In [44]:

```
### Exercise 1
def mycoprime(N):
    result = 0
    for i in range(N):
        if gcd(i, N) == 1:
            result += 1
    return result

print("Exercise 1")
print(mycoprime(100))
print("Compare with euler_phi:", mycoprime(100) == euler_phi(100))
```

```
Exercise 1
40
Compare with euler_phi: True
```

In [45]:

```
### Exercise 2.1
def verify_goldbach(N):
    if N % 2 != 0 and N <= 2:
        return False

    for p in primes(2, N):
        for q in primes(2, N - p + 1):
            if p + q == N:
                return True

def try_goldbach(k):
    count = 0
    for i in range(2, k+2):
        if verify_goldbach(i*2):
            count +=1
        else:
            print(i*2)
    print("{} out of {} verified".format(count, k))

print("Exercise 2.1")
try_goldbach(1000)



### Exercise 2.2
def goldbach_pairs(N):
    pairs = []

    if N % 2 != 0 and N <= 2:
        return pairs

    for p in primes(2, N):
        for q in primes(2, N - p + 1):
            if p + q == N:
                if (p, q) not in pairs and (q, p) not in pairs: # check duplicates
                    pairs.append((p, q))
    return pairs

def try_goldbach_pairs(k):
    for i in range(2, k+2):
        print(goldbach_pairs(i*2))
print("Exercise 2.2")
try_goldbach_pairs(100)
```

```
Exercise 2.1
1000 out of 1000 verified
Exercise 2.2
[(2, 2)]
[(3, 3)]
[(3, 5)]
[(3, 7), (5, 5)]
[(5, 7)]
[(3, 11), (7, 7)]
[(3, 13), (5, 11)]
[(5, 13), (7, 11)]
[(3, 17), (7, 13)]
[(3, 19), (5, 17), (11, 11)]
[(5, 19), (7, 17), (11, 13)]
[(3, 23), (7, 19), (13, 13)]
[(5, 23), (11, 17)]
[(7, 23), (11, 19), (13, 17)]
[(3, 29), (13, 19)]
[(3, 31), (5, 29), (11, 23), (17, 17)]
[(5, 31), (7, 29), (13, 23), (17, 19)]
[(7, 31), (19, 19)]
[(3, 37), (11, 29), (17, 23)]
[(5, 37), (11, 31), (13, 29), (19, 23)]
[(3, 41), (7, 37), (13, 31)]
[(3, 43), (5, 41), (17, 29), (23, 23)]
[(5, 43), (7, 41), (11, 37), (17, 31), (19, 29)]
[(3, 47), (7, 43), (13, 37), (19, 31)]
[(5, 47), (11, 41), (23, 29)]
[(7, 47), (11, 43), (13, 41), (17, 37), (23, 31)]
[(3, 53), (13, 43), (19, 37)]
[(5, 53), (11, 47), (17, 41), (29, 29)]
[(7, 53), (13, 47), (17, 43), (19, 41), (23, 37), (29, 31)]
[(3, 59), (19, 43), (31, 31)]
[(3, 61), (5, 59), (11, 53), (17, 47), (23, 41)]
[(5, 61), (7, 59), (13, 53), (19, 47), (23, 43), (29, 37)]
[(7, 61), (31, 37)]
[(3, 67), (11, 59), (17, 53), (23, 47), (29, 41)]
[(5, 67), (11, 61), (13, 59), (19, 53), (29, 43), (31, 41)]
[(3, 71), (7, 67), (13, 61), (31, 43), (37, 37)]
[(3, 73), (5, 71), (17, 59), (23, 53), (29, 47)]
[(5, 73), (7, 71), (11, 67), (17, 61), (19, 59), (31, 47), (37, 41)]
[(7, 73), (13, 67), (19, 61), (37, 43)]
[(3, 79), (11, 71), (23, 59), (29, 53), (41, 41)]
[(5, 79), (11, 73), (13, 71), (17, 67), (23, 61), (31, 53), (37, 47), (41, 43)]
[(3, 83), (7, 79), (13, 73), (19, 67), (43, 43)]
[(5, 83), (17, 71), (29, 59), (41, 47)]
[(7, 83), (11, 79), (17, 73), (19, 71), (23, 67), (29, 61), (31, 59), (37, 53), (43, 47)]
[(3, 89), (13, 79), (19, 73), (31, 61)]
[(5, 89), (11, 83), (23, 71), (41, 53), (47, 47)]
[(7, 89), (13, 83), (17, 79), (23, 73), (29, 67), (37, 59), (43, 53)]
[(19, 79), (31, 67), (37, 61)]
[(3, 97), (11, 89), (17, 83), (29, 71), (41, 59), (47, 53)]
[(5, 97), (13, 89), (19, 83), (23, 79), (29, 73), (31, 71), (41, 61), (43, 59)]
