LECTURE 06

NUMBER THEORY I

MODULAR ARITHMETIC

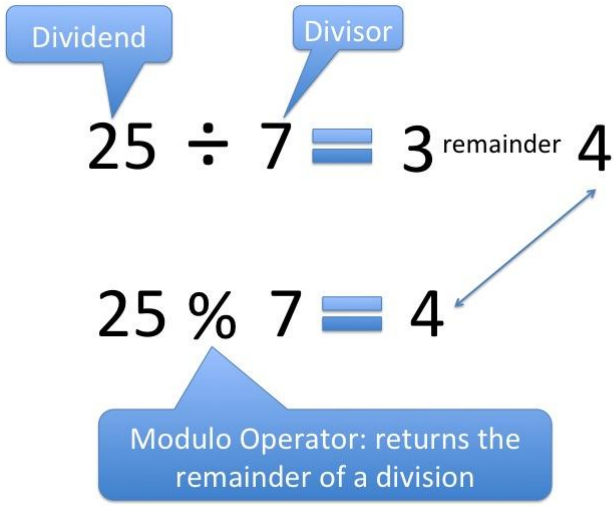
Big-O Coding

Website: www.bigocoding.com

Giới thiệu tổng quan

Modulo operation: Trong lập trình gọi là phép chia lấy dư.

Kí hiệu: %



Trong đó:

- **Dividend = 25:** số bị chia.
- **Divisor = 7:** số chia.
- **Quotient = 3:** thương.
- **Remainder = 4:** phần dư.

**** Lưu ý: 0 ≤ Remainder < Divisor*

Congruence relation

Congruence relation (Quan hệ đồng dư): Nếu 2 số nguyên a và b cùng chia cho m và có số dư giống nhau thì a và b gọi là đồng dư theo modulo (mô-đun) m.

Ký hiệu: $a \equiv b \pmod{m}$

Ví dụ: cho a = 11, b = 5, m = 3.

- 11 % 3 = 2.
- 5 % 3 = 2.

➔ a và b đồng dư theo modulo m = 3 ($11 \equiv 5 \pmod{3}$)

Một số tính chất phép modulo

1. Phép cộng: $(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$

Ví dụ: Tính $(10 + 5) \% 3$

$= ((10 \% 3) + (5 \% 3)) \% 3$

$= (1 + 2) \% 3$

$= 0$

2. Phép trừ: $(a - b) \bmod m = ((a \bmod m) - (b \bmod m)) \bmod m$

3. Phép nhân: $(a * b) \bmod m = ((a \bmod m) * (b \bmod m)) \bmod m$

4. Phép chia: $(a / b) \bmod m$ chỉ có nghĩa khi a chia hết b, phép chia **không** có tính chất phân phối như phép cộng, trừ và nhân. Để giải bài toán này dùng phương pháp **nghịch đảo modulo (phần sau)**.

Bài toán minh họa 1

Modular Exponentiation: là bài toán tính phép lũy thừa modulo. Tính phần dư phép lũy thừa của 2 số.

Ví dụ 1: Cho 3 số a, b và m. Hãy tính $(a^b)\ \%\ m$.

a = 2, b = 5, m = 3. Tính $(2^5)\ \%\ 3$.

Ta có: $(2^5)\ \%\ 3$

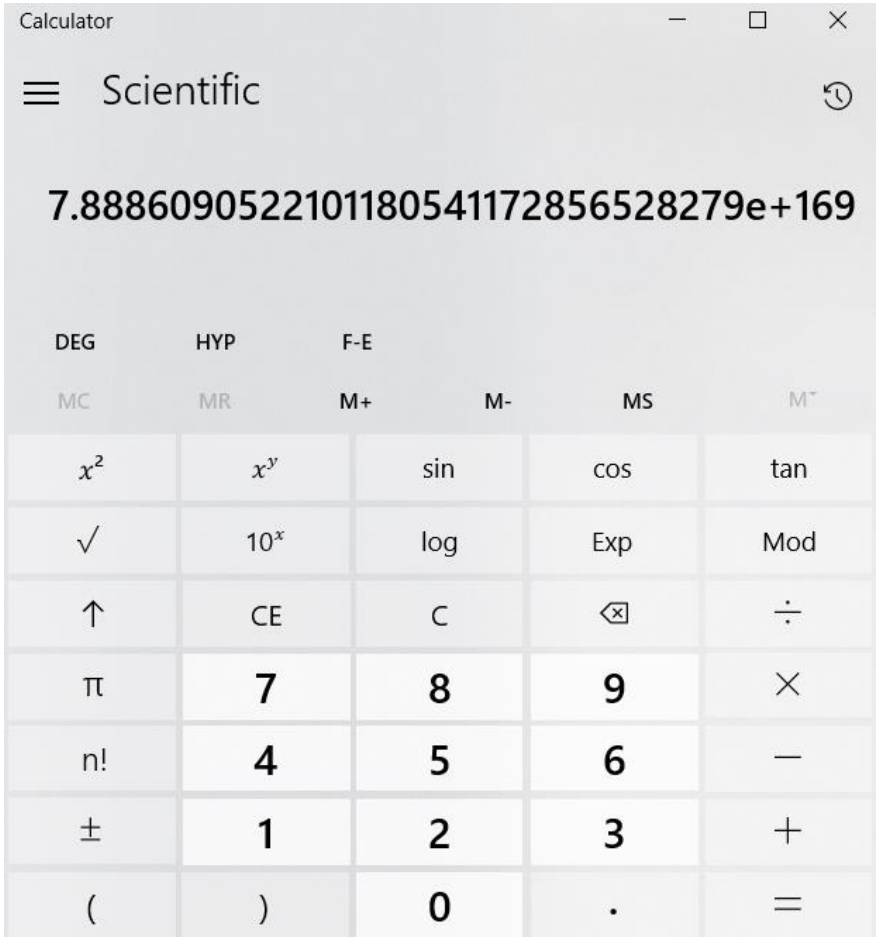
$= 32\ \%\ 3$

$= 2$

➔ Kết quả: $(2^5)\ \%\ 3\ =\ 2$

Bài toán minh họa 1

Ví dụ 2: Cho a = 50, b = 100, m = 13. Tính $(50^{100})\ \%\ 13$ ➔ Tràn số.



