Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-технічний інститут

Спеціальні розділи обчислювальної математики

Виконав:
Студент гр. ФБ-23 Моісеєнко Дмитро

Київ-2025

Комп'ютерний практикум №2. Багаторозрядна модулярна арифметика

Мета роботи: Отримання практичних навичок програмної реалізації багаторозрядної арифметики; ознайомлення з прийомами ефективної реалізації критичних по часу ділянок програмного коду та методами оцінки їх ефективності.

3. Завдання до комп'ютерного практикуму

А) Доопрацювати бібліотеку для роботи з m-бітними цілими числами, створену на

комп'ютерному практикумі No1, додавши до неї такі операції:

1) обчислення НСД та НСК двох чисел;

2) додавання чисел за модулем;

3) віднімання чисел за модулем;

4) множення чисел та піднесення чисел до квадрату за модулем;

5) піднесення числа до багаторозрядного степеня d по модулю n .

Модулярну арифметику рекомендовано реалізовувати на базі редукції Баррета, піднесення до степеня – на базі схеми Горнера. Мова програмування, семантика функцій та спосіб реалізації можуть обиратись довільним чином.

Хід виконання роботи:

Загальні алгоритми в коді:

C: > Users > Dmytro_21 > AppData > Local > Programs > Python > Python312 > Lib > site-packages > Laba2.py > ...

```python
# 1. Звичайний алгоритм Евкліда
def gcd_euclidean(a: int, b: int) -> int:
    while b != 0:
        a, b = b, a % b
    return a

# 2. Розширений алгоритм Евкліда
def extended_gcd(a: int, b: int) -> tuple[int, int, int]:
    if b == 0:
        return a, 1, 0
    else:
        d, x1, y1 = extended_gcd(b, a % b)
        x = y1
        y = x1 - (a // b) * y1
        return d, x, y

# 3. Бінарний алгоритм Евкліда (Стейна)
def gcd_stein(a: int, b: int) -> int:
    if a == 0:
        return b
    if b == 0:
        return a

    k = 0
    while (a | b) & 1 == 0:
        a >>= 1
        b >>= 1
        k += 1

    while a & 1 == 0:
        a >>= 1

    while b != 0:
        while b & 1 == 0:
            b >>= 1
        if a > b:
            a, b = b, a
        b -= a

    return a << k

```

C: > Users > Dmytro_21 > AppData > Local > Programs > Python > Python312 > Lib > site-packages > Laba2.py > ...

```python
# 4. Барретт редукція
def barrett_reduce(x: int, m: int) -> int:
    k = m.bit_length()
    mu = (1 << (2 * k)) // m
    q = (x * mu) >> (2 * k)
    r = x - q * m
    while r >= m:
        r -= m
    return r

# 5. Монтгомері редукція
def montgomery_reduce(t: int, n: int, n_inv: int, r: int) -> int:
    m = ((t % r) * n_inv) % r
    u = (t + m * n) // r
    if u >= n:
        u -= n
    return u

# Монтгомері множення
def montgomery_mul(a: int, b: int, n: int) -> int:
    r = 1 << (n.bit_length() + 1)
    r_inv = pow(r, -1, n)
    n_inv = -pow(n, -1, r) % r

    a_mont = (a * r) % n
    b_mont = (b * r) % n
    t = a_mont * b_mont
    ab = montgomery_reduce(t, n, n_inv, r)
    return (ab * r_inv) % n

# 6. Модульне піднесення до степеня з методом Монтгомері
def modexp_montgomery(base: int, exponent: int, modulus: int) -> int:
    if modulus == 1:
        return 0
    r = 1 << (modulus.bit_length() + 1)
    r_inv = pow(r, -1, modulus)
    n_inv = -pow(modulus, -1, r) % r

    base_mont = (base * r) % modulus
    result_mont = (1 * r) % modulus
```

```python
# 6. Модульне піднесення до степеня з методом Монтгомері
def modexp_montgomery(base: int, exponent: int, modulus: int) -> int:
    if modulus == 1:
        return 0
    r = 1 << (modulus.bit_length() + 1)
    r_inv = pow(r, -1, modulus)
    n_inv = -pow(modulus, -1, r) % r

    base_mont = (base * r) % modulus
    result_mont = (1 * r) % modulus

    while exponent > 0:
        if exponent & 1:
            result_mont = montgomery_reduce(result_mont * base_mont, modulus, n_inv, r)
        base_mont = montgomery_reduce(base_mont * base_mont, modulus, n_inv, r)
        exponent >>= 1

    return (result_mont * r_inv) % modulus
```

Основна функція коді для виконання:

```
# Основна функція
def main():
    a = 123456789123456789123456789
    b = 987654321987654321987654321
    m = 1000000007
    e = 123456

    print("Вхідні значення:")
    print("a =", a)
    print("b =", b)
    print("modulus =", m)
    print("exponent =", e)
    print()

    print("Алгоритм Евкліда: GCD =", gcd_euclidean(a, b))
    print("Розширений алгоритм Евкліда: ", end="")
    d, x, y = extended_gcd(a, b)
    print(f"GCD = {d}, x = {x}, y = {y}")
    print("Бінарний алгоритм Евкліда (Стейна): GCD =", gcd_stein(a, b))
    print("Баррет редукція a*b mod m =", barrett_reduce(a * b, m))
    print("Монтгомері множення a*b mod m =", montgomery_mul(a, b, m))
    print("Монтгомері піднесення до степеня a^e mod m =", modexp_montgomery(a, e, m))

if __name__ == "__main__":
    main()
```

Результат:

```
C:\Users\Dmytro_21\AppData\Local\Programs\Python\Python312\Lib\site-packages>python Laba2.py
Вхідні значення:
a = 123456789123456789123456789
b = 987654321987654321987654321
modulus = 1000000007
exponent = 123456

Алгоритм Евкліда: GCD = 9000000009000000009
Розширений алгоритм Евкліда: GCD = 9000000009000000009, x = -8, y = 1
Бінарний алгоритм Евкліда (Стейна): GCD = 9000000009000000009
```

Висновок: У цій лабораторній роботі було реалізовано та перевірено алгоритми модулярної арифметики для багаторозрядних чисел: алгоритм Евкліда, розширений алгоритм Евкліда, Баррет редукцію, Монтгомері множення та модульне піднесення до степеня.