Integer Factorization

Mehmet Acar

Advisor: Tülay Ayyıldız Akoğlu

17 May 2023

Project Definition

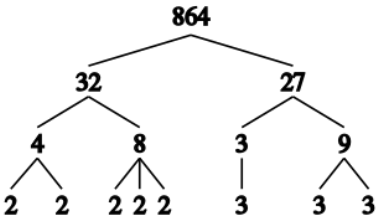
In number theory, integer factorization is the decomposition, when possible, of a positive integer into a product of smaller integers. If the factors are further restricted to be prime numbers, the process is called prime factorization, and includes the test whether the given integer is prime (in this case, one has a "product" of a single factor).

Project Definition

Factoring a positive integer n means finding positive integers p and q such that the product of p and q equals n and such that both p and q are greater than 1, p and q are called the factors of n and n = p . q is called a factorization of n.

Project Definition

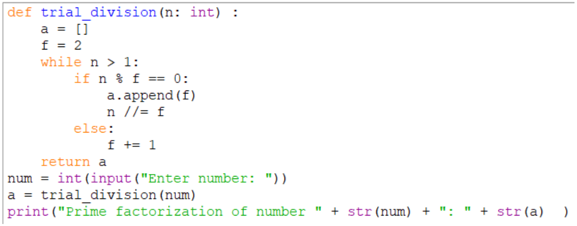
To illustrate integer factorization into prime factors, take for example the integer 864 which can be factored into two numbers 32 and 27, 32 can in turn be factorized into 4 and 8, whereas 27 can be factorized into 3 and 9, thus we have: $4 = 2 \times 2 = 2^2$, $8 = 2 \times 2 \times 2 = 2^3$ and $3 = 3^1$, $9 = 3 \times 3 = 3^2$. Then if we collect the factors together the prime factorization of 864 can be written as $2^5 \times 3^3$. This can also be illustrated by the image below



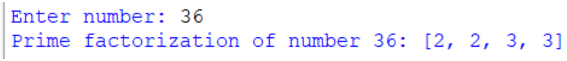
Project Design - Trial Division Algorithm

Trial division is the most laborious but easiest to understand of the integer factorization algorithms. The essential idea behind trial division tests to see if an integer n, the integer to be factored, can be divided by each number in turn that is less than n. For example, for the integer n = 12, the only numbers that divide it are 1, 2, 3, 4, 6, 12. Selecting only the largest powers of primes in this list gives that $12 = 3 \times 4 = 3 \times 2^2$.
Example: $18 = 2 \times 3 \times 3$ So prime decomposition of 18 is 2, 3, 3

Trial Division Algorithm Example



```
def trial_division(n: int) :
    a = []
    f = 2
    while n > 1:
        if n % f == 0:
            a.append(f)
            n //= f
        else:
            f += 1
    return a
num = int(input("Enter number: "))
a = trial_division(num)
print("Prime factorization of number " + str(num) + ": " + str(a)   )
```

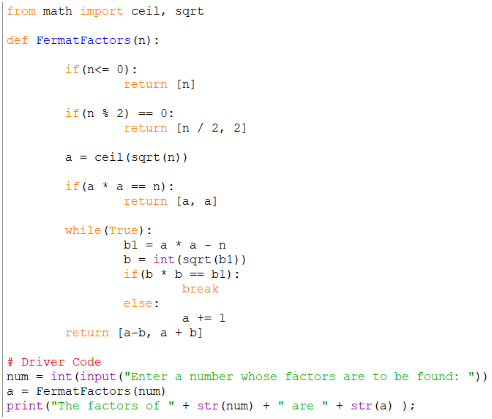


```
Enter number: 36
Prime factorization of number 36: [2, 2, 3, 3]
```

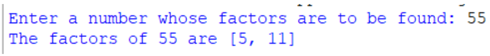
Project Design - Fermat's Factorization Method

- Fermat's factorization method relies on the fact that every odd number can be represented as a difference of squares of two numbers. That is,
- N = X^2- Y^2 = (X + Y) * (X - Y). Here 'X' is greater than 'Y' and (x + y) and (x - y) are factors of N.
- We start with finding an integer 'K' such that K * K is greater than N.
- Then we find the difference between K * K and N. Let the difference be denoted as D.
- If D is a perfect square, then we stop. Let S be the square root of D. Therefore, our answer is given by "S * S - K * K". As a result, factors of N are given by (S - K) and (S + K).

