CECS 229: Programming Assignment #2

Due Date:

Sunday, 2/25 @ 11:59 PM

Submission Instructions:

Complete the programming problems in the file named `pa2.py` . You may test your implementation on your Repl.it workspace by running main.py . When you are satisfied with your implementation,

1. Submit your Repl.it workspace
2. Download the file `pa2.py` and submit it to the appropriate CodePost auto-grader folder.

Objectives:

1. Use the Sieve of Eratosthenes to find all primes in a given range.
2. Design a computational algorithm for finding the Bézout coefficients of two integers.
3. Use Bézout coefficients to calculate the GCD.

NOTE:

Unless otherwise stated in the FIXME comment, you may not change the outline of the algorithm provided by introducing new loops or conditionals, or by calling any built-in functions that perform the entire algorithm or replaces a part of the algorithm.

Problem 1:

Complete the function `primes(a, b)` which uses the Sieve of Eratosthenes to create and return a list of all the primes p satisfying $a \leq p \leq b$.

- INPUT:
  - `a` - a positive integer greater than or equal to 1 (`ValueError` if an integer less than 1 is given); the lower bound
  - `b` - a positive integer greater than or equal to `a` (`ValueError` if `b` < `a`); the upper bound
- OUTPUT:
  - a **set** of all the primes p satisfying `a` $\leq p \leq$ `b`

EXAMPLE:

`>> primes(1, 10)`

`{2, 3, 5, 7}`

`>> primes(50, 100)`

`{53, 59, 61, 67, 71, 73, 79, 83, 89, 97}`

NOTE: The order of the elements might be different in your output in comparison to the expected output, and that is okay! Your output is correct as long as you have all the primes.

In [1]:
```
def primes(a, b):
    if a < 1 or b < a: # handling invalid range
        raise ValueError("Invalid range given")

    if a == 1: # handling starting point a = 1
        a = 2  # this ensures 1 is not listed as a prime

    # FIXME: initialize `stop` which is the stopping criteria for
    #        the loop in the Sieve of Eratosthenes
    stop = "FIXME: Replace this string"

    # FIXME: initialize a Python set called `P` that contains
    #        all integers in the range [a, b]
    P = set("FIXME: Replace this string")

    for x in range(2, stop):

        # FIXME: use Python list comprehension to create a set
        #        of multiples of x in the range [2, b];
        # HINT: the set of multiples of x can be expressed as
        #       k * x, where k is an integer; hence the comprehension should
        #       loop over values that satisfy k * x <= b
        multiples_x = set("FIXME: replace this string")

        P -= multiples_x  # removing the multiples of x from the set P
    return P
```

Problem 2:

Complete the function `bezout_coeffs(a, b)` that computes the Bezout coefficients `s` and `t` of `a` and `b`.

1. INPUT:
   - `a` , `b` - distinct integers
2. OUTPUT: `{a: s, b: t}` - dictionary where keys are the input integers and values are their corresponding Bezout coefficients.

EXAMPLE: `>> bezout_coeffs(414, 662)`

`{414 : 8, 662 : -5}`

NOTE:

The algorithm for the function `bezout_coeff(a,b)` uses the logic employed in the following example:

Suppose $a = 13, \;\; b = 21$. We seek s and t such that gcd$(13, 21) = 13s + 21t$

Let's begin by defining $s_0 = 1, \;\; t_0 = 0, \;\; a_1 = 13, \;\; b_1 = 21$. At every round in attempting to attain the gcd, we will refer to s_i and t_i as the current coefficients of 13 and 21, respectively.

Round 1:

$21 = 1 \cdot 13 + 8$

$$\implies 8 = 21 - 1 \cdot 13 \text{ We will call this EQN 1}$$

$$\implies s_1 = -1, \quad t_1 = 1$$

NOTICE:

$$s_1 = -\; (\; b_1 \textbf{ div } a_1 \;) = -(21 \textbf{ div } 13) = -1$$

Round 2:

$a_2 = 8, \;\; b_2 = 13$

$13 = 1 \cdot 8 + 5$

$$\implies 5 = 13 - 1 \cdot 8$$

$$= 1 \cdot 13 - 1(21 - 1 \cdot 13) \text{ from EQN 1}$$

$$= 2 \cdot 13 - 1 \cdot 21$$

$$\implies s_2 = 2, \quad t_2 = -1$$

NOTICE:

$$s_2 = s_0 - s_1 \;(\; b_2 \textbf{ div } a_2)$$

$$= 1 - 1 \;(\; 13 \textbf{ div } 8)$$

$$= 1 - \;(-1)(1)$$

$$= 2$$

$$t_2 = t_0 - t_1 \;(\; b_2 \textbf{ div } a_2)$$

$$= 0 - 1 \;(\; 13 \textbf{ div } 8)$$

$$= 0 - 1 \;(1)$$

$$= -1$$

Round 3:

$a_3 = 5, \;\; b_3 = 8$

$8 = 1 \cdot 5 + 3$

$$\implies 3 = 8 - \underbrace{1}_{b_3 \textbf{ div } a_3} \cdot 5$$

$$= 1 \cdot (\underbrace{1}_{t_1} \cdot 21 \underbrace{-1}_{s_1} \cdot 13) - \underbrace{1}_{b_3 \textbf{ div } a_3} (\underbrace{2}_{s_2} \cdot 13 \underbrace{-1}_{t_2} \cdot 21)$$

$$= -3 \cdot 13 + 2 \cdot 21$$

$$\implies s_3 = -3, \quad t_3 = 2$$

NOTICE:

$$s_3 = s_1 - s_2 \; (\; b_3 \textbf{ div } a_3)$$
$$= -1 - (2)(1)$$
$$= -3$$

$$t_3 = t_1 - t_2 \; (\; b_3 \textbf{ div } a_3)$$
$$= 1 - (-1)(1)$$
$$= 2$$

$\vdots$

Round k:

