КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп’ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Лабораторна робота №3

з предмету «Математичні основи захисту інформації»

Варіант №13

Підготував:

Грищенко Юрій, ІПС-42

Київ – 2022

Завдання 1.

Знайти d - найбільший спільний дільник чисел a,b,
розв'язати рівняння au+bv=d,
розв'язати рівняння as+bt=c,

```
1. a = 39
   b = 30
   c = 18
   Відповідь:
   d = 3
   u = -3
   v = 4
   s = -18
   t = 24

2. a = 5811988
   b = 4964974
   c = 6174
   Відповідь:
   d = 98
   u = 12210
   v = -14293
   s = 769230
   t = -900459

3. a = 160272279849904686
   b = 27342113381205372
   c = 20268
   Відповідь:
   d = 18
   u = -468658569941359
   v = 2747153317246111
   s = -527709549753970234
   t = 3093294635219120986
```

Теорія

<u>Розширений алгоритм Евкліда</u> — це розширення алгоритму Евкліда. Окрім знаходження найбільшого спільного дільника для цілих a і b, як це робить алгоритм Евкліда, він також знаходить цілі x і y (одне з яких зазвичай від'ємне), які задовольняють рівнянню Безу $ax+by=gcd(a,b)$

Для розв'язання другого рівняння, можна використати s = u * c/d; t = v * c/d

Код програми

```
public class Task1 {
    public static GCDExtendedResult gcdExtended(long a, long b) {
        if (a == 0)
            return new GCDExtendedResult(b, 0, 1);

        GCDExtendedResult result = gcdExtended(b % a, a);
```

```
        return new GCDExtendedResult(
                result.gcd,
                result.y - (b / a) * result.x, //new x
                result.x //new y
        );
    }

    static class GCDExtendedResult {
        public final long gcd;
        public final long x;
        public final long y;

        public GCDExtendedResult(long gcd, long x, long y) {
            this.gcd = gcd;
            this.x = x;
            this.y = y;
        }
    }



    public static void main(String[] args) {
        long a = 160272279849904686L;
        long b = 273421133812053 72L;
        long c = 20268;

        GCDExtendedResult result = gcdExtended(a, b);
        long d = result.gcd;
        long s = result.x * (c / d);
        long t = result.y * (c / d);

        System.out.println("d = " + d);
        System.out.println("u = " + result.x);
        System.out.println("v = " + result.y);
        System.out.println("s = " + s);
        System.out.println("t = " + t);
    }
}
```

Приклади

1)

a = 39

b = 30

c = 18

Результат:

```
d = 3
u = -3
v = 4
s = -18
t = 24
```

Дійсно, 39*-3 + 30*4=-117+120=3;   39*-18 + 30*24 = -702 * 720 = 18

2)

a = 5811988
b = 4964974
c = 6174
Результат:

```
d = 98
u = 12210
v = -14293
s = 769230
t = -900459
```

3)
a = 160272279849904686
b = 27342113381205372
c = 20268

Результат:

```
d = 18
u = -468658569941359
v = 2747153317246111
s = -527709549753970234
t = 3093294635219120986
```

Завдання 2.

Використовуючи дані числа a,b,c з варіантів 1-31 обчислити a^5, b^7, c^9 в полі лишків за модулем 23.

Теорія

Застосуємо алгоритм *Повторюване піднесення до квадрата:*

Вхід: Числа a, d, m.
Вихід: Число $y = a^d \pmod m$.
Метод:
1. Записати число d в двійковій системі числення $d = d_0 d_1 \cdots d_{r-1} d_r$.
2. $y := 1$; $s := a$;
3. for $i = r, r - 1, \ldots, 0$ do
3.1. $y := y \cdot y \pmod m$;

3.2. if $d_i = 1$ then $y := y \cdot a \pmod m$;
4. return(y).

Оскільки кожне обчислення на кроці 3) потребує не більше трьох множень за модулем m і цей крок виконується $r \le \log2\ m$ разів, то складність алгоритму виражається величиною $O(\log2\ m)$

Код програми

```
public class Task2 {
    public static void main(String[] args) {
        long a = 1602722798499046861;
        long b = 2734211338120537 2L;
        long c = 20268L;

        System.out.println("a^5 % 23 = " + binaryModPow(a, 5, 23));
        System.out.println("b^7 % 23 = " + binaryModPow(b, 7, 23));
        System.out.println("c^9 % 23 = " + binaryModPow(c, 9, 23));
    }

    static long binaryModPow(long a, long power, long mod) {
        long result = 1;
        a = a % mod;

        if (a == 0)
            return 0;
        while (power > 0) {
            if ((power & 1) != 0)
                result = (result * a) % mod;
            power = power >> 1;
            a = (a * a) % mod;
        }
        return result;
    }
}
```

Приклади

1)
a = 39
b = 30
c = 18
Результат:

```
a^5 % 23 = 6
b^7 % 23 = 5
c^9 % 23 = 12
```

2)
a = 5811988
b = 4964974
c = 6174

Результат:

```
a^5 % 23 = 13
b^7 % 23 = 14
c^9 % 23 = 20
```

3)
a = 160272279849904686
b = 27342113381205372
c = 20268

Результат:

```
a^5 % 23 = 3
b^7 % 23 = 1
c^9 % 23 = 11
```

Завдання 3.