Phương án giải quyết

3. Phép nhân: $(a * b) \bmod m = ((a \bmod m) * (b \bmod m)) \bmod m$

Ta có: $(50^{100})\ \%\ 13$

$= (\underbrace{50 * 50 * 50 * \ldots * 50}_{\text{99 phép nhân}})\ \%\ 13 = 3$

```cpp
long long modularExponentiation(int a, int b, int m) {
    long long result = 1;
    for (int i = 1; i <= b; i++) {
        result *= a;
        result %= m;
    }
    return result;
}
```

Độ phức tạp: O(b)

Phương án giải quyết

```python
def modularExponentiation(a, b, m):
    result = 1
    for i in range(1, b + 1):
        result *= a
        result %= m
    return result
```

```java
private static long modularExponentiation(int a, int b, int m) {
    long result = 1;
    for (int i = 1; i <= b; i++) {
        result *= a;
        result %= m;
    }
    return result;
}
```

Độ phức tạp: O(b)

Phương án cải tiến

Bước 1: Áp dụng đồng dư để giảm số a nhỏ lại.

$$a^b \bmod m = ((a \bmod m)^b) \bmod m$$

Ví dụ: $50^{100} \% 13$

mà $50 \% 13 = 11$

➔ $50^{100} \% 13 = 11^{100} \% 13$

Bước 2: Áp dụng thuật toán **Exponentiation by squaring** (Thuật toán bình phương và nhân) để giảm số b.

Đây là thuật toán tính nhanh lũy thừa của một số với số mũ là số tự nhiên. Trong trường hợp cơ số là số nguyên có thể được rút gọn theo một modulo nào đó.

Phương án cải tiến

Công thức:

$$a^b = \begin{cases} a * (a^2)^{\frac{b-1}{2}}, & nếu\ b\ là\ số\ lẻ \\ (a^2)^{\frac{b}{2}}, & nếu\ b\ là\ số\ chẵn \end{cases}$$

Dừng khi b = 0.

b lẻ

Ví dụ: $11^{99}\ \%\ 13$

$= 11 * (11^2)^{\frac{99-1}{2}}\ \%\ 13$

$= 11 * (11^2)^{49}\ \%13$

b chẵn

Ví dụ: $11^{100}\ \%\ 13$

$= (11^2)^{\frac{100}{2}}\ \%\ 13$

$= (11^2)^{50}\ \%\ 13$

Bước 1: Áp dụng tính chất đồng dư

Tìm đồng dư của số a (mod m) để giảm bớt lại giá trị cần tính toán:

$$a^b \bmod m = ((a \bmod m)^b) \bmod m$$

Ta có: $50^{100} \% 13$

mà $50 \% 13 = 11$

➔ $50^{100} \% 13$

$= 11^{100} \% 13$

Lúc này a đã giảm xuống nhỏ rồi, nhưng b vẫn còn rất lớn, nếu tính giá trị 11^{100} vẫn sẽ bị tràn số ➔ Áp dụng bước 2 để giảm nhỏ số b lại.

Bước 2: Áp dụng thuật toán bình phương và nhân (1)

Xét a = 11, b = 100, m = 13 ➔ b chẵn.

$a^b = (a^2)^{\frac{b}{2}}, \quad$ *nếu b là số chẵn*

$$11^{100} \,\%\, 13$$

$$= (11^2)^{50} \,\%\, 13$$

$$= 121^{50} \,\%\, 13$$

Áp dụng lại bước 1: Tìm đồng dư của số a (mod m) để giảm bớt lại giá trị cần tính toán:

$$121 \,\%\, 13 = 4$$

➔ $121^{50} \,\%\, 13$

$$= \mathbf{4^{50} \,\%\, 13}$$

Bước 2: Áp dụng thuật toán bình phương và nhân (2)

Xét a = 4, b = 50, m = 13 ➔ b chẵn.

$a^b = (a^2)^{\frac{b}{2}}, \quad$ *nếu b là số chẵn*

$$4^{50} \,\%\, 13$$

$$= (4^2)^{25} \,\%\, 13$$

$$= 16^{25} \,\%\, 13$$

Áp dụng lại bước 1: Tìm đồng dư của số a (mod m) để giảm bớt lại giá trị cần tính toán:

$$16 \,\%\, 13 = 3$$

➔ $16^{25} \,\%\, 13$

$$= \mathbf{3^{25} \,\%\, 13}$$

Bước 2: Áp dụng thuật toán bình phương và nhân (3)

Xét a = 3, b = 25, m = 13 ➔ b lẻ.

$$a^b = a * (a^2)^{\frac{b-1}{2}}, \quad \text{nếu } b \text{ là số lẻ}$$

$$3^{25} \,\%\, 13$$

$$= \; 3.(3^2)^{12} \,\%\, 13$$

$$= \; 3.(9^{12}) \,\%\, 13$$

Áp dụng lại bước 1: Tìm đồng dư của số a (mod m) để giảm bớt lại giá trị cần tính toán:

$$9 \,\%\, 13 \; = \; 9$$

➔ $3.(9^{12}) \,\%\, 13$ sẽ không thay đổi.

$$= \; \mathbf{3.(9^{12}) \,\%\, 13}$$

Bước 2: Áp dụng thuật toán bình phương và nhân (4)

$a^b = (a^2)^{\frac{b}{2}}, \quad$ *nếu b là số chẵn*

Xét a = 9, b = 12, m = 13 ➔ b chẵn.

$$3.(9^{12})\ \%\ 13$$

$$= \ 3.(9^2)^6\ \%\ 13$$

$$= \ 3.(81^6)\ \%\ 13$$

Áp dụng lại bước 1: Tìm đồng dư của số a (mod m) để giảm bớt lại giá trị cần tính toán:

$$81\ \%\ 13\ = \ 3$$

➔ $3.(81^6)\ \%\ 13$

$$= \ \mathbf{3.(3^6)\ \%\ 13}$$

Bước 2: Áp dụng thuật toán bình phương và nhân (5)