For any round $k \geq 2$, the corresponding s_k and t_k values are given by

- $s_k = s_{k-2} - s_{k-1} \; (\; b_k \textbf{ div } a_k)$
- $t_k = t_{k-2} - t_{k-1} \; (\; b_k \textbf{ div } a_k)$

You should verify for yourself that for any a, b,

- $s_0 = 1$
- $t_0 = 0$
- $s_1 = -(\; b \textbf{ div } a)$
- $t_1 = 1$

In [22]:
```
def bezout_coeffs(a, b):
    """
    computes the Bezout coefficients of two given positive integers
    :param a: int type; positive integer
    :param b: int type; positive integer
    :returns: dict type; a dictionary with parameters a and b as keys,
```

```
            and their corresponding Bezout coefficients as values.
    :raises: ValueError if a < 0 or b < 0
    """
    if a < 0 or b < 0:
      raise ValueError(
        f"bezout_coeffs(a, b) does not support negative arguments.")
    s0 = 1
    t0 = 0
    s1 = -1 * (b // a)
    t1 = 1

    temp = b
    bk = a
    ak = temp % a

    while ak != 0:
      temp_s = s1
      temp_t = t1

      # FIXME: Update s1 according to the formula for sk
      s1 = "FIXME: Replace this string"

      # FIXME: Update t1 according to the formula for tk
      t1 = "FIXME: Replace this string"

      s0 = temp_s
      t0 = temp_t
      temp = bk

      # FIXME: Update bk and ak
      bk = "FIXME: Replace this string"
      ak = "FIXME: Replace this string"

    # FIXME: Replace each string with the correct coefficients of a and b
    return {a: "FIXME: replace this string", b: "FIXME: replace this string"}
```

Problem 3:

Create a function `gcd(a, b)` that computes the greatest common divisor of `a` and `b` using the `bezout_coeff` function you implemented for problem 2 lecture. No credit will be given to functions that employ any other implementation. For example, using the built-in function `math.gcd()` as part of our implementation will not receive any credit.

INPUT:

```
* `a`,`b` - integers
```

OUTPUT: `d` - the gcd

EXAMPLE:

```
>> gcd(414, 662)
```

```
2
```

HINT

The GCD of any two numbers must be positive by definition.

In [26]:
```python
def gcd(a,b):
    """
    computes the greatest common divisor of two given integers
    :param a: int type;
    :param b: int type;
    :returns: int type; the gcd of a and b
    """
    A = abs(a)
    B = abs(b)
    if A == B:
        pass # FIXME: replace this pass with the correct return value
    bez = bezout_coeffs(A, B)
    return # FIXME: replace this pass with the correct return value
```

Problem 4:

Complete the function `mod_inv(a, m)` that returns the inverse of `a` modulo `m` that is in the range $[1, m-1]$. If the gcd of `a` and `m` is not 1, then the function raises a `ValueError` exception.

Example:

```
>> mod_inv(3, 11)
```

```
4
```

In [ ]:
```python
def mod_inv(a, m):
    """
    computes the inverse of a given integer a under a given modulo m
    :param a: int type; the integer of interest
    :param m: int type; the modulo
    :returns: int type; the integer in range [0, m) that is the inverse of a under mod
    :raises: ValueError if m < 0 or if a and m are not relatively prime
    """
    if m < 0:
        raise ValueError(f"mod_inv(a, m) does not support negative modulo m = {m}")
    g = gcd(a, m)
    if g != 1:
        raise ValueError(f"mod_inv(a, m) does not support integers that are not relati
    A = a
    while A < 0:

        A += """FIXME: replace this string so that by the end of the loop, A is in ran

    inverse = """FIXME: replace this string with the inverse of a under modulo m"""

    while inverse < 0:

        inverse += """FIXME: replace this string so that by the end of the loop, the i
```

```
    return inverse
```

Problem 5:

Complete the function `solve_mod_equiv(a, b, m, low, high)` that returns a set all integers x satisfying,

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

and,

$$low \leq x \leq high$$

If $\gcd(a, m) \neq 1$, `ValueError` is raised.

INPUT:

- `a`, `b`, `low`, `high` - integers
- `m` - positive integer, greater than 1

OUTPUT: Python set `{}` of all integers satisfying the conditions above.

EXAMPLE:

`>> solve_mod_equiv(3, 4, 7, -5, 5)`

`{-1}`

In [3]:
```
def solve_mod_equiv(a, b, m, low, high):
    """
    computes all solutions to the equivalence ax ≡ b (mod m)
    that are in the range [low, high]
    :param a: int type; the coefficient of the unknown value x
    :param b: int type; the integer that ax is equivalent to
              under modulo m
    :param m: int type; the modulo
    :param low: int type; the lower bound for the solutions
    :param high: int type; the upper bound for the solutions
    :raises: ValueError if high < low or if m < 0
    """
    if high < low:
        raise ValueError(f"solve_mod_equiv() does not support the upper bound {high} l
    if m < 0:
        raise ValueError(f"solve_mod_equiv() does not support negative modulo m = {m}'
    a_inv = mod_inv(a, m)

    k_low = """FIXME: replace this string with the correct lower bound for k, if x = m

    k_high = """FIXME: replace this string with the correct upper bound for k, if x =

    x = """FIXME: replace this string with the Python list comprehension that uses x =
    return set(x)
```