Fermat's Factorization Method Example



```python
from math import ceil, sqrt

def FermatFactors(n):

        if(n<= 0):
                return [n]

        if(n % 2) == 0:
                return [n / 2, 2]

        a = ceil(sqrt(n))

        if(a * a == n):
                return [a, a]

        while(True):
                b1 = a * a - n
                b = int(sqrt(b1))
                if(b * b == b1):
                        break
                else:
                        a += 1
        return [a-b, a + b]

# Driver Code
num = int(input("Enter a number whose factors are to be found: "))
a = FermatFactors(num)
print("The factors of " + str(num) + " are " + str(a) );
```



```
Enter a number whose factors are to be found: 55
The factors of 55 are [5, 11]
```

Project Requirements - 1

- Literature review about integer factorization
- Research integer factorization algorithms
- Decide an algorithm for improvement
- Implement an improved algorithm in Python
- Preparing a GUI for user

Project Requirements - 2

Software Requirements
- I decide to the implement selected algorithm in Python which has version 3.10.1

Hardware Requirements
- No hardware requirements are needed

Success Criteria

- Factorize an integer to the prime factors with selected algorithm
- Adding some new parts to the selected algorithm logically
- GUI should be work for user

Selected Algorithm for İmprovement

I selected Pollard's Rho algorithm for improvement. Pollard's Rho algorithm is one of the most-used integer factorization algorithms. This is because its running time complexity is proportional to the square root of the size of the smallest prime factor and the amount of space used to execute the algorithm is much less than others. There are other methods for prime factorization but we prefer Pollard's Rho because we don't have to test all possible integers until a divisor is found, which means it will improve the time complexity. This algorithm is more efficient than Fermat's and wheel factorizations when comparing time and space complexity.

How Pollard’s Rho Algorithm Works

1. Start with random x and c. Take y equal to x and $f(x) = x^2 + c$.
2. While a divisor isn’t obtained
    1. Update x to f(x) (modulo n) [Tortoise Move]
    2. Update y to f(f(y)) (modulo n) [Hare Move]
    3. Calculate GCD of |x-y| and n
    4. If GCD is not unity
        1. If GCD is n, repeat from step 2 with another set of x, y and c
        2. Else GCD is our answer

Pollard's Rho Algorithm Example

Let us suppose $n = 187$ and consider different cases for different random values.
An Example of random values such that algorithm finds result: $y = x = 2$ and $c = 1$, Hence, our $f(x) = x^2 + 1$.
In the first step, we calculate $f(x)$ and $f(f(y))$ values
$f(x) = 2^2 + 1 = 5$
$f(f(y)) = f(2^2 + 1) = f(5) = 5^2 + 1 = 26$
Then, we applying module operation to the $f(x)$ and $f(f(y))$ numbers.
$f(x) \pmod{n} = 5 \pmod{187} = 5$
$f(f(y)) \pmod{n} = 26 \pmod{187} = 26$
Later, we update our x and y values with $f(x) \pmod{n}$ and $f(f(y)) \pmod{n}$ values. So, our new x value is 5 and our new y value is 26.

Pollard's Rho Algorithm Example

Then we calculate GCD value. In order to find GCD value, calculate the GCD of |x-y| and n.

GCD(|x-y|,n) = GCD(|5-26|,187) = GCD(21,187) = 1

Because of GCD value is equal to 1, we continue to the algorithm. In order to find the one prime factor of the input number(n), our GCD value should not be 1 or n.

Now, we calculate f(x) and f(f(y)) values again with new x and y values

$f(x) = 5^2 + 1 = 26$

$f(f(y)) = f(26^2 + 1) = f(677) = 677^2 + 1 = 458330$

Then, we applying module operation to the f(x) and f(f(y)) numbers.