Використовуючи дані числа a,b,c з варіантів 1-31 обчислити a^-1, b^-1, c^-1 в полі лишків за модулем 17.

Теорія

Можна використати розширений алгоритм Евкліда.

$ax+my=1$

$ax=1\,(mod\,m)$

$a=x^{-1}(mod\,m)$

Код програми

```
public class Task3 {
    static long modInverse(long x, long mod) {
        Task1.GCDExtendedResult result = Task1.gcdExtended(x, mod);

        if (result.gcd != 1) {
            throw new ArithmeticException("No inverse");
        } else {
            long inverse = result.x % mod;
            if (inverse > 0)
                return inverse;
            else
                return (inverse + mod) % mod;
        }
    }

    public static void main(String[] args) {
        long mod = 17;

        for (long x : new long[]{39, 30, 18, 5811988, 4964974, 6174, 160272279849904686L,
27342113381205372L, 20268}) {
            System.out.printf("%d ^ -1 = %d (mod %d)\n", x, modInverse(x, mod), mod);
```

```
        }
    }
}
```

Результати

```
39 ^ -1 = 7 (mod 17)
30 ^ -1 = 4 (mod 17)
18 ^ -1 = 1 (mod 17)
5811988 ^ -1 = 14 (mod 17)
4964974 ^ -1 = 7 (mod 17)
6174 ^ -1 = 6 (mod 17)
160272279849904686 ^ -1 = 9 (mod 17)
27342113381205372 ^ -1 = 2 (mod 17)
20268 ^ -1 = 13 (mod 17)
```

Завдання 4.

Використовуючи дані числа a,b,c з варіантів 1-31 обчислити a^1/2, b^1/2, c^1/2 в полі лишків за модулем 29.

Теорія

3.34 Algorithm Finding square roots modulo a prime p

INPUT: an odd prime p and an integer a, $1 \leq a \leq p-1$.
OUTPUT: the two square roots of a modulo p, provided a is a quadratic residue modulo p.

1. Compute the Legendre symbol $\left(\frac{a}{p}\right)$ using Algorithm 2.149. If $\left(\frac{a}{p}\right) = -1$ then return(a does not have a square root modulo p) and terminate.
2. Select integers b, $1 \leq b \leq p-1$, at random until one is found with $\left(\frac{b}{p}\right) = -1$. ($b$ is a quadratic non-residue modulo p.)
3. By repeated division by 2, write $p - 1 = 2^s t$, where t is odd.
4. Compute $a^{-1} \bmod p$ by the extended Euclidean algorithm (Algorithm 2.142).
5. Set $c \leftarrow b^t \bmod p$ and $r \leftarrow a^{(t+1)/2} \bmod p$ (Algorithm 2.143).
6. For i from 1 to $s-1$ do the following:
   6.1 Compute $d = (r^2 \cdot a^{-1})^{2^{s-i-1}} \bmod p$.
   6.2 If $d \equiv -1 \pmod p$ then set $r \leftarrow r \cdot c \bmod p$.
   6.3 Set $c \leftarrow c^2 \bmod p$.
7. Return(r, $-r$).

Цей алгоритм рандомізований, бо таким чином виконується пошук квадратичного нелишка b на кроці 2. Немає відомого алгоритму, який би виконував це у поліноміальному часі.

