Lista 2

repo: https://github.com/KacperBudnik/AiSD/tree/main/List%202

In [1]:
```
import math
import random
```

Sposób głupi polega na losowaniu po kolei liczby i sprawdzaniu czy spełniają założenia (dana kombinacja jest sprawdzana tylko raz)

In [2]:
```
def fun0(n):#Głupie
    k=0
    a=[x for x in range(1,n+1)]
    while a:
        x=a.pop(random.randint(0,len(a)-1))
        b=[x for x in range(1,n+1)]
        while b:
            y=b.pop(random.randint(0,len(b)-1))
            c=[x for x in range(1,n+1)]
            while c:
                z=c.pop(random.randint(0,len(c)-1))
                k+=8
                if x**2+y**2==z**2 and x+y+z==n:
                    return(x,y,z,k)
    return (0,0,0,k)
```

Sposób najwolniejszy ale czasem może rozwiązać problem jedynie kilkoma operacjami (Poniżej, za 1110 razem program znalazł trójkę przy 8 operacjach)

In [3]:
```
i=0
while True:
    i+=1
    d=fun0(12)
    if d[-1]<10:
        break;

print(i,d)
```

```
102 (3, 4, 5, 8)
```

Drugi sposób pochodzi z ćwiczeń

In [4]:
```
def fun(n):# Z ćwiczeń
    k=0
    for a in range(1,n+1):
        for b in range(1,n+1):
            for c in range(1,n+1):
                k+=8
                if a**2+b**2==c**2 and a+b+c==n:
                    return (a,b,c,k)
    return (0,0,0,k)
```

Trzeci jest ulepszeniem drugiego

In [5]:
```
def fun2(n): # Z ćwiczeń ulepszone
    k=0
    for a in range(1,n+1):
        for b in range(1,n+1):
            k+=7
            if a**2+b**2 == (n-a-b)**2:
```

```
                return (a,b,n-a-b,k)
    return (0,0,0,k)
```

Mój pomysł polega na czy dla danego 'C' istnieją liczby a,b spełniające żądane warunki. Jeśli mamy znane C, wystrczy jednocześnie zwiększać a i zmniejszać b dopóki nie spełnimy warunków (lub a będzie równe 0)

In [6]:
```
# Własny pomysł
def fun3_help(n,c):
    k=0
    b=int(c//math.sqrt(2))
    a=n-c-b
    m=c**2
    k+=10
    while a**2+b**2<m and a>0:
        a-=1
        b+=1
        k+=7

    return (a,b,k)



def fun3(n):
    k=0
    for i in range(1,n//2):
        d=fun3_help(n,i)
        k+=1+d[-1]
        if d[0]!=0:
            k+=5
            if d[0]**2+d[1]**2==i**2:
                return (d[0],d[1],i,k)
    return (0,0,0,k)
```

Ulepszenie tego sposobu:

• Liczba n musi być parzysta

• Szukamy trójki pierwotnej, więc c jest nie parzyste (z tego wynika, że a i b nie mają tej samej parzystości)

• Zmniejszamy a lub zwiększamy b dopóki nie trafimy na odpowiednie, bądź a będzie mniejsze od b

• Jeżeli n jest podzielne przez a+b+c, to mamy szukaną trójkę (pierwotną mnożymy przez uzyskaną liczbę)

• Dodatkowo program może szukać (jeśli `every=True`) wszystkich trójek spełniające warunki zadania

In [7]:
```
# Włąsny ulepszony
def fun3_help_v2(n,c,k):
    b=1
    a=c-1
    m=c**2

    e=a**2+b**2

    k+=5

    while a>b:
```

```
            k+=1
            if e>m:
                a-=1
                k+=2
            elif e<m:
                b+=1
                k+=3
            else:
                k+=2
                if (n/(a+b+c)).is_integer():
                    k+=4
                    break
                else:
                    a-=1
                    b+=1
                    k+=6
            e=a**2+b**2
            k+=3

        d=n/(a+b+c)
        k+=3

        if d.is_integer() and a**2+b**2==m:
            k+=6
            d=int(d)
            k+=4
            return (d*a,d*b,d*c,k)
        return (0,0,0,k)



def fun3_v2(n,every=False):
    k=0
    if every:
        k+=1
        e=[]
        k+=2
        for i in range(5,n//2,2):
            k+=1
            d=fun3_help_v2(n,i,k)
            k+=d[-1]
            k+=1
            if d[0]!=0:
                k+=1
                e.append((d[0],d[1],d[2]))
        return (set(e),k)
    else:
        k+=3
        for i in range(5,n//2,2):
            k+=1
            d=fun3_help_v2(n,i,k)
            k+=d[-1]
            k+=2
            if d[0]!=0:
                return (d[0],d[1],d[2],k)
        return (0,0,0,k)
```

Ostatnie sposób wykorzystuje własność, że

$a = m^2 - n^2$

$b = 2mn$

$c = m^2 + n^2$

Dla pewnych liczb całkowitych m,n które są względnie pierwsze. Niestety, ponieważ m i n są względnie pierwsze program znajduje zawsze trójki pierwotne, lecz pozostałe jedynie gdy spełniają dodatkowe założenia (szukana trójka jest wielokrotnością kwadratu liczby naturalnej).

