duża.

2. Aby nieco usprawnić działanie powyższej metody uwzględniłam fakt, że aby trójkąt istniał suma długości jego dwóch boków musi być większa od długości trzeciego, czyli dla dowolnej kombinacji $a,\ b,\ c$ prawdą jest, że:
   1) $a + b > c$;
   2) $a + b + c = n$ (to wiemy z warunków zadania);

   Z (2) widzimy, że $a + b = n - c$ i po podstawieniu do (1) otrzymujemy $c < \frac{n}{2}$. Zatem możemy ograniczyć zakres naszych poszukiwań do $[0, \frac{n}{2})$.
   Wtedy wyniki działania funkcji wyglądają następująco:

   ```
   find_abc2(1000)
   ```
   ```
   [True, 200, 375, 425, 705539791]
   ```
   ```
   find_abc2(12)
   ```
   ```
   [True, 3, 4, 5, 2564]
   ```

   Widzimy, że ilość operacji się zmniejszyła, ale dalej jest duża.

3. Skoro liczby a, b, c są całkowite, to wiemy, że muszą być różne, ponieważ jeśli $a = b$ to $c = a\sqrt{2}$. Bez utraty ogólności możemy założyć, że $a < b < c$, więc możemy zmodyfikować pętle:

```
find_abc3(1000)
```

```
[True, 200, 375, 425, 163783726]
```

```
find_abc3(12)
```

```
[True, 3, 4, 5, 340]
```

4. Możemy zapisać $c$jako $n - a - b$ w ten sposób możemy zredukować nasz program z trzech do dwóch pętli for. Wtedy:

```
find_abc4(1000)
```

```
[True, 200, 375, 425, 802951]
```

```
find_abc4(12)
```

```
[True, 3, 4, 5, 164]
```

   W tym przypadku możemy już zauważyć znaczne zmniejszenie ilości wykonanych operacji.

5. Wykonując kolejne przekształcenia, możemy uzależnić a i b od c:
   Z warunku początkowego możemy zapisać:
   $a + b + c = n$, (1)
   $a + b = n - c$.
   Aby trójkąt był trójkątem pitagorejskim musi być spełniony również warunek
   $a^2 + b^2 = c^2$,
   $a^2 + 2ab + b^2 = c^2 + 2ab$;
   $(a + b)^2 = c^2 + 2ab$;
   $(n - c)^2 = c^2 + 2ab$;
   $2ab = (n - c)^2 - c^2$.
   Teraz wyznaczymy wzór na różnicę a i b za pomocą c:
   $a^2 + b^2 - 2ab = c^2 - (n - c)^2 + c^2$;
   $(a - b)^2 = c^2 - n^2 + 2nc$.
   Powracając do (1) wyznaczymy b:
   $b = n - c - a$;
   $2b = n - c - (a - b)$;
   $b = \frac{n-c-(a-b)}{2}$.

Od razu możemy zapisać wzór na a:

$a = n - c - b.$

Z wcześniejszych założeń wiemy, że

$a < b;$

$a - b < 0.$

Zatem:

$a - b = -\sqrt{c^2 - n^2 + 2nc}.$

Dodatkowo możemy ustalić dolną granicę c:

$c^2 - n^2 + 2nc > 0;$

$\Delta = 4n^2 + 4n^2 = 8n^2;$

$c_1 = -n - n\sqrt{2};$

$c_2 = -n + n\sqrt{2};$

$c > -n + n\sqrt{2}.$

Wtedy otrzymujemy następujące wyniki:

```
find_abc5(1000)
```

```
[True, 200, 375, 425, 107]
```

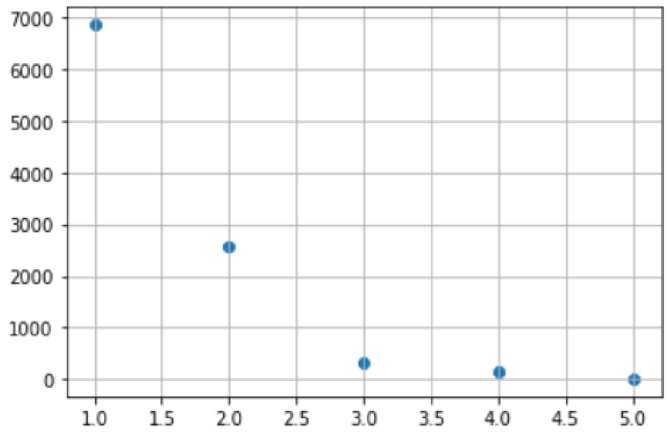
```
find_abc5(12)
```

```
[True, 3, 4, 5, 17]
```

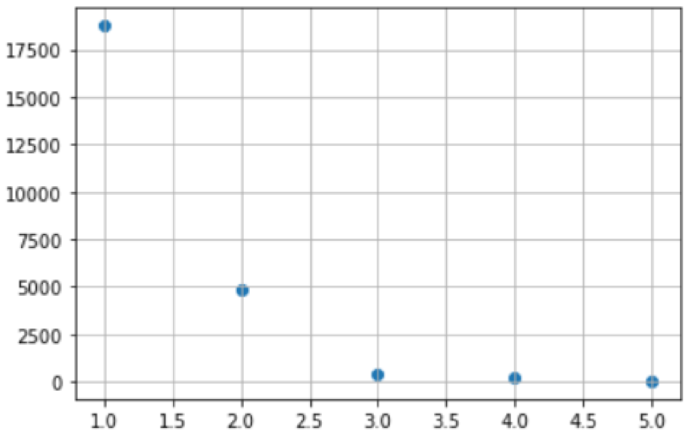
Wnioski:

Możemy zobaczyć zależność ilości operacji potrzebnych do znalezienia trójki pitagorejskiej od numeru użytej funkcji:

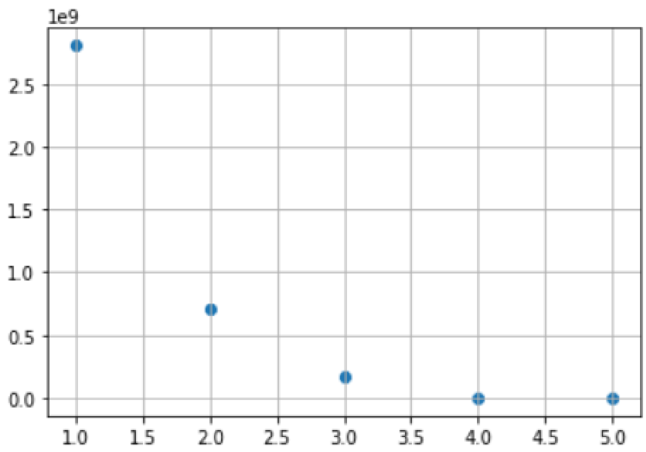
1) dla $n = 12$:



2) dla $n = 11$:



3) dla $n = 1000$:



Widzimy, że dla $n = 1000$ udało nam się zredukować ilość operacji ponad $26 * 10^6$ $-$krotnie. Jednak nawet dla najmniejszego sensownego $n = 12$ zmiana jest znaczna, bo ok 400 $-$krotna. Przy $n = 11$program musi wykonać maksymalną ilość operacji dla tej liczby, ponieważ poszukiwana trójka nie istnieje. Mimo że nawet w optymalnej wersji musi być wykonana większa ilość operacji, wypada ona ponad tysiąc razy lepiej od metody podstawowej.

link do kodu:
https://github.com/Swinkawkrawacie/algorytmy2021-22/blob/6c3e80e867edd5ae05fb7c662b6eed0ac8ef3f6c/lista2.ipynb