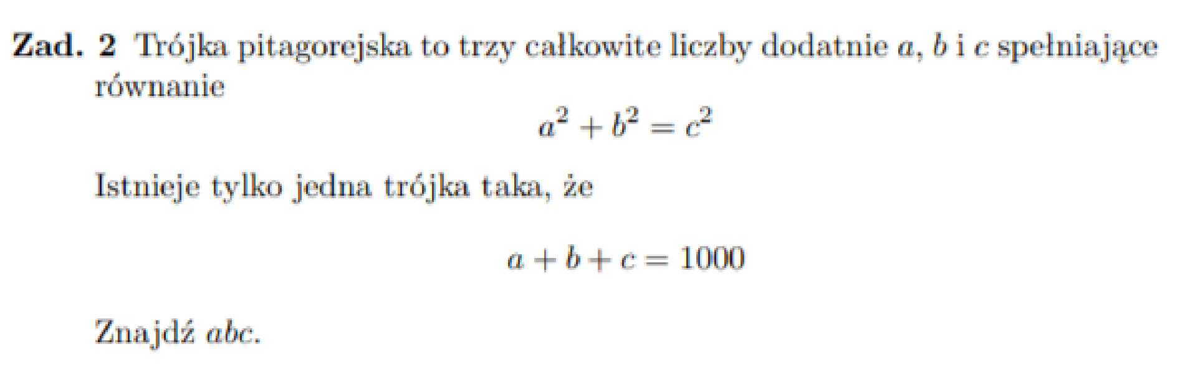
Algorytmy i stuktury danych - Lista 2

Alicja Myśliwiec - gr. wtorek 7:30



Zad. 2 Trójka pitagorejska to trzy całkowite liczby dodatnie a, b i c spełniające równanie

$$a^2 + b^2 = c^2$$

Istnieje tylko jedna trójka taka, że

$$a + b + c = 1000$$

Znajdź abc.

In [167…
```python
import math
import time
```

Pierwsze rozwiązanie, które będzie stopniowo optymalizowane

In [168…
```python
def pythagorean_triple_1(l):

    operations = 0
    for a in range(1, l):
        for b in range(1, l):
            for c in range(1, l):
                operations += 9
                if a ** 2 + b ** 2 == c ** 2 and a + b + c == l:

                    return True, [a, b, c], operations

    return False, [-1, -1, -1], operations
```

In [169…
```python
pythagorean_triple_1(29)
```

Out[169…
```
(False, [-1, -1, -1], 197568)
```

In [170…
```python
pythagorean_triple_1(30)
```

Out[170…
```
(True, [5, 12, 13], 33264)
```

Wiemy, że b musi być większe o a, tak jak i c musi być większe od c - stąd ograniczymy range pętli.

In [231…
```python
def pythagorean_triple_2(l):

    operations = 0
    for a in range(1, l):
        operations += 1
        for b in range(a + 1, l):
            operations += 1
            for c in range(b + 1, l):
                operations += 9
                if a ** 2 + b ** 2 == c ** 2 and a + b + c == l:

                    return True, [a, b, c], operations

    return False, [-1, -1, -1], operations
```

In [232…
```python
pythagorean_triple_2(100)
```

Out[232…
```
(False, [-1, -1, -1], 1416591)
```

In [233…
```python
pythagorean_triple_2(132)
```

Out[233…
```
(True, [11, 60, 61], 746065)
```

Trójka pitagorejska ma swoje zastosowanie w trójkątach, dlatego też dla kolejnej optymalizacji skorzystamy z nierówności trójkąta.

$$c < a + b$$

In [234…
```python
def pythagorean_triple_3(l):

    operations = 0
    for a in range(1, l):
        operations += 1
        for b in range(a + 1, l):
            limit = a + b
            operations += 2
            for c in range(b + 1, limit):
                operations += 9
                if a ** 2 + b ** 2 == c ** 2 and a + b + c == l:

                    return True, [a, b, c], operations

    return False, [-1, -1, -1], operations
```

In [256…
```python
pythagorean_triple_3(100)
```

Out[256…
```
(False, [-1, -1, -1], 1421442)
```

In [237…
```python
pythagorean_triple_3(132)
```

Out[237…
```
(True, [11, 60, 61], 57033)
```

Warunek a + b + c == l oraz samą pętle dla c można uprościć poprzez założenie, że $c = l - a - b$

In [238…
```python
def pythagorean_triple_4(l):

    operations = 0
    for a in range(1, l):
        operations += 1
        for b in range(a + 1, l):
            c = l - a - b
            operations += 7
            if a ** 2 + b ** 2 == c ** 2:

                    return True, [a, b, c], operations

    return False, [-1, -1, -1], operations
```