$a^b = (a^2)^{\frac{b}{2}}, \quad$ *nếu b là số chẵn*

Xét a = 3, b = 6, m = 13 ➔ b chẵn.

$$3.(3^6)\ \%\ 13$$

$$= 3.(3^2)^3\ \%\ 13$$

$$= 3.(9^3)\ \%\ 13$$

Áp dụng lại bước 1: Tìm đồng dư của số a (mod m) để giảm bớt lại giá trị cần tính toán:

$$9\ \%\ 13 = 9$$

➔ $3.(9^3)\ \%\ 13$ sẽ không thay đổi.

$$= \mathbf{3.(9^3)\ \%\ 13}$$

Bước 2: Áp dụng thuật toán bình phương và nhân (6)

Xét a = 9, b = 3, m = 13 ➔ b lẻ.

$$a^b = a * (a^2)^{\frac{b-1}{2}}, \quad \text{nếu } b \text{ là số lẻ}$$

$$3.(9^3)\ \%\ 13$$

$$= 3.(9.(9^2)^1)\ \%\ 13$$

$$= 27.(81^1)\ \%\ 13$$

Áp dụng: Tính chất phân phối của phép nhân và đồng dư của số a (mod m) để giảm bớt lại giá trị cần tính toán:

$$27\ \%\ 13 = 1$$

➔ $27.(81^1)\ \%\ 13$

$$= 81^1\ \%\ 13$$

$$81\ \%\ 13 = 3$$

➔ $81^1\ \%\ 13$

$$= 3^1\ \%\ 13$$

Bước 2: Áp dụng thuật toán bình phương và nhân (7)

Xét a = 3, b = 1, m = 13 ➔ b lẻ.

$$a^b = a * (a^2)^{\frac{b-1}{2}}, \quad \text{nếu } b \text{ là số lẻ}$$

$$3^1 \,\%\, 13$$

$$= 3.3^0 \,\%13$$

$$= 3 \,\%\, 13$$

$$= 3$$

➔ **Dừng thuật toán khi b = 0.**

$$(50^{100}) \,\%\, 13$$

$$= 3 \,\%\, 13$$

$$= 3$$

Source Code Modular Exponentiation

```cpp
#include <iostream>
using namespace std;
int modularExponentiation(int a, int b, int m)
{
    int result = 1;
    a %= m;
    while (b > 0)
    {
        // If b is odd, multiply a with result
        if (b % 2 == 1)
            result = (result * a) % m;
        // b must be even now
        b /= 2;
        a = (a * a) % m;
    }
    return result;
}
```

Source Code Modular Exponentiation

```
18. int main()
19. {
20.     int a = 50, b = 100, m = 13;
21.     int result = modularExponentiation(a, b, m);
22.     cout << "Modular Exponentiation: " << a << " ^ " << b
                << " (mod " << m << ") = " << result << endl;
23.     return 0;
24. }
```

Source Code Modular Exponentiation

```
1.  def modularExponentiation(a, b, m):
2.      result = 1
3.      a %= m
4.      while b > 0:
5.          if b % 2 == 1:
6.              result = (result * a) % m
7.          b //= 2
8.          a = (a * a) % m
9.      return result

10. if __name__ == "__main__":
11.     a = 50
12.     b = 100
13.     m = 13
14.     result = modularExponentiation(a, b, m)
15.     print('Modular Exponentiation: ', a, ' ^ ', b, ' (mod ', m, ') = ',
                                                        result, sep = '')
```

Source Code Modular Exponentiation

```java
public class Main {
    private static int modularExponentiation(int a, int b, int m) {
        int result = 1;
        a %= m;
        while (b > 0) {
            // If b is odd, multiply a with result
            if (b % 2 == 1)
                result = (result * a) % m;
            // b must be even now
            b /= 2;
            a = (a * a) % m;
        }
        return result;
    }
```

Source Code Modular Exponentiation

```
15.     public static void main(String[] args) throws Exception {
16.         int a = 50, b = 100, m = 13;
17.         int result = modularExponentiation(a, b, m);
18.         System.out.printf("Modular Exponentiation: %d ^ %d
                                (mod %d) = %d\n", a, b, m, result);
19.     }
20. }
```

BONUS: MÃ NGUỒN MODULAR EXPONENTIATION VIẾT THEO XỬ LÝ BIT

Source Code Modular Exponentiation BIT


```cpp
#include <iostream>
using namespace std;
int modularExponentiation(int a, int b, int m)
{
    int result = 1;
    a = a % m;
    while (b > 0)
    {
        // If b is odd, multiply a with result
        if (b & 1)
            result = (result * a) % m;
        // b must be even now
        b = b >> 1;
        a = (a * a) % m;
    }
    return result;
}
```

Source Code Modular Exponentiation BIT

```
18. int main()
19. {
20.     int a = 50, b = 100, m = 13;
21.     int result = modularExponentiation(a, b, m);
22.     cout << "Modular Exponentiation: " << a << " ^ " << b
                << " (mod " << m << ") = " << result << endl;
23.     return 0;
24. }
```

Source Code Modular Exponentiation BIT

```
1.  def modularExponentiation(a, b, m):
2.      result = 1
3.      a %= m
4.      while b > 0:
5.          if b & 1:
6.              result = (result * a) % m
7.          b >>= 1
8.          a = (a * a) % m
9.      return result
10.   if __name__ == "__main__":
11.     a = 50
12.     b = 100
13.     m = 13
14.     result = modularExponentiation(a, b, m)
15.     print('Modular Exponentiation: ', a, ' ^ ', b,
                          ' (mod ', m, ') = ', result, sep = '')
```

Source Code Modular Exponentiation BIT

```java
public class Main {
    private static int modularExponentiation(int a, int b, int m) {
        int result = 1;
        a %= m;
        while (b > 0) {
            // If b is odd, multiply a with result
            if ((b & 1) == 1)
                result = (result * a) % m;
            // b must be even now
            b >>= 1;
            a = (a * a) % m;
        }
        return result;
    }
```

Source Code Modular Exponentiation BIT

```
15.     public static void main(String[] args) throws Exception {
16.         int a = 50, b = 100, m = 13;
17.         int result = modularExponentiation(a, b, m);
18.         System.out.printf("Modular Exponentiation: %d ^ %d
                                (mod %d) = %d\n", a, b, m, result);
19.     }
20. }
```

Bài toán minh họa 2

Modular multiplicative inverse: Cho bạn 2 số b và m hãy tìm nghịch đảo của b theo modulo m.