[(3, 101), (7, 97), (31, 73), (37, 67), (43, 61)]
[(3, 103), (5, 101), (17, 89), (23, 83), (47, 59), (53, 53)]
[(5, 103), (7, 101), (11, 97), (19, 89), (29, 79), (37, 71), (41, 67), (47, 61)]
[(3, 107), (7, 103), (13, 97), (31, 79), (37, 73), (43, 67)]
[(3, 109), (5, 107), (11, 101), (23, 89), (29, 83), (41, 71), (53, 59)]
[(5, 109), (7, 107), (11, 103), (13, 101), (17, 97), (31, 83), (41, 73), (43, 71), (47, 67), (53, 61)]
[(3, 113), (7, 109), (13, 103), (19, 97), (37, 79), (43, 73)]
[(5, 113), (11, 107), (17, 101), (29, 89), (47, 71), (59, 59)]
[(7, 113), (11, 109), (13, 107), (17, 103), (19, 101), (23, 97), (31, 89), (37, 83), (41, 79), (47, 73), (53, 67), (59, 61)]
[(13, 109), (19, 103), (43, 79), (61, 61)]
[(11, 113), (17, 107), (23, 101), (41, 83), (53, 71)]
[(13, 113), (17, 109), (19, 107), (23, 103), (29, 97), (37, 89), (43, 83), (47, 79), (53, 73), (59, 67)]
[(19, 109), (31, 97), (61, 67)]
[(3, 127), (17, 113), (23, 107), (29, 101), (41, 89), (47, 83), (59, 71)]
[(5, 127), (19, 113), (23, 109), (29, 103), (31, 101), (43, 89), (53, 79), (59, 73), (61, 71)]
[(3, 131), (7, 127), (31, 103), (37, 97), (61, 73), (67, 67)]
[(5, 131), (23, 113), (29, 107), (47, 89), (53, 83)]
[(7, 131), (11, 127), (29, 109), (31, 107), (37, 101), (41, 97), (59, 79), (67, 71)]
[(3, 137), (13, 127), (31, 109), (37, 103), (43, 97), (61, 79), (67, 73)]
[(3, 139), (5, 137), (11, 131), (29, 113), (41, 101), (53, 89), (59, 83), (71, 71)]
[(5, 139), (7, 137), (13, 131), (17, 127), (31, 113), (37, 107), (41, 103), (43, 101), (47, 97), (61, 83), (71, 73)]
[(7, 139), (19, 127), (37, 109), (43, 103), (67, 79), (73, 73)]
[(11, 137), (17, 131), (41, 107), (47, 101), (59, 89)]
[(11, 139), (13, 137), (19, 131), (23, 127), (37, 113), (41, 109), (43, 107), (47, 103), (53, 97), (61, 89), (67, 83), (71, 79)]
[(3, 149), (13, 139), (43, 109), (73, 79)]
[(3, 151), (5, 149), (17, 137), (23, 131), (41, 113), (47, 107), (53, 101), (71, 83)]
[(5, 151), (7, 149), (17, 139), (19, 137), (29, 127), (43, 113), (47, 109), (53, 103), (59, 97), (67, 89), (73, 83)]
[(7, 151), (19, 139), (31, 127), (61, 97), (79, 79)]
[(3, 157), (11, 149), (23, 137), (29, 131), (47, 113), (53, 107), (59, 101), (71, 89)]
[(5, 157), (11, 151), (13, 149), (23, 139), (31, 131), (53, 109), (59, 103), (61, 101), (73, 89), (79, 83)]
[(7, 157), (13, 151), (37, 127), (61, 103), (67, 97)]
[(3, 163), (17, 149), (29, 137), (53, 113), (59, 107), (83, 83)]
[(5, 163), (11, 157), (17, 151), (19, 149), (29, 139), (31, 137), (37, 131), (41, 127), (59, 109), (61, 107), (67, 101), (71, 97), (79, 89)]
[(3, 167), (7, 163), (13, 157), (19, 151), (31, 139), (43, 127), (61, 109), (67, 103), (73, 97)]
[(5, 167), (23, 149), (41, 131), (59, 113), (71, 101), (83, 89)]
[(7, 167), (11, 163), (17, 157), (23, 151), (37, 137), (43, 131), (47, 127), (61, 113), (67, 107), (71, 103), (73, 101)]
[(3, 173), (13, 163), (19, 157), (37, 139), (67, 109), (73, 103), (79, 97)]
[(5, 173), (11, 167), (29, 149), (41, 137), (47, 131), (71, 107), (89, 89)]
[(7, 173), (13, 167), (17, 163), (23, 157), (29, 151), (31, 149), (41, 139), (43, 137), (53, 127), (67, 113), (71, 109), (73, 107), (79, 101), (83, 97)]
[(3, 179), (19, 163), (31, 151), (43, 139), (73, 109), (79, 103)]