In [239…
```python
pythagorean_triple_4(160)
```

Out[239…
```
(True, [32, 60, 68], 31259)
```

In [240…
```python
pythagorean_triple_4(162)
```

Out[240…
```
(False, [-1, -1, -1], 90321)
```

Można również spróbować obejść pętle dla b, wyznaczając jego wartość z dwóch danych nam równań.

$$\begin{cases} c = l - a - b \\ a^2 + b^2 = c^2 \end{cases} \Rightarrow \quad b = \frac{l^2 - 2al}{2l - 2a}$$

In [241…
```python
def pythagorean_triple_5(l):

    operations = 0
    for a in range(1, l):
        b = int((l ** 2 - 2 * a * l)/(2 * (l - a)))
        c = l - a - b
        operations += 16
        if a ** 2 + b ** 2 == c ** 2 and b > a:

                return True, [a, b, c], operations

    return False, [-1, -1, -1], operations
```

In [242…
```python
pythagorean_triple_5(1000)
```

Out[242…
```
(True, [200, 375, 425], 3200)
```

Coś co można by zrobić na początku, ja jednak robię na końcu - ograniczenie a, ponieważ zakładając że jest to najmniejsza liczba z szukanej trójki, nigdy nie będzie większa niż 1/3 całej sumy.

In [243…
```python
def pythagorean_triple_6(l):

    limit = l // 3
    operations = 1
    for a in range(1, limit):
        b = int((l ** 2 - 2 * a * l)/(2 * (l - a)))
        c = l - a - b
        operations += 14
        if a ** 2 + b ** 2 == c ** 2:

                return True, [a, b, c], operations

    return False, [-1, -1, -1], operations
```

In [244…
```python
pythagorean_triple_6(1000)
```

Out[244…
```
(True, [200, 375, 425], 2801)
```

Jak widać w przypadku, gdy istnieje trójka, jest to subtelna optymalizacja - jednak jest to w stanie 3 krotnie skrócić poszukiwania, gdy taka trójka nie istnieje:

In [247…
```python
pythagorean_triple_5(100)
```

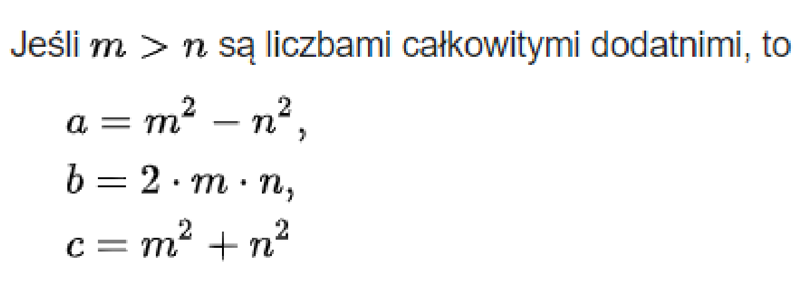
Out[247…
```
(False, [-1, -1, -1], 1584)
```

In [248…
```python
pythagorean_triple_6(100)
```

Out[248…
```
(False, [-1, -1, -1], 449)
```

Parametryzacja a, b i c

Z pomocą parametrów m i n jesteśmy w stanie zoptymalizować jedną pętle, jednak są one w stanie znaleźć tylko trójki pierwotne - czyli takie, które nie mają innych wspólnych dzielników niż 1.



Jeśli $m > n$ są liczbami całkowitymi dodatnimi, to

$$a = m^2 - n^2,$$
$$b = 2 \cdot m \cdot n,$$
$$c = m^2 + n^2$$

In [185…
```python
def primary_pythagorean_triple_parametric(l):
    n, c, operations = 1, 1, 0
    list_true, list_false = [True], [False, [-1, -1, -1]]

    while c < l:
        for m in range(1, n):
            a = n ** 2 - m ** 2
            b = 2 * m * n
            c = n ** 2 + m ** 2

            sum = a + b + c
            operations += 11

            if sum == l:
                list_of_numbers = [a , b , c]
                list_of_numbers.sort()

                for ele in [list_of_numbers, operations]:
                    list_true.append(ele)
                return list_true

        operations += 1
        n += 1

    list_false.append(operations)
    return list_false
```

In [186…
```python
print(primary_pythagorean_triple_parametric(40))
```

```
[True, [8, 15, 17], 47]
```

In [262…
```python
print(primary_pythagorean_triple_parametric(200))
```

```
[False, [-1, -1, -1], 616]
```

Z wcześniejszych wywołań wiemy, że liczba 150 ma swoją trójkę. Nie jest ona jednak trójką pierwotna, dlatego powyższa funkcja jej nie znajdzie - potrzebna jest modyfikacja.