Gọi x là nghịch đảo của b theo modulo m:

$$bx \bmod m = 1$$ **Hoặc** $$bx \equiv 1 \bmod m$$

Giống như nghịch đảo của số nguyên: khi số nguyên đó nhân với số nghịch đảo của nó sẽ ra 1.

Ví dụ về nghịch đảo số nguyên: $\frac{1}{5}$ hay 5^{-1} là nghịch đảo của 5.

➔ Vì $5 * \frac{1}{5} = 1$.

Bài toán minh họa 2

Các ví dụ về ngịch đảo của b theo modulo m.

Ví dụ 1: 9 là nghịch đảo của 5 theo modulo 11.

(Vì 9 * 5 = 45 mà 45 % 11 = 1)

Ví dụ 2: 18 là nghịch đảo của 7 theo modulo 25.

(Vì 18 * 7 = 126 mà 126 % 25 = 1)

Ví dụ 3: Tìm nghịch đảo của 4 theo modulo 14.

Không có kết quả. Vì không tồn tại số nguyên x sao cho $(x * 4) \% 14 = 1$.

**** Lưu ý: không phải mọi số đều có số nghịch đảo theo modulo.*

Một số tính chất phép modulo

Big-O Orange

1. Phép cộng: $(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$

Ví dụ: Tính $(10 + 5) \% 3$

$= ((10 \% 3) + (5 \% 3)) \% 3$

$= (1 + 2) \% 3$

$= 0$

2. Phép trừ: $(a - b) \bmod m = ((a \bmod m) - (b \bmod m)) \bmod m$

3. Phép nhân: $(a * b) \bmod m = ((a \bmod m) * (b \bmod m)) \bmod m$

4. Phép chia: $(a / b) \bmod m$ chỉ có nghĩa khi a chia hết b, phép chia **không** có tính chất phân phối như phép cộng, trừ và nhân. Để giải bài toán này dùng phương pháp **nghịch đảo modulo (phần sau)**.

Bài toán minh họa 2

Tính: $(75 / 5) \% 11$

$= (75 * 9) \% 11$ (do 9 là nghịch đảo của 5 theo modulo 11)