Код програми

```
public class Task4 {
    static long jacobi(long a, long n) {
        if (a == 0)
```

```
            return 0;
        if (a == 1)
            return 1;
        long e = 0;
        while (a % 2 == 0) {
            e++;
            a /= 2;
        }
        long s;
        if (e % 2 == 0 || n % 8 == 1 || n % 8 == 7)
            s = 1;
        else
            s = -1;

        if (n % 4 == 3 && a % 4 == 3)
            s = -s;

        n = n % a;
        if (a == 1)
            return s;
        else
            return s * jacobi(n, a);
    }

    static long squareRootMod(long a, long mod) {
        a = a % mod;
        if (a == 0)
            throw new ArithmeticException("0 does not have square root modulo " + mod);

        long legendre = jacobi(a, mod);
        if (legendre == -1)
            throw new ArithmeticException("a does not have square root modulo " + mod);

        long b;
        do {
            b = (long) (Math.random() * (mod - 2) + 1);
        } while (jacobi(b, mod) != -1);

        long s = 0;
        long t = mod - 1;
        while (t % 2 == 0) {
            s++;
            t /= 2;
        }
        long aInverse = Task3.modInverse(a, mod);
        long c = Task2.binaryModPow(b, t, mod);
        long r = Task2.binaryModPow(a, (t + 1) / 2, mod);

        for (int i = 1; i <= s - 1; i++) {
            //dPow = 2 ** (s-i-1)
            long dPow = 1;
            for (int i2 = 1; i2 <= s - i - 1; i2++)
                dPow *= 2;
            long d = Task2.binaryModPow((r * r * aInverse) % mod, dPow, mod);
            if (d % mod == mod - 1) {
                r = (r * c) % mod;
            }
            c = c*c % mod;
        }
        return r;
```

```java
    }
    public static void main(String[] args) {
        int mod = 29;

        for (long x : new long[]{39, 30, 18, 5811988, 4964974, 6174, 160272279849904686L,
27342113381205372L, 20268}) {
            try {
                long sqrt = squareRootMod(x, mod);
                System.out.printf("sqrt(%d) mod %d: %d and %d\n", x, mod, sqrt, mod - sqrt);
            } catch (ArithmeticException e) {
                System.out.println("Error for " + x + ": " + e.getMessage());
            }
        }
        // Extra examples
        int x = 4;
        mod = 5;
        long sqrt = squareRootMod(x, mod);
        System.out.printf("sqrt(%d) mod %d: %d and %d\n", x, mod, sqrt, mod - sqrt);

        x = 81;
        mod = 11;
        sqrt = squareRootMod(x, mod);
        System.out.printf("sqrt(%d) mod %d: %d and %d\n", x, mod, sqrt, mod - sqrt);

        x = 81;
        mod = 5;
        sqrt = squareRootMod(x, mod);
        System.out.printf("sqrt(%d) mod %d: %d and %d\n", x, mod, sqrt, mod - sqrt);
    }
}
```

Результати

```
Error for 39: a does not have square root modulo 29
sqrt(30) mod 29: 1 and 28
Error for 18: a does not have square root modulo 29
Error for 5811988: a does not have square root modulo 29
Error for 4964974: 0 does not have square root modulo 29
Error for 6174: a does not have square root modulo 29
Error for 160272279849904686: a does not have square root modulo 29
Error for 27342113381205372: a does not have square root modulo 29
Error for 20268: a does not have square root modulo 29
```

Бачимо, що надані значення a, b, c трохи “невдалі”, тому розглянемо додаткові приклади:

```
sqrt(4) mod 5: 3 and 2
sqrt(81) mod 11: 9 and 2
sqrt(81) mod 5: 1 and 4
```

Завдання 5.

Використовуючи нижченаведені дані, обчислити функцію Ойлера для

1) a=366, b=431, c= 739.
2) a=636, b=1401, c= 3974.
3) a=13!, b=8!, c= 21!.
4) a=36!, b=43!, c= 73!.

Теорія

Функція Ойлера φ(n) визначається для всіх натуральних чисел n, значенням якої є кількість натуральних чисел, менших від n і взаємно простих з n. Для n = 1 покладають φ(1) = 1.

Для невеликих значень n значення φ(n) можна знайти простим підрахунком.

Для обчислення значення φ(n) для довільного n існує формула, за якою ці значення знаходяться.

$$\varphi(n) = \frac{n}{p_1 p_2 \ldots p_k}(p_1 - 1)(p_2 - 1)\ldots(p_k - 1) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})\ldots(1 - \frac{1}{p_k}),$$