In [8]:
```
def fun4(n):
    k=2
    if n%2!=0:
        return (0,0,0,k)
    x=int(math.sqrt(n))
    y=1
    m=n//2
    k+=3
    while x>y:
        k+=3
        d=x*(x+y)
        if m>d:
            y+=1
            k+=2
        elif m<d:
            x-=1
            k+=3
        else:
            k+=2
            return (x**2-y**2,2*x*y,x**2+y**2,k)

    return (0,0,0,k)
```

Dla tej funkcji złożoność asymptotyczna to $o(\sqrt{n})$, (zaczynamy od $x = \sqrt{n}$ oraz $y = 1$ i zbliżamy te liczby do siebie o 1 w każdej iteracji)

Ulepszona funkcja szuka wszystkich miejsc (funkcja fun4_v2_help od fun4 różni się tylko brakiem sprawdzania czy liczba jest parzysta)

In [9]:
```
def fun4_v2_help(n):
    x=int(math.sqrt(n))
    y=1
    m=n//2
    k=3
    while x>y:
        k+=3
        d=x*(x+y)
        if m>d:
            y+=1
            k+=2
        elif m<d:
            x-=1
            k+=3
        else:
            k+=2
            return (x**2-y**2,2*x*y,x**2+y**2,k)

    return (0,0,0,k)

def fun4_v2(n):
    k=1
    for i in range(6,n+1,2):
        d=fun4_v2_help(i)
        k+=d[-1]+1
        if d[1]!=0:
            k+=3
            if (n/sum(d[:-1])).is_integer():
```

```
                k+=3
                e=int(n/sum(d[:-1]))
                k+=3
                return(d[0]*e,d[1]*e,d[2]*e,k)
    return(0,0,0,k)
```

Porównanie metod

Czas działania

In [10]:
```
%%timeit
fun0(12)
```

```
865 µs ± 21.1 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [11]:
```
%%timeit
fun(200)
```

```
930 ms ± 2.55 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [12]:
```
%%timeit
fun2(1000)
```

```
126 ms ± 528 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [13]:
```
%%timeit
fun3(1000)
```

```
1.05 ms ± 14 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [14]:
```
%%timeit
fun3_v2(1000)
```

```
38.2 µs ± 615 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [15]:
```
%%timeit
fun4(1000)# Ale może nie znaleźć
```

```
3.58 µs ± 99 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

In [16]:
```
%%timeit
fun4_v2(1000)
```

```
20.5 µs ± 793 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

In [17]:
```
%%timeit
fun3_v2(1000,True)#wszystkie
```

```
34.4 ms ± 449 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [18]:
```
%%timeit
fun3_v2(10**15)
```

```
38.5 µs ± 479 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [19]:
```
%%timeit
fun4_v2(10**15)
```

```
21.1 µs ± 384 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Liczba operacji

In [20]:
```
fun0(12)[-1]
```

```
Out[20]: 1936
```

```
In [21]: fun(200)[-1]
```

```
Out[21]: 12599080
```

```
In [22]: fun2(1000)[-1]
```

```
Out[22]: 1395625
```

```
In [23]: fun3(1000)[-1]
```

```
Out[23]: 11035
```

```
In [24]: fun3_v2(1000)[-1]
```

```
Out[24]: 6438
```

```
In [25]: fun4(1000)[-1]
```

```
Out[25]: 96
```

```
In [26]: fun4_v2(1000)[-1]
```

```
Out[26]: 401
```

```
In [27]: fun3_v2(1000,True)[-1] # Szuka wszystkich
```

```
Out[27]: 2279448508650520187119070718691325581058448958340607577561075387593201986227 5
```

```
In [28]: fun3_v2(2100)
```

```
Out[28]: (700, 525, 875, 49)
```

```
In [29]: fun3_v2(2100, True)
```

```
Out[29]: ({(630, 600, 870),
  (700, 525, 875),
  (840, 350, 910),
  (850, 336, 914),
  (875, 300, 925),
  (924, 225, 951),
  (975, 140, 985)},
 144861106074391049362502241273472822169020941793345106795162230801496032303737 06512
085192936308313759984049151828356059641845092439158428535888615900095247368 15)
```

Tabela

funkcja n	12	200	1000	2100	1001	50566
fun0(n)	855 μs	5.34 s	∞	∞	∞	∞
fun(n)	194 μs	945 ms	∞	∞	∞	∞
fun2(n)	17.9 μs	4.86 ms	130 ms	187 ms	654 ms	∞
fun3(n)	8.44 μs	161 μs	1.02 ms	1.53 ms	5.48 ms	∞
fun3_v2(n)	3.09 μs	38.3 μs	37.4 μs	3.19 μs	35.3 ms	∞
fun4_v2(n)	4.44 μs	21.1 μs	21.1 μs	4.41 μs	1.62 ms	511ms
fun3_v2(n,True)	3.43 μs	1.36 ms	34.8 ms	163 ms	35 ms	∞