Lezione4

Euclidean algorithm

An algorithm used to compute, in a efficient way, the greatest common divisor between two numbers.

```
def euclid(c: int, d: int) -> int:
        while d != 0:
                swap_variable = c % d
                c = d
                d = swap_variable
        return c
```

The Euclidean algorithm finds the greatest common divisor (GCD) of two numbers by repeatedly dividing the larger number by the smaller one and replacing the larger with the remainder until the remainder is zero. The last non-zero remainder is the GCD.

Extended Euclidean Algorithm

An extension of the Euclidean algorithm that computes, in addition to the greatest common divisor (GCD) of integers a and b, the coefficients of Bézout's identity, i.e., integers x and y such that:

$$ax + by = \gcd(a, b)$$

In our use case, we need it to compute the modular inverse of a number. The modular inverse of `a` modulo `b` is the integer `x` such that:

$$ax \equiv 1 \pmod{b}$$

This only exists if $\gcd(a, b) = 1$.

```
def extended_euclid(a: int, b: int) -> int:
    original_b = b
    x0, x1 = 1, 0  # Coefficients for Bézout's identity
    while b != 0:
        quotient = a // b  # Integer division
        remainder = a - quotient * b  # This is a % b
        a, b = b, remainder

        # Update the coefficients for Bézout's identity
        x0, x1 = x1, x0 - quotient * x1
```

```
    # If gcd(a, b) is 1, we have an inverse
    if a == 1:
        return x0 % original_b  # Ensure the result is positive modulo b
```

The **Extended Euclidean Algorithm** works similarly to the standard Euclidean algorithm but keeps track of additional variables x and y as it performs the divisions. These values are used to express the GCD as a linear combination of the original inputs. Additionally, when $\gcd(a, b) = 1$, it provides the modular inverse, which is used in cryptographic algorithms and other number theory applications.