Код програми

Скористаємося класами BigInteger та BigDecimal стандартної бібліотеки Java, оскільки, наприклад, 36! не вміщається в ціле 64-бітне число.

```
import java.math.BigDecimal;
import java.math.BigInteger;
import java.math.MathContext;
import java.math.RoundingMode;

public class Task5 {
    static BigInteger euler(BigInteger n) {
        MathContext context = new MathContext(1000, RoundingMode.FLOOR);

        BigDecimal result = new BigDecimal(n, context);

        // p = 2; p * p <= n; p++
        for (BigInteger p = BigInteger.TWO; p.multiply(p).compareTo(n) <= 0; p =
p.add(BigInteger.ONE)) {
            // n % p == 0
            if (n.mod(p).compareTo(BigInteger.ZERO) == 0) {
                while (n.mod(p).compareTo(BigInteger.ZERO) == 0)
                    n = n.divide(p);
                // result *= (1 - 1 / p)
                result = result.multiply(BigDecimal.ONE.subtract(BigDecimal.ONE.divide(new
BigDecimal(p), context)));
            }
        }
        // if (n > 1) result *= 1 - 1 / n
        if (n.compareTo(BigInteger.ONE) > 0)
            result = result.multiply(BigDecimal.ONE.subtract(BigDecimal.ONE.divide(new
BigDecimal(n), context)));
        return result.toBigInteger();
    }

    static BigInteger factorial(int n) {
        BigInteger result = BigInteger.ONE;
        for (int i = 2; i <= n; i++)
```

```
            result = result.multiply(BigInteger.valueOf(i));
        return result;
    }

    public static void main(String[] args) {
        for (BigInteger x : new BigInteger[]{
                BigInteger.valueOf(366),
                BigInteger.valueOf(431),
                BigInteger.valueOf(739),

                BigInteger.valueOf(636),
                BigInteger.valueOf(1401),
                BigInteger.valueOf(3974),

                factorial(13),
                factorial(8),
                factorial(21),

                factorial(36),
                factorial(43),
                factorial(73),
        }) {
            System.out.printf("Euler(%d) = \n   %d\n", x, euler(x));
        }
    }
}
```

Результати

```
Euler(366) =
   120
Euler(431) =
   430
Euler(739) =
   738
Euler(636) =
   208
Euler(1401) =
   932
Euler(3974) =
   1986
Euler(6227020800) =
   1194393600
Euler(40320) =
   9216
Euler(51090942171709440000) =
   8737778439290880000
Euler(371993326789901217467999448150835200000000) =
   56859980302182536397372832874496000000000
Euler(60415263063373835637355132068513997507264512000000000) =
   8562014771617157265317326095157560134860800000000000
Euler(44701154615126843408912571381250511100768007002829050158190800923704221040671
83317016903680000000000000000) =

56344476525421120344653630343366055835490106978126626568385219382512049604997016453
1200000000000000000000
```

Результати перевірені за допомогою Wolfram Alpha.

Завдання 6.

Знайти розклад на прості множники таких чисел:
1) a=36б, b=431, c= 739.
2) a=36б6, b=4313, c= 7399.
3) a=6б!, b=31!, c= 39!.

Теорія

Скористаємося очевидним ітеративним алгоритмом.

Код програми

```
import java.math.BigInteger;

public class Task6 {
    static void printPrimes(BigInteger n) {
        System.out.println("Divisors of " + n + ':');
        BigInteger c = BigInteger.TWO;

        while (n.compareTo(BigInteger.ONE) > 0) { // n > 1
            if (n.mod(c).compareTo(BigInteger.ZERO) == 0) { // n % c == 0
                System.out.print(c + ", ");
                n = n.divide(c);
            } else {
                c = c.add(BigInteger.ONE);
            }
        }
        System.out.println();
    }

    public static void main(String[] args) {
        for (BigInteger x : new BigInteger[]{
                BigInteger.valueOf(366),
                BigInteger.valueOf(431),
                BigInteger.valueOf(739),

                BigInteger.valueOf(3666),
                BigInteger.valueOf(4313),
                BigInteger.valueOf(7399),

                Task5.factorial(66),
                Task5.factorial(31),
                Task5.factorial(39),
        }) {
            printPrimes(x);
        }
    }
}
```

Результати

```
Divisors of 366:
2, 3, 61,
Divisors of 431:
```

```
431,
Divisors of 739:
739,
Divisors of 3666:
2, 3, 13, 47,
Divisors of 4313:
19, 227,
Divisors of 7399:
7, 7, 151,
Divisors of
5443449390774430640037292402478427526442930643887988745328601268696710811484160000
0000000000:
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 7, 7,
7, 7, 7, 7, 7, 7, 7, 7, 11, 11, 11, 11, 11, 11, 13, 13, 13, 13, 13, 17, 17, 17, 19,
19, 19, 23, 23, 29, 29, 31, 31, 37, 41, 43, 47, 53, 59, 61,
Divisors of 822283865417792281772556288000000:
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 7, 7, 7, 7, 11, 11, 13,
13, 17, 19, 23, 29, 31,
Divisors of 2039788208119744335864028173990289735680000000:
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 5, 5,
5, 5, 5, 5, 5, 7, 7, 7, 7, 7, 11, 11, 11, 13, 13, 13, 17, 17, 19, 19, 23, 29, 31,
37,
```

Алгоритм працює достатньо швидко навіть для значення 66! = 544344...

Перелік літературних джерел

1. Конспект лекцій «Математичні основи захисту інформації» С.Л. Кривий
2. Menezes A., van Oorschot P., Vanstons S. Handbook of Applied Cryptography. – CRC Press. – 1996. – 661 p.