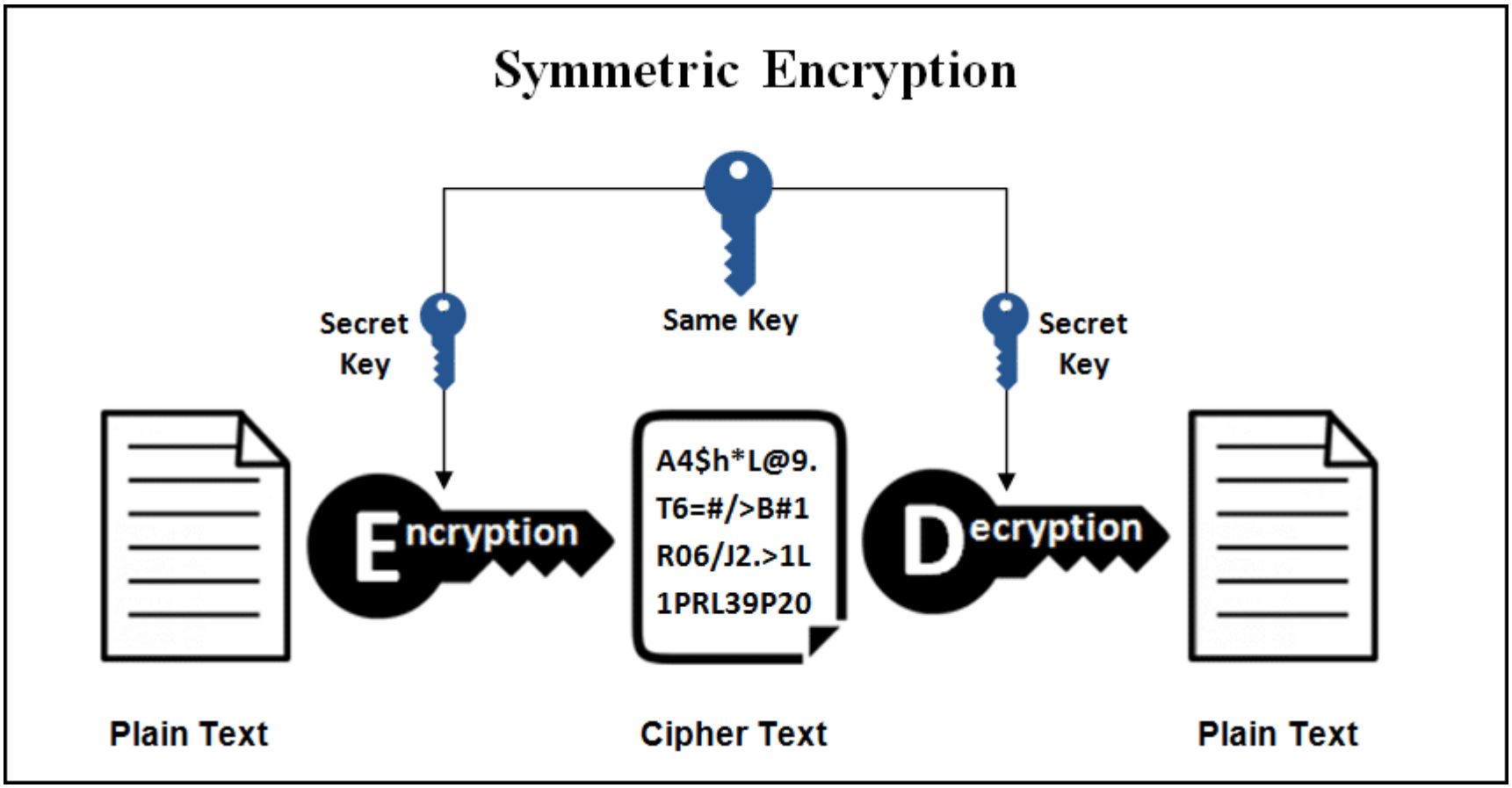
$= ((75 \% 11) * (9 \% 11)) \% 11$

$= (9 * 9) \% 11$

$= 81 \% 11$

$= 4$

Modular multiplicative inverse



Ireland, Kenneth; Rosen, Michael (1990), A Classical Introduction to Modern Number Theory (2nd ed.), Springer-Verlag

Ý tưởng tìm Modular multiplicative inverse

Ta có: (75 / 5) % 11

Tìm nghịch đảo của 5 theo modulo 11.

step	b	x	m	r	Conclusion
1	5	1	11	5	r = (5 * 1) % 11 = 5 **khác** 1 tăng biến x
2	5	2	11	10	r = (5 * 2) % 11 = 10 **khác** 1 tăng biến x
3	5	3	11	4	r = (5 * 3) % 11 = 4 **khác** 1 tăng biến x
4	5	4	11	9	r = (5 * 4) % 11 = 9 **khác** 1 tăng biến x
5	5	5	11	3	r = (5 * 5) % 11 = 3 **khác** 1 tăng biến x
6	5	6	11	8	r = (5 * 6) % 11 = 8 **khác** 1 tăng biến x
7	5	7	11	2	r = (5 * 7) % 11 = 2 **khác** 1 tăng biến x
8	5	8	11	7	r = (5 * 8) % 11 = 7 **khác** 1 tăng biến x
9	**5**	**9**	**11**	**1**	**r = (5 * 9)%11 = 1 ➔ Dừng thuật toán**

Độ phức tạp: O(m)

Source Code Modular multiplicative inverse

```cpp
#include<iostream>
using namespace std;
int modInverse(int b, int m)
{
    b = b % m;
    for (int x = 1; x < m; x++)
    {
        int r = (b * x) % m;
        if (r == 1)
            return x;
    }
    return -1;
}
int main()
{
    int b = 5, m = 11;
    cout << modInverse(b, m);
    return 0;
}
```

Source Code Modular multiplicative inverse

```
1.  def modInverse(b, m):
2.      b = b % m
3.      for x in range(1, m):
4.          r = (b * x) % m
5.          if r == 1:
6.              return x
7.      return -1

8.  if __name__ == "__main__":
9.      b = 5
10.     m = 11
11.     print(modInverse(b, m))
```

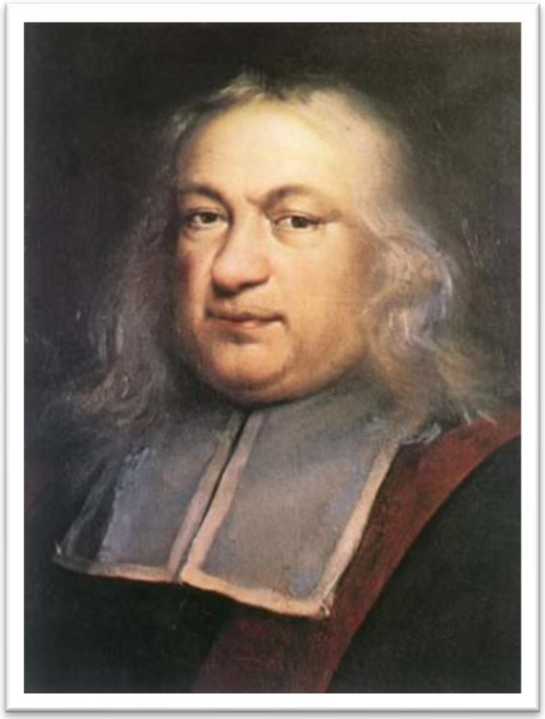
Source Code Modular multiplicative inverse

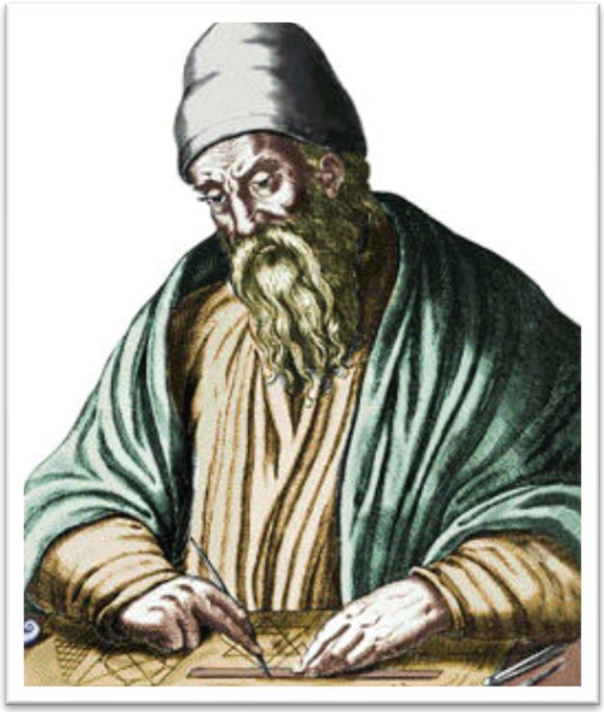
```java
public class Main {
    private static int modInverse(int b, int m) {
        b = b % m;
        for (int x = 1; x < m; x++) {
            int r = (b * x) % m;
            if (r == 1)
                return x;
        }
        return -1;
    }

    public static void main(String[] args) throws Exception {
        int b = 5, m = 11;
        System.out.println(modInverse(b, m));
    }
}
```