[(3, 181), (5, 179), (11, 173), (17, 167), (47, 137), (53, 131), (71, 113), (83, 101)]
[(5, 181), (7, 179), (13, 173), (19, 167), (23, 163), (29, 157), (37, 149), (47, 139), (59, 127), (73, 113), (79, 107), (83, 103), (89, 97)]
[(7, 181), (31, 157), (37, 151), (61, 127), (79, 109)]
[(11, 179), (17, 173), (23, 167), (41, 149), (53, 137), (59, 131), (83, 107), (89, 101)]
[(11, 181), (13, 179), (19, 173), (29, 163), (41, 151), (43, 149), (53, 139), (61, 131), (79, 113), (83, 109), (89, 103)]
[(3, 191), (13, 181), (31, 163), (37, 157), (43, 151), (67, 127), (97, 97)]
[(3, 193), (5, 191), (17, 179), (23, 173), (29, 167), (47, 149), (59, 137), (83, 113), (89, 107)]
[(5, 193), (7, 191), (17, 181), (19, 179), (31, 167), (41, 157), (47, 151), (59, 139), (61, 137), (67, 131), (71, 127), (89, 109), (97, 101)]
[(3, 197), (7, 193), (19, 181), (37, 163), (43, 157), (61, 139), (73, 127), (97, 103)]
[(3, 199), (5, 197), (11, 191), (23, 179), (29, 173), (53, 149), (71, 131), (89, 113), (101, 101)]
```

In [60]:

```
from itertools import product
### Exercise 3
# Given N
def ex3(N):
    # 1) Create a ring
    R = Integers(N)

    # 2) Check if the unit group is cyclic
    UG = R.unit_group() # obtain the group of units
    print(UG.is_cyclic())

    # 3) Print the generators of the group of units
    print("Generators: ", UG.gens())

    # 4) Create a list containing all elements in the group
    # We need to compute {g_0^n_0*g_1^n_1*...*g_k^n_k | for all 0 <= i <= k and {0, 1 , ... n_i} }

    elements = []

    for gen in UG.gens(): # for each generator
        elements_of_gen = []
        for i in range(gen.order()): # creates a list of [g, g^2, g^3, ... , g^n]
            elements_of_gen.append(gen^i)
        elements.append(elements_of_gen)

    # compute the pairwise product of all items and store them in elements[0]
    # Note: this is not the efficient way to do this, it is better to use an iterator
    for i in range(len(elements)-1):
        elements[0] = [x * y for x, y in product(elements[0], elements[i+1])]

    print("Elements:", sorted(set(elements[0]))) # set() to remove duplicates

    # check if the elements are correct:
    # How it works: UG.list() returns the tuple of all elements
    # list(UG.list()) creates a list from the tuple and we compare it to our element list.
    assert(set(elements[0]) == set(list(UG.list())))


    # 5) Test the Lagrange theorem for a random element
    # How the test works: The order of the random element should divide the order of the group
    # i.e. n % (order of an element) = 0

    n = UG.order() # order of the group
    print("Lagrange theorem holds: ", n % UG.random_element().order() == 0)

ex3(12)
```

```
False
Generators:  (f0, f1)
Elements: [1, f1, f0, f0*f1]
Lagrange theorem holds:  True
```

In [46]:

```
### Exercise 4
def perfect_power(N):
    # We use the sage built in function for this
    # it returns (x, b) and b == 1 implies it is not a perfect power.
    x, b = N.perfect_power()
    if b == 1:
        return false
    else:
        return (x, b)

print("Exercise 4")
print(perfect_power(589700950065322297792301221688 23))
print(perfect_power(123^1237))
print(perfect_power(456456456))
```

```
Exercise 4
(34567, 7)
(123, 1237)
False
```

In [ ]: