

Practical 3

Aim: Implementation of Extended Euclidean Algorithm.

Description:

The Extended Euclidean Algorithm is an extension of the standard Euclidean Algorithm. It not only computes the GCD of two numbers but also finds the coefficients x and y , known as the Bezout coefficients, which represent a linear combination of the inputs resulting in the GCD.

- **Key Concepts:**

- The GCD is calculated recursively by reducing the problem to smaller pairs of integers $(b, a \bmod b)$.
- At each step, the coefficients x and y are updated using the results from the recursive calls.

- **Applications:**

- Solving Diophantine equations.
- Computing modular inverses in number theory.
- Cryptographic algorithms like RSA.

Code:

1. ExtendedEuclidean.java

```
public class ExtendedEuclidean {
    // Helper class to store results
    static class Result {
        int gcd, x, y;

        Result(int gcd, int x, int y) {
            this.gcd = gcd;
            this.x = x;
            this.y = y;
        }
    }

    // Function implementing the extended Euclidean algorithm
    public static Result extendedEuclid(int a, int b) {
        if (b == 0) {
            return new Result(a, 1, 0); // Base case: gcd(a, 0) = a
        }

        // Recursive call
        Result result = extendedEuclid(b, a % b);

        // Update x and y using the results of the recursion
        int gcd = result.gcd;
        int x1 = result.x;
```

```

    int y1 = result.y;

    int x = y1;
    int y = x1 - (a / b) * y1;

    return new Result(gcd, x, y);
}

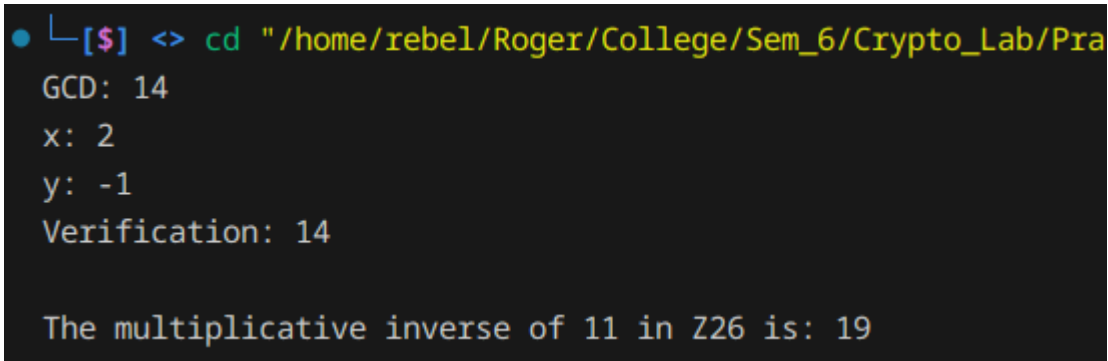
public static void main(String[] args) {
    int a = 56, b = 98;

    Result result = extendedEuclid(a, b);

    System.out.println("GCD: " + result.gcd);
    System.out.println("x: " + result.x);
    System.out.println("y: " + result.y);
    System.out.println("Verification: " + (a * result.x + b * result.y));
}
}

```

Output:



```

• L[$] <> cd "/home/rebel/Roger/College/Sem_6/Crypto_Lab/Pra
GCD: 14
x: 2
y: -1
Verification: 14

The multiplicative inverse of 11 in Z26 is: 19

```

Code Explanation:

1. Helper Class: Result

- A simple class to store the results of the algorithm, including the GCD and coefficients x and y.

2. Function: `extendedEuclid(int a, int b)`

- This function implements the recursive logic of the Extended Euclidean Algorithm:
 - **Base Case:** If $b=0$, the GCD is a , and $x=1, y=0$ (since $a \cdot 1 + b \cdot 0 = a$).
 - **Recursive Case:** Calls the function with $(b, a \bmod b)$ and updates the coefficients x and y using the results of the recursive call.
 - Computes:
 - $x=y1$

- $y = x_1 - (a/b) \cdot y_1$

3. Main Method

- Takes two integers a and b as input.
- Calls the extendedEuclid function to compute the GCD and coefficients x and y.
- Prints:
 - The GCD of a and b.
 - The coefficients x and y.
 - Verifies the result by calculating $a \cdot x + b \cdot y$.

4. Key Points:

- Recursive structure ensures efficient computation.
- The values of x and y are updated during back-substitution, ensuring the linear combination holds true.

Complexity Analysis:

Time Complexity

The **time complexity** of the Extended Euclidean Algorithm is the same as the standard Euclidean Algorithm, which is $O(\log(\min(a,b)))$.

Explanation:

- At each recursive step, the problem reduces from (a,b) to (b, a mod b).
- The size of b decreases significantly (approximately by half in binary representation) at each step.
- Thus, the number of recursive calls is proportional to the number of bits in the smaller number, min(a,b).

Space Complexity

The **space complexity** is $O(\log(\min(a,b)))$ due to the recursive call stack.

Explanation:

- The depth of the recursion is proportional to the number of steps in the algorithm, which is $O(\log(\min(a,b)))$.
- Each recursive call requires space on the stack, but no additional space is allocated for data structures. The coefficients x and y are computed in constant space during back-substitution.

Summary:

- **Time Complexity:** $O(\log(\min(a,b)))$
- **Space Complexity:** $O(\log(\min(a,b)))$