

CRYPTOGRAPHY AND NETWORK SECURITY

LAB 3 : WRITE THE CODE AND EXECUTE FOR THE FOLLOWING ALGORITHMS:

- **1. CHINESE REMAINDER THEOREM**
- **2. EXTENDED EUCLIDEAN ALGORITHM**

NAME : OM SUBRATO DEY

REGISTER NO.: 21BAI1876

1. CHINESE REMAINDER THEOREM:

CODE:

```
def extended_euclidean(a, b):
    """
    Returns a tuple (g, x, y) such that  $g = \gcd(a, b)$  and  $ax + by = g$ 
    """
    if a == 0:
        return b, 0, 1
    else:
        g, x1, y1 = extended_euclidean(b % a, a)
        x = y1 - (b // a) * x1
        y = x1
        return g, x, y

def modular_inverse(a, m):
    """
    Returns the modular inverse of a under modulo m, if it exists
    """
    g, x, _ = extended_euclidean(a, m)
    if g != 1:
        raise ValueError(f"Modular inverse does not exist for {a} and {m}")
    else:
        return x % m

def chinese_remainder_theorem(n, a):
    """
    Solves the system of simultaneous congruences using the Chinese
    Remainder Theorem.
    n: List of moduli
    a: List of remainders
    """
```

```

Returns the smallest x such that  $x \equiv a[i] \pmod{n[i]}$  for all i
"""

product = 1
for ni in n:
    product *= ni

result = 0
for ni, ai in zip(n, a):
    pi = product // ni
    mi = modular_inverse(pi, ni)
    result += ai * mi * pi

return result % product

# Example usage:
n = [1, 8, 7]
a = [6, 2, 1]
x = chinese_remainder_theorem(n, a)
print(f"The solution to the system of congruences is  $x \equiv \{x\} \pmod{\{n[0] * n[1] * n[2]\}}$ ")

```

OUTPUT:

```

main.py
1 # @SUBMIT DEV - 210411974
2 def extended_euclidean(a, b):
3     """
4     Returns a tuple (g, x, y) such that g = gcd(a, b) and ax + by = g
5     """
6     if a == 0:
7         return b, 0, 1
8     else:
9         g, x1, y1 = extended_euclidean(b % a, a)
10        x = y1 - (b // a) * x1
11        y = x1
12        return g, x, y
13
14 def modular_inverse(a, m):
15     """
16     Returns the modular inverse of a under modulo m, if it exists
17     """
18     g, x, _ = extended_euclidean(a, m)
19     if g != 1:
20         raise ValueError("Modular inverse does not exist for {a} and {m}")
21     else:
22         return x % m
23
24 def chinese_remainder_theorem(n, a):
25     """
26     Solves the system of simultaneous congruences using the Chinese Remainder Theorem.
27     n: List of moduli
28     a: List of remainders
29     Returns the smallest x such that x = a[i] (mod n[i]) for all i
30     """
31     product = 1
32     for ni in n:
33         product *= ni
34
35     result = 0
36     for ni, ai in zip(n, a):
37         pi = product // ni
38         mi = modular_inverse(pi, ni)
39         result += ai * mi * pi
40
41     return result % product
42
43 # Example usage:
44 n = [1, 8, 7]
45 a = [6, 2, 1]
46 x = chinese_remainder_theorem(n, a)
47 print(f"The solution to the system of congruences is x = {x} (mod {n[0] * n[1] * n[2]})")
48

```

Output

```

The solution to the system of congruences is x = 50 (mod 56)
=== Code Execution Successful ===

```

2. EXTENDED EUCLIDEAN ALGORITHM:

CODE:

```
# OM SUBRATO DEY - 21BAI1876
```

```
def extended_gcd(a, b):
```

```
    # Base case
```

```
    if a == 0:
```

```
        return b, 0, 1
```

```
    # Recursively call the function
```

```
    gcd, x1, y1 = extended_gcd(b % a, a)
```

```
    # Update x and y using results of recursive call
```

```
    x = y1 - (b // a) * x1
```

```
    y = x1
```

```
    return gcd, x, y
```

```
# Example usage
```

```
a = 212021
```