In [249…
```python
def pythagorean_triple_parametric(l):
    n, c, operations = 1, 1, 0
    list_true, list_false = [True], [False, [-1, -1, -1]]

    while c < l:
        for m in range(1, n):
            a = n ** 2 - m ** 2
            b = 2 * m * n
            c = n ** 2 + m ** 2

            sum = a + b + c
            operations += 11
            if l % sum == 0:

                factor = l // sum
                operations += 2

                if factor != 0:
                    operations += 3

                    list_of_numbers = [a * factor, b * factor, c * factor]
                    list_of_numbers.sort()

                    for ele in [list_of_numbers, operations]:
                        list_true.append(ele)
                    return list_true

        operations += 1
        n += 1

    list_false.append(operations)
    return list_false
```

In [250…
```python
print(pythagorean_triple_parametric(1000))
```

```
[True, [200, 375, 425], 52]
```

In [255…
```python
print(pythagorean_triple_parametric(1200785))
```

```
[False, [-1, -1, -1], 3308476]
```

Porównanie czasu wykonania oraz wykonanych operacji

In [97]:
```python
start = time.time()
data_1 = pythagorean_triple_1(1000)
stop = time.time()
print(stop-start, data_1[2])
```

```
554.3041272163391 1790786250
```

In [162…
```python
start = time.time()
data_2 = pythagorean_triple_2(1000)
stop = time.time()
print(stop-start, data_2[2])
```

```
235.42063212394714 888566250
```

In [163…
```python
start = time.time()
data_3 = pythagorean_triple_3(1000)
stop = time.time()
print(stop-start, data_3[2])
```

```
51.32726311683655 205070100
```

In [164…
```python
start = time.time()
data_4 = pythagorean_triple_4(1000)
stop = time.time()
print(stop-start, data_4[2])
```

```
0.610797643661499 1432608
```

In [165…
```python
start = time.time()
data_5 = pythagorean_triple_5(1000)
stop = time.time()
print(stop-start, data_5[2])
```

```
0.000997304916381836 3200
```

In [166…
```python
start = time.time()
pythagorean_triple_parametric(1000)
stop = time.time()

print(stop-start)
```

```
0.0
```

In [98]:
```python
start = time.time()
for i in range(1, 1000):
    pythagorean_triple_parametric(1000)
stop = time.time()
data_p = pythagorean_triple_parametric(1000)
print((stop-start)/1000, data_p[2])
```

```
3.1697511672973636e-05 56
```

Jak widać nienajlepszym pomysłem jest wywołanie funkcji dla 1000 w przypadku 3 pierwszych funkcji, dopiero od 4 optymlizacji (obcięcie jednej pętli) jakkolwiek opłacalne jest czekać na wynik - dla rozwiązania z parametrem sprzęt nie jest w stanie zmierzyć czasu funckji wykonanej jednokrotnie :)

Szukanie wszystkich trójek

In [263…
```python
def all_pythagorean_triple(l):

    operations, triplets = 2, []
    limit = l // 3

    for a in range(1, limit):
        b = int((l ** 2 - 2 * a * l)/(2 * (l - a)))
        c = l - a - b
        operations += 14
        if a ** 2 + b ** 2 == c ** 2:
            triple = [a, b, c]
            triple.sort()
            if triple in triplets:
                break
            triplets.append(triple)

    if len(triplets) == 0:

        return False, [-1, -1, -1], operations
    return True, triplets, operations
```

In [264…
```python
all_pythagorean_triple(100)
```

Out[264…
```
(False, [-1, -1, -1], 450)
```

In [265…
```python
all_pythagorean_triple(1000)
```

Out[265…
```
(True, [[200, 375, 425]], 4650)
```

In [260…
```python
all_pythagorean_triple(1200)
```

Out[260…
```
(True,
 [[48, 575, 577],
  [75, 560, 565],
  [200, 480, 520],
  [240, 450, 510],
  [300, 400, 500]],
 5588)
```

In [261…
```python
all_pythagorean_triple(2100)
```

Out[261…
```
(True,
 [[140, 975, 985],
  [225, 924, 951],
  [300, 875, 925],
  [336, 850, 914],
  [350, 840, 910],
  [525, 700, 875],
  [600, 630, 870]],
 8822)
```

Sprawdźmy czy wszystkie sugerowane trójki się zgadzają

In [208…
```python
def find_c2(a, b):
    return math.sqrt(a ** 2 + b ** 2)
```

In [209…
```python
find_c2(48, 575)    # 577
```

Out[209…
```
577.0
```

In [210…
```python
find_c2(75, 560)    # 565
```

Out[210…
```
565.0
```

In [211…
```python
find_c2(200, 480)   # 520
```

Out[211…
```
520.0
```

In [212…
```python
find_c2(240, 450)   # 510
```

Out[212…
```
510.0
```

In [213…
```python
find_c2(300, 400)   # 500
```

Out[213…
```
500.0
```

LINK GITHUB

https://github.com/AlutkaMalutka/Programowanie_python/tree/main/semestr_3/Lista_2-3s-