Tìm nhanh Modular multiplicative inverse của một số

Để tìm **nhanh** nghịch đảo modulo của một số ta dùng 2 phương pháp sau:

- Fermat's little theorem (Định lý Fermat nhỏ)
- Extended Euclidean algorithm (Giải thuật Euclid mở rộng)





FERMAT'S LITTLE THEOREM

Fermat's little theorem

Fermat's little theorem (Định lý Fermat nhỏ): Nếu m là một **số nguyên tố** và b **không** chia hết cho m thì $b^{m-1} - 1$ chia hết cho m (hay nói cách khác b^{m-1} modulo m dư 1).

$$b^{m-1} \equiv 1 \ mod \ m$$

Ví dụ: b = 5, m = 11 (m là số nguyên tố và b không chia hết cho m)

➔ $(5^{11-1} - 1) \ \% \ 11$

= $(9.765.625 - 1) \ \% \ 11$

= $9.765.624 \ \% \ 11$

= 0

**** Lưu ý: cần sử dụng lại bài toán Modular Exponentiation để tính kết quả.*

Fermat's little theorem

Nếu ta có: $b^{m-1} \equiv 1 \ (mod \ m)$

➔ $b * b^{m-2} \equiv 1 \ (mod \ m)$

Suy ra $x = \ b^{m-2} \ mod \ m$ (x là nghịch đảo của b theo modulo m)

Ví dụ: b = 5, m = 11 (m là số nguyên tố và b không chia hết cho m)

➔ $5 * 5^{11-2} \equiv 1 \ (mod \ 11)$

Mà $x = b^{m-2} \bmod 11$

➔ $x = 5^{11-2} \ \% \ 11$

$= 5^9 \ \% \ 11$

$= 9$

Vậy 9 là nghịch đảo của 5 theo modulo 11.

Source Code Fermat's little theorem

```cpp
#include <iostream>
using namespace std;
int modularExponentiation(int a, int b, int m)
{
    int res = 1;
    a = a % m;
    while (b > 0)
    {
        if (b % 2 == 1)
            res = (res * a) % m;
        b = b / 2;
        a = (a * a) % m;
    }
    return res;
}
```

Source Code Fermat's little theorem

```
16. int modInverse(int b, int m)
17. {
18.     int res = modularExponentiation(b, m - 2, m);
19.     if ((res*b) % m == 1)
20.         return res;
21.     return -1;
22. }
```


```
23. int main()
24. {
25.     int b = 5;
26.     int m = 11;
27.     cout << modInverse(b, m) << endl;
28.     return 0;
29. }
```

Source Code Fermat's little theorem

```
1. def modularExponentiation(a, b, m):
2.     result = 1
3.     a %= m
4.     while b > 0:
5.         if b % 2 == 1:
6.             result = (result * a) % m
7.         b //= 2
8.         a = (a * a) % m
9.     return result
```

Source Code Fermat's little theorem

```
10. def modInverse(b, m):
11.     res = modularExponentiation(b, m - 2, m)
12.     if (res * b) % m == 1:
13.         return res
14.     return -1
```


```
15. if __name__ == "__main__":
16.     b = 5
17.     m = 11
18.     print(modInverse(b, m))
```

Source Code Fermat's little theorem

```java
public class Main {
    private static int modularExponentiation(int a, int b, int m) {
        int res = 1;
        a = a % m;
        while (b > 0) {
            if (b % 2 == 1)
                res = (res*a) % m;
            b = b / 2;
            a = (a * a) % m;
        }
        return res;
    }
```

Source Code Fermat's little theorem

```
13.     private static int modInverse(int b, int m) {
14.         int res = modularExponentiation(b, m - 2, m);
15.         if ((res*b) % m == 1)
16.             return res;
17.         return -1;
18.     }
```


```
13.     public static void main(String[] args) throws Exception {
14.         int b = 5, m = 11;
15.         System.out.println(modInverse(b, m));
16.     }
17. }
```

EXTENDED EUCLIDEAN ALGORITHM

Bonus: Euclidean algorithm

Thuật toán Euclid: là thuật toán dùng để tính ước số chung lớn nhất (UCLN), có tên tiếng Anh Greatest Common Divisor (GCD).

Ký hiệu: UCLN(a, b) **hoặc** GCD(a, b)

*Với a, b là hai số nguyên, ta thực hiện chia a cho b được thương q, số dư r (r ≥ 0) tức là r = a – q*b*, khi đó ta có:

$$GCD(a,b) = \begin{cases} b \text{ nếu } r = 0 \\ GCD(b,r) \text{ nếu } r \neq 0 \end{cases}$$

Bonus: Euclidean algorithm

Để tìm ước số chung lớn nhất của a và b, thuật toán Euclid sẽ dùng tính chất modulo, lấy 2 số modulo với nhau (a mod b). Sau đó b gán cho a, phần dư r của phép modulo đó sẽ được gán lại cho b. Quá trình này lặp đi lặp lại cho đến khi một trong 2 số bằng 0 thì dừng và số kia chính là ước số chung lớn nhất của 2 số ban đầu a và b.

Ví dụ: Tính GCD(174, 44)

step	a	b	r	Conclusion
1	174	44	42	➔ r = a % b = 174 % 44 = 42 (gán a = b, b = r)
2	44	42	2	➔ r = a % b = 44 % 42 = 2 (gán a = b, b = r)
3	42	2	0	➔ r = a % b = 44 % 2 = 0 (gán a = b, b = r)
4	2	0	0	**Dừng thuật toán b = 0**

➔ **Kết quả GCD(a, b) = 2**

Độ phức tạp: **O(log(min(a, b)))**

Source Code Euclidean algorithm

```cpp
int gcd(int a, int b)
{
    int remainder;
    while (b != 0)
    {
        remainder = a % b;
        a = b;
        b = remainder;
    }
    return a;
}
```

Source Code Euclidean algorithm

```
1. def gcd(a, b):
2.     while b != 0:
3.         remainder = a % b
4.         a = b
5.         b = remainder
6.     return a
```

Source Code Euclidean algorithm

```
1. private static int gcd(int a, int b) {
2.     int remainder;
3.     while (b != 0) {
4.         remainder = a % b;
5.         a = b;
6.         b = remainder;
7.     }
8.     return a;
9. }
```

Extended Euclidean algorithm

Phương trình Diophantine tuyến tính có dạng:

$$ax + by = c$$

Trong đó:

a, b, c là ba hệ số nguyên.

x, y là các ẩn nhận giá trị nguyên.