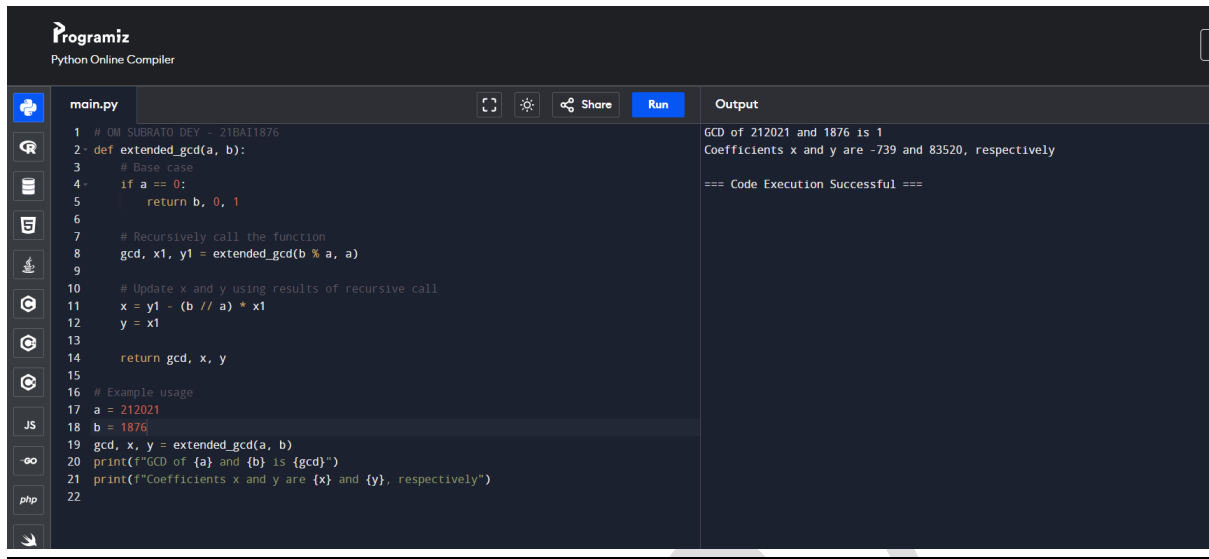
```
b = 1876
```

```
gcd, x, y = extended_gcd(a, b)
```

```
print(f"GCD of {a} and {b} is {gcd}")
```

```
print(f"Coefficients x and y are {x} and {y}, respectively")
```

OUTPUT:



The screenshot displays the Programiz Python Online Compiler interface. The editor on the left contains a Python script named `main.py` that implements the extended Euclidean algorithm. The script defines a function `extended_gcd(a, b)` which returns the greatest common divisor (gcd) and coefficients `x` and `y` such that `ax + by = gcd`. The script includes comments explaining the base case, recursive call, and the update of `x` and `y`. It also shows an example usage with `a = 212021` and `b = 1876`. The output on the right shows the result of the execution: "GCD of 212021 and 1876 is 1" and "Coefficients x and y are -739 and 83520, respectively". The execution was successful.

```
1 # OM SUBRATO DEV - 21BA11876
2 def extended_gcd(a, b):
3     # Base case
4     if a == 0:
5         return b, 0, 1
6
7     # Recursively call the function
8     gcd, x1, y1 = extended_gcd(b % a, a)
9
10    # Update x and y using results of recursive call
11    x = y1 - (b // a) * x1
12    y = x1
13
14    return gcd, x, y
15
16 # Example usage
17 a = 212021
18 b = 1876
19 gcd, x, y = extended_gcd(a, b)
20 print(f"GCD of {a} and {b} is {gcd}")
21 print(f"Coefficients x and y are {x} and {y}, respectively")
22
```

Output

```
GCD of 212021 and 1876 is 1
Coefficients x and y are -739 and 83520, respectively

=== Code Execution Successful ===
```