INF 245 Homework 1A

By Vegard Berge

Task 1) Implement Extended Euclidean Algorithm.

Let:

> ⚫ a = 620709603821307061, b = 390156375246520685

find d = $d = gcd(a, b)$ and integers u, v such that $d = ua + vb$

```
def ExtendedEuclideanAlgorithm(a,b):
  """ Recursively finds the coeffisiants u, v s.t gcd(a, b) = ua + vb """
    ifa== 0:
        returnb, 0, 1
    gcd, u_1, v_1 = ExtendedEuclideanAlgorithm(b%a,a)
    return gcd, v_1 - (b//a) * u_1, u_1
```

> 💡 *I also implemented a small verification function in order to check the results:*
>
> ```
> def VerifyEEA(a, b, u, v) -> bool:
>     return GCD(a, b) == u*a + v*b
> ```

> Also implemented GCD to check answer, code here:

```
def GCD(a, b):  # rec. find modulus until b exponent == 0
    if b == 0: return abs(a)
    else: return GCD(b, a % b)
```

> Result of task is: (EEA = Extended Euc. Algo)

```
EEA: 1299709 u is -128541328501 v is 204499636942
Does u * 620709603821307061 + v * 390156375246520685 = GCD(a, b)?: True
```

Task 2) Implement binary exponentiation modulo n.

Comput $b = a^m mod\ n$ for $(a, m, n) = (393492946341, 103587276991, 72447943125)$

Since modulo operation dont interfere with multiplication we can have:

$$a * b \equiv (a \ mod \ m) * (b \ mod \ m) \ mod \ m$$

```
def BinaryExponentiationWithoutRecursion(a: int,b: int,mod: int):

    # Modulo function to perform binary exponentiation - without recursion
    temp, base_number = 1, a
    while b > 0:  # exponent larger than zero
        if b % 2 == 1:  # if not even
            temp = (temp * base_number) % mod  # pow(x*y, 1, mod)
        base_number, b = (base_number * base_number) % mod, b // 2
    return temp % mod
```

> Output from program:

```
Binary exponentiation modulo n: 49107059316

Process finished with exit code 0
```

Task 3) Implement elimination algorithm (reduce to row echelon form) and solve system of linear congruence:

$$\begin{bmatrix} 1 & -2 & -2 & -2 & -1 \\ 0 & 3 & -2 & -3 & 1 \\ 3 & 0 & 0 & 1 & -1 \\ 3 & -3 & -2 & 0 & 1 \\ 0 & -3 & 3 & -3 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \equiv \begin{bmatrix} 2 \\ 3 \\ 2 \\ 1 \\ 2 \end{bmatrix} mod \ n$$

$n = 456995412589$

💡 I think code is well documented enough, for a simple row echelon form algorithm:

```
def RowReduceEchelonForm(m: list, modulus: int) -> Type[list] | list:
    # A is concat of matrix A and vector
    # Our matrix M = m x (t + 1)
    MATRIX = copy.deepcopy(m)
    c, r = 0, 0
    while r != len(MATRIX):
        # find pivot el
        Mij = MATRIX[r][c]
        z = 0
        d = GCD(Mij, modulus)
        if d > 1:  # if GCD(Mij, N) > 1, terminate
            return list
        if d == 1:  # if GCD is one, find Z * b congruent with 1 mod N
            z = pow(Mij, -1, modulus)
            # apply z to all elements of row
            MATRIX[r] = [MATRIX[r][x] * z % modulus for x in range(len(m[0]))]
        # pivot element completely divides modulo
        if Mij % modulus == 0:
```

```
            # switch rows r + 1
            for x in range(r + 1, len(MATRIX)):
                if MATRIX[x][Mij] != 0 % modulus:
                    MATRIX[r], MATRIX[r + 1] = MATRIX[r + 1], MATRIX[r]
                # already switched, now reduce

        # make zero space for r + 1 under pivot element
        for e in range(r + 1, len(MATRIX)):
            # apply zero element mult. to all elements of row r + 1
            MATRIX[e] = [(MATRIX[e][x] - MATRIX[e][r] % modulus * MATRIX[r][x]) % modulus for x in range(len(m[0]))]
        Mij, c, r = Mij + 1, c + 1, r + 1
    return MATRIX
```

Task 4) Implement the algorithm to compute Jacobi symbol (a/n), where a is an integer and n is an odd positive integer. Comput

$$(-776439811/50556018318800449023)$$

For this code I use the properties of Jacobi Symbol:

Let n be an odd positive integer and $n = \prod_{i=1}^{k}(a/p_i)_i^e$

I follow lecture notes, and have that by agreement $(a/1) = 1$. If $gcd(a, n) > 1$ then $(a/p_i) = 0$ for some p_i so, by the definition (above) we have that $(a/n) = 0$. In my implementation I use some of the eight/nine rules of Jacobi symbol:

1. If $a \equiv b(modn), => (a/n) = (b/n)$
2. $(m/n)(n/m) = (-1)^{\frac{n-1}{2}\ \frac{m-1}{2}}$
3. $(2/n) = (-1)^{\frac{n^2-1}{8}}$
4. $(-1/n) = (-1)^{\frac{n-1}{2}}$

```
def JacobiSymbol(a, n, d=1):
    if a == 0:
        return 0  # (0/n) = 0
    if a == 1:
        return d  # (1/n) = 1

    if a < 0:
        # property of Jacobi
        # (a/n) = (-a/n)*(-1/n)
        a = -a
        if n % 4 == 3:
            # (-1/n) = -1 if n = 3 (mod 4)
            d = -d

    while a:
        if a < 0:
            # (a/n) = (-a/n)*(-1/n)
            a = -a
            if n % 4 == 3:
                # (-1/n) = -1 if n = 3 (mod 4)
                d = -d
        while a % 2 == 0:
            a = a // 2
            if n % 8 == 3 or n % 8 == 5:
                d = -d
        # swap
        a, n = n, a
        if a % 4 == 3 and n % 4 == 3:
```

```
            d = -d
        a = a % n
        if a > n // 2:
            a = a - n
    if n == 1:
        return d
    return 0
```

Output of the algorithm is:

```
C:/Users/Veggi/Documents/UiB/INF245/INF245/MandatoryAssignment1
Jacobi of -776439811 and 505560183188OO449023 is: -1

Process finished with exit code 0
```

Task 5) Implement Solovay-Strassen test to check the primality of an odd positive integer n. Prove that $n = 2^{127} - 1$ is a probable prime with the error probability $< 1/2^{20}$

> 💡 *Here I use some of the functions implemented earlier in the task!*

```
def Solovay_Strassen_Test(n, k=20) -> str:
    """
    :input: n, a value to test primality
    :out: composite if test fails, probably prime else
    :rtype: str
    """
    # check n odd prime > 1
    assert (n > 1)
    import math
    for i in range(k):
        # Solovay strassen test:
        # 1) gcd(a, n) != 1 => composite
        # 2) a ** (n-1) / 2 congruent with jacobi(a/n) mod n
        a = random.randint(2, n)  # random is ge && le
        if GCD(a, n) != 1:  # save compute time if gcd(a, n) != 1
            return "Composite"
        x = (n + JacobiSymbol(a, n)) % n
        # a ** ((n - 1) / 2) can be re - written to our bin. exp method as:
        mod = BinaryExponentiationWithoutRecursion(a, (n - 1) // 2, n)
        if (x == 0) or mod != x:
            return "Composite"
    return "Probably Prime"
```