Phương trình trên có nghiệm nguyên, phụ thuộc vào mối quan hệ **GCD(a, b) và c:**

- Nếu c **không** chia hết cho GCD(a, b) ➔ Phương trình vô nghiệm.
- Nếu c chia hết cho GCD(a, b) ➔ Phương trình vô số nghiệm.

Thuật toán Euclid mở rộng được sử dụng giải phương trình Diophantine.

Extended Euclidean algorithm

Ví dụ: $174x + 44y = 6$. Tìm **một** nghiệm nguyên của phương trình.

$$r = a - q * b$$

step	a	b	q	r	Conclusion
1	174	44	3	42	➔ 42 = 174 – 3 * 44
2	44	42	1	2	➔ 2 = 44 – 1 * 42 ⇔ 2 = 44 – 1 * (174 – 3 * 44) ⇔ 2 = 174 * (-1) + 44 * (4)
3	42	2	21	0	➔ 0 = 42 – 21 * 2
4	2	0			**Dừng thuật toán b=0**

Extended Euclidean algorithm

step	a	b	q	r	Conclusion
1	174	44	3	42	➔ 42 = 174 – 3 * 44
2	44	42	1	2	➔ 2 = 44 – 1 * 42 ⇔ 2 = 44 – 1 * (174 – 3 * 44) ⇔ 2 = 174 * (-1) + 44 * (4)
3	42	2	21	0	➔ 0 = 42 – 21 * 2
4	2	0			**Dừng thuật toán b=0**

step 2: 2 = 174 * (-1) + 44 * (4)

⇔ 3 * 2 = 3 * (174 * (-1) + 44 * (4)) (nhân 3 cho cả 2 vế đẳng thức)

⇔ 174 * (-3) + 44 * (12) = 6

$$174x + 44y = 6$$

Kết quả:

- GCD(174, 44) = 2
- Nghiệm phương trình: x = -3, y = 12.

Extended Euclidean algorithm

Ví dụ 2: $19x + 141y = 1$. Tìm **một** nghiệm nguyên của phương trình.

Thực hiện phép mod 141 cả 2 vế:

$(19x + 141y)\ mod\ 141 = 1\ mod\ 141$

$\Leftrightarrow\ 19x\ mod\ 141 + 0 = 1$

$\Leftrightarrow\ 19x\ mod\ 141 = 1$

$\Leftrightarrow\ 19x \equiv 1\ (mod\ 141)$

➔ **Giải phương trình:** $19x + 141y = 1$ ➔ Tìm được x (nghịch đảo của 19 modulo 141).

Extended Euclidean algorithm

Tìm nghịch đảo modulo của x: $19x \equiv 1 \ (mod \ 141)$

➔ **Chuyển thành phương trình:** $19x + \ 141y = 1$

- b = 19, m = 141
- quotient $(q) \ = \ b/m$
- remainder $(r) \ = \ b \ \% \ m \ = \ b - \ q \ * \ m$

step	b	m	q	r	Conclusion
1	19	141	0	19	19 = 19 – 0 * 141 19 = 19 * (1)
2	141	19	7	8	8 = 141 – 7 * 19 8 = 19 * (-7) + 141 * (1)
3	19	8	2	3	3 = 19 – 2 * 8 3 = 19 – 2 * (19 * (-7) + 141 * (1)) 3 = 19 * (15) + 141 * (-2)

Extended Euclidean algorithm

Tìm nghịch đảo modulo của x: $19x \equiv 1 \ (mod\ 141)$

➔ **Chuyển thành phương trình:** $19x + 141y = 1$

step	b	m	q	r	Conclusion
1	19	141	0	19	19 = 19 * (1)
2	141	19	7	8	8 = 19 * (-7) + 141 * (1)
3	19	8	2	3	3 = 19 * (15) + 141 * (-2)
4	8	3	2	2	2 = 8 – 2 * 3 2 = (19 * (-7) + 141 * (1)) - 2 * (19 * (15) + 141 * (-2)) 2 = 19 * (-37) + 141 * (5)
5	3	2	1	1	1 = 3 – 1 * 2 1 = (19 * (15) + 141 * (-2)) – 1 * (19 * (-37) + 141 * (5)) 1 = 19 * (52) + 141 * (-7)
6	2	1	2	0	0 = 2 – 2 * 1
7	1	0			Dừng thuật toán m = 0

Extended Euclidean algorithm

Tìm nghịch đảo modulo của x: $19x \equiv 1 \ (mod \ 141)$

➔ **Chuyển thành phương trình:** $19x + \ 141y = 1$

step	b	m	quotient	remainder	Conclusion
1	19	141	0	19	19 = 19 * (1)
2	141	19	7	8	8 = 19 * (-7) + 141 * (1)
3	19	8	2	3	3 = 19 * (15) + 141 * (-2)
4	8	3	2	2	2 = 19 * (-37) + 141 * (5)
5	3	2	1	1	**1 = 19 * (52) + 141 * (-7)**
6	2	1	2	0	0 = 2 – 2 * 1
7	1	0			Dừng thuật toán m = 0

Kết luận: 19x + 141y = 1 (x = 52, y = -7)

Do đó: $19 = 52^{-1} \ (mod \ 141)$, 52 là nghịch đảo của 19 theo modulo 141.