$f(x) \pmod{n} = 26 \pmod{187} = 26$

$f(f(y)) \pmod{n} = 458330 \pmod{187} = 180$

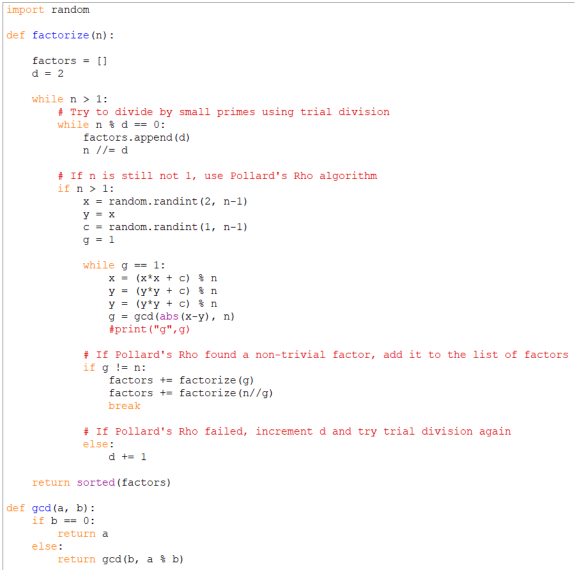
Pollard's Rho Algorithm Example

Later, we update our x and y values with f(x) (mod n) and f(f(y)) (mod n) values. So, our new x value is 26 and our new y value is 180.
Then we calculate GCD value. In order to find GCD value, calculate the GCD of |x-y| and n.
GCD(|x-y|,n) = GCD(|26-180|,187) = GCD(154,187) = 11
Because of GCD value is equal to 11, we find the one prime factor of input number(n). Other prime factor of n is calculated with division of n with founded first prime factor. So, other prime factor is equal to 187/11 which is 17.

Improvements On the Pollard's Rho Algorithm

If the number n had more than two prime factors, and the division yielded another composite number, we could break that number into its prime factors by again going through the algorithm.
And, in order to do this I combine Pollard's Rho algorithm with Trial Division algorithm.

Improved Code of Pollard’s Rho Algorithm



```python
import random

def factorize(n):

    factors = []
    d = 2

    while n > 1:
        # Try to divide by small primes using trial division
        while n % d == 0:
            factors.append(d)
            n //= d

        # If n is still not 1, use Pollard's Rho algorithm
        if n > 1:
            x = random.randint(2, n-1)
            y = x
            c = random.randint(1, n-1)
            g = 1

            while g == 1:
                x = (x*x + c) % n
                y = (y*y + c) % n
                y = (y*y + c) % n
                g = gcd(abs(x-y), n)
                #print("g",g)

            # If Pollard's Rho found a non-trivial factor, add it to the list of factors
            if g != n:
                factors += factorize(g)
                factors += factorize(n//g)
                break

            # If Pollard's Rho failed, increment d and try trial division again
            else:
                d += 1

    return sorted(factors)

def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

Improved Code of Pollard's Rho Algorithm

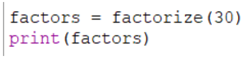
This code implements a function called factorize which takes an integer n as input and returns a sorted list of its prime factors. The function first tries to divide n by small primes using trial division, and then uses Pollard's Rho algorithm if n is still not 1. The implementation of the factorize function relies on another helper function called gcd, which computes the greatest common divisor of two numbers.
The factorize function first initializes an empty list called factors to store the prime factors. It then initializes a variable d to 2 and enters a loop that runs while n is greater than 1. Inside the loop, it uses trial division to check if n is divisible by d. If it is, it appends d to the list of factors, updates n by dividing it by d, and continues trying to divide n by d until it is no longer divisible.

Improved Code of Pollard’s Rho Algorithm

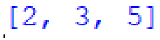
If n is still greater than 1 after the trial division step, Pollard’s Rho algorithm is used to find a non-trivial factor of n. The algorithm generates two random numbers x and y and calculates their modular squares until a greatest common divisor g is found that is greater than 1 and less than n. If g is not equal to n, the function recursively calls factorize() on g and n//g to find their prime factors and appends them to the list of factors.
If Pollard’s Rho algorithm fails to find a non-trivial factor, d is incremented and trial division is tried again.
The function finally returns a sorted list of prime factors of n.

Test Results of Improved Code

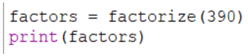
I tested the code with different integers and results are given in the below figures.



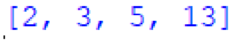
```
factors = factorize(30)
print(factors)
```



```
[2, 3, 5]
```



```
factors = factorize(390)
print(factors)
```



```
[2, 3, 5, 13]
```

Project Design Plan

- Until now, I selected one integer factorization algorithm which is Pollard's Rho algorithm and I add some parts to the this algorithm.
- At the rest of of the semester, first of all I focus on large integers. I try to find the prime factors of large integers.
- Then, I prepare a graphical user interface for taking input number from user. Then, I printed prime factors of input to the screen in interface.

References I

[One] [Two] [Thr] [Fou]

- *Fermat's factorization method*, https://www.codingninjas.com/codestudio/library/fermat-s-factorization-method.
- *Integer factorization*, https://en.wikipedia.org/wiki/Integer_factorization#:~:text=In%20number%20theory%2C%20integer%20factorization,a%20product%20of%20smaller%20integers.
- *Integer factorization algorithms*, https://iq.opengenus.org/integer-factorization-algorithms/.
- *Integer factorization algorithms*, https://www.diva-portal.org/smash/get/diva2:1460632/FULLTEXT01.pdf.