Julia Grzegorzewska, Wiktoria Fimińska

Link do repozytorium: https://github.com/grzesiaaa/Algorytmy_lista2

RAPORT LISTA 2

Znalazłyśmy 5 sposobów na znalezienie trójki pitagorejskiej o podanym obwodzie trójkąta. Każda z nich w kolejności jest bardziej optymalna od poprzedniej.

Pierwszy z nich został zaprezentowany na zajęciach. Polega on na tym, że każdy bok jest poszukiwany w pętli spośród liczb mniejszych od obwodu trójkąta. Następnie sprawdzane są zależności $a^2 + b^2 = c^2$ oraz $a + b + c = l$, gdzie l to obwód trójkąta . Jeśli wartości spełniają je, program wyrzuca krotkę z True, długościami boków i liczbą operacji.

```
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
    else:
        return False, None, None, None, operations
```

W następnym korzystamy już tylko z dwóch pętli, a trzeci bok otrzymujemy z różnicy obwodu i sumy reszty boków, jeśli spełniają powyższe zależności wyrzucają krotkę taką, jak w poprzednim programie.

```
def pythagorean_triple_2(l):
    operations = 0
    for a in range(1, l):
        for b in range(a, l):
            operations += 7
            c = l - a - b
            if a ** 2 + b ** 2 == c ** 2:
                return True, a, b, c, operations
    else:
        return False, None, None, None, operations
```

W trzecim ograniczamy poszukiwane wartości do połowy wartości obwodu i działamy analogicznie jak w poprzednim.

```
def pythagorean_triple_3(l):
    operations = 0
    for a in range(1, l//2):
        for b in range(a, l//2):
            operations += 7
            c = l - a - b
            if a ** 2 + b ** 2 == c ** 2:
                return True, a, b, c, operations
    else:
        return False, None, None, None, operations
```

Ostatni rozwiązuje problem najszybciej. Pierwszy bok poszukiwany jest w pętli od 1 do trzeciej części obwodu, a dwa kolejne są uzależnione od niego oraz obwodu. Tym razem przy sprawdzaniu zależności $a^2 + b^2 = c^2$ nie zapisujemy potęgi jako „a**2” lecz „a*a” i dzięki temu program działa szybciej.

```
def pythagorean_triple_4(l):
    operations = 0
    for a in range(1, l//3):
        operations += 14
        b = (2*a*l-l*l)//(2*(a-l))
        c = l - a - b
        if a * a + b * b == c * c:
            return True, a, b, c, operations
    else:
        return False, None, None, None, operations
```

W każdym przykładzie liczona jest też liczba operacji oraz przy wywoływaniu funkcji obliczamy czas jej obliczeń. Zrobiłyśmy obliczenia za pomocą pętli ponieważ czas był tak mały, że często wyskakiwał wynik równy 0. Poniżej wyniki dla l = 90

Nazwa funkcji	Liczba operacji	Czas wykonania
pythagorean_triple_1	601920	0.034354041528701786 s
pythagorean_triple_2	5012	0.0003429825305938721 s
pythagorean_triple_3	2492	0.00017083234786987304 s
pythagorean_triple_4	126	1.5979766845703125e-06 s