Extended Euclidean algorithm

step	b	m	quotient	remainder	Conclusion
1	19	141	0	19	19 = 19 * (1)
2	141	19	7	8	8 = 19 * (-7) + 141 * (1)
3	19	8	2	3	3 = 19 * (15) + 141 * (-2)
4	8	3	2	2	2 = 19 * (-37) + 141 * (5)
5	3	2	1	1	**1 = 19 * (52) + 141 * (-7)**
6	2	1	2	0	0 = 2 – 2 * 1
7	1	0			Dừng thuật toán m = 0

Các công thức cần nhớ:

- $q = b / m$
- $r = b \,\%\, m$ *hoặc* $r = b - q * m$
- $x_i = x_{i-2} - q_i * x_{i-1}$

$(Với\ i = 3\ thì\ x_3 = x_1 - q_3 * x_2 \Rightarrow 15 = 1 - 2 * (-7))$

- $y_i = y_{i-2} - q_i * y_{i-1}$

$(Với\ i = 3\ thì\ y_3 = y_1 - q_3 * y_2 \Rightarrow -2 = 0 - 2 * (1))$

Extended Euclidean algorithm

Các giá trị khởi đầu:

- $x = 1, y = 0$ *(do step 1: 19 = b * 1)*
- $x_0 = 0, y_0 = 1$

  $(Với\ i = 2\ thì\ x_2 = x_0 - q_2 * x_1 \Rightarrow -7 = x_0 - 7 * 1)$

  $(Với\ i = 2\ thì\ y_2 = y_0 - q_2 * y_1 \Rightarrow 1 = y_0 - 7 * 0)$
- $Tương\ tự:\ x_{-1} = 1, y_{-1} = 0$

Extended Euclidean algorithm

```cpp
#include <iostream>
#include <vector>
using namespace std;
vector<int> extendedEuclid(int b, int m)
{
    vector<int> result;
    int x1 = 0, y1 = 1;
    int x2 = 1, y2 = 0;
    int q, r;
    int x = 1, y = 0;
    while (m != 0)
    {
        q = b / m;
        r = b % m;
        x = x2 - q * x1;
        y = y2 - q * y1;
        x2 = x1, y2 = y1;
        x1 = x, y1 = y;
        b = m, m = r;
    }
```

Extended Euclidean algorithm

```
21.      result.push_back(b);
22.      result.push_back(x2);
23.      result.push_back(y2);
24.      return result;
25.  }
```


```
26.  void modInverse(int b, int m)
27.  {
28.      vector<int> result = extendedEuclid(b, m);
29.      int gcd = result[0];
30.      int x = result[1];
31.      int y = result[2];
32.      if (gcd != 1)
33.          cout << "Inverse doesn't exist" << endl;
34.      else
35.          cout << "Modular multiplicative inverse is: "
                 << (x + m) % m << endl;
36.  }
```

Extended Euclidean algorithm

```
34. int main()
35. {
36.     int b = 19, m = 141;
37.     modInverse(b, m);
38.     return 0;
39. }
```

Extended Euclidean algorithm

```
1.  def extendedEuclid(b, m):
2.      result = []
3.      x1 = 0
4.      y1 = 1
5.      x2 = 1
6.      y2 = 0
7.      x = 1
8.      y = 0
9.      while m != 0:
10.         q = b // m
11.         r = b % m
12.         x = x2 - q * x1
13.         y = y2 - q * y1
14.         x2 = x1
15.         y2 = y1
16.         x1 = x
17.         y1 = y
18.         b = m
19.         m = r
```

Extended Euclidean algorithm

```
20.     result.append(b)
21.     result.append(x2)
22.     result.append(y2)
23.     return result
```


```
20. def modInverse(b, m):
21.     result = extendedEuclid(b, m)
22.     gcd = result[0]
23.     x = result[1]
24.     y = result[2]
25.     if gcd != 1:
26.         print("Inverse doesn't exist")
27.     else:
28.         print("Modular multiplicative inverse is:", (x + m) % m)
```

Extended Euclidean algorithm

```
33.     if __name__ == "__main__":
34.     b = 19
35.     m = 141
36.     modInverse(b, m)
```

Extended Euclidean algorithm

```java
import java.lang.reflect.Array;
import java.util.ArrayList;
public class Main {
    private static ArrayList<Integer> extendedEuclid(int b, int m) {
        ArrayList<Integer> result = new ArrayList<>();
        int x1 = 0, y1 = 1;
        int x2 = 1, y2 = 0;
        int q, r;
        int x = 1, y = 0;
        while (m != 0) {
            q = b / m;
            r = b % m;
            x = x2 - q * x1;
            y = y2 - q * y1;
            x2 = x1;
            y2 = y1;
            x1 = x;
            y1 = y;
            b = m;
            m = r;
        }
```

Extended Euclidean algorithm

```
22.             result.add(b);
23.             result.add(x2);
24.             result.add(y2);
25.             return result;
26.         }
```


```
22.         private static void modInverse(int b, int m) {
23.             ArrayList<Integer> result = extendedEuclid(b, m);
24.             int gcd = result.get(0);
25.             int x = result.get(1);
26.             int y = result.get(2);
27.             if (gcd != 1)
28.                 System.out.println("Inverse doesn't exist");
29.             else
30.                 System.out.printf("Modular multiplicative inverse is: %d\n",
                                                                    (x + m) % m);
36.         }
```

Extended Euclidean algorithm

```
37.     public static void main(String[] args) throws Exception {
38.         int b = 19, m = 141;
39.         modInverse(b, m);
40.     }
41. }
```

Tổng kết

Để tìm **nhanh** nghịch đảo modulo của một số ta dùng 2 phương pháp sau:

- Fermat's little theorem (Định lý Fermat nhỏ)

  ➔ Áp dụng khi m là một số nguyên tố và b không chia hết cho m.

- Extended Euclidean algorithm (Giải thuật Euclid mở rộng)

  ➔ Áp dụng khi GCD(b, m) = 1. Giải phương trình $bx + my = 1$.

Độ phức tạp: O(log(m))

Hỏi đáp