

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI



BÀI TẬP LỚN BỘ MÔN: Kỹ thuật lập trình

Đề tài: Xây dựng chương trình sinh số nguyên tố (lớn hơn N cho trước)
Xây dựng thuật toán kiểm tra tính nguyên tố Miller Rabin
Áp dụng cài đặt thuật toán mã hóa RSA

Giảng viên: **Thầy Vũ Thành Nam**

Lớp : Toán Tin 02- K66
Họ và tên: Nguyễn Quốc Anh
MSSV: 20210061

Hà Nội, năm 2023

MỤC LỤC

I. Giới thiệu thuật toán:	3
1. Về thuật toán Miller_Rabin:	3
Kiểm tra Miller-Rabin:	3
Tiêu chuẩn Miller-Rabin:	3
2. Về thuật toán RSA:	4
Lịch sử	4
Tạo khóa	4
Mã hóa	5
Giải mã	5
II. Cụ thể hơn về chương trình, chạy input và output:	5
1. Xây dựng chương trình:	5
Module Miller_Rabin:	5
Chương trình thực hiện toàn bộ yêu cầu bài toán:	7
2. Một số nguyên tắc tạo lập chương trình “tốt” đã dùng:	10
Output:	10
3. Kiểm thử:	13

I. Giới thiệu thuật toán:

1. Về thuật toán Miller_Rabin:

Kiểm tra Miller-Rabin:

Là một thuật toán xác suất để kiểm tra tính nguyên tố cũng như các thuật toán kiểm tra tính nguyên tố: Kiểm tra Fermat và Kiểm tra Solovay-Strassen. Nó được đề xuất đầu tiên bởi Gary L. Miller như một thuật toán tất định, dựa trên giả thiết Riemann tổng quát; Michael O. Rabin đã sửa chữa nó thành một thuật toán xác suất.

Khi sử dụng kiểm tra Miller-Rabin chúng ta căn cứ vào một mệnh đề $Q(p, a)$ đúng với các số nguyên tố p và mọi số tự nhiên a thuộc A và kiểm tra xem chúng có đúng với số n muốn kiểm tra và một số a thuộc A được chọn ngẫu nhiên hay không. Nếu mệnh đề $Q(n, a)$ không đúng, tất yếu n không phải là số nguyên tố, còn nếu $Q(n, a)$ đúng, số n có thể là số nguyên tố với một xác suất nào đó. Khi tăng số lần thử, xác suất để n là số nguyên tố tăng lên.

Tiêu chuẩn Miller-Rabin:

Bây giờ giả sử p là một số nguyên tố lẻ, khi đó $p - 1$ là số chẵn và ta có thể viết $p - 1$ dưới dạng $2s * m$, trong đó s là một số tự nhiên ≥ 1 và m là số lẻ - Điều này nghĩa là ta rút hết các thừa số 2 khỏi $p - 1$. Lấy số a bất kỳ trong tập $\{1, 2, \dots, p-1\}$.

Xét dãy số $x_k = a^{2^k * m}$ với $k = 0, 1, 2, \dots, s$. Khi đó $x_k = (x_{k-1})^2$, với $k = \{1, 2, \dots, s\}$ và $x_s = a^{p-1}$

Từ định lý Fermat nhỏ:

$$a^{p-1} \equiv 1 \pmod{p}$$

hay

$$x_s \equiv 1 \pmod{p}$$

hay

$$x_{s-1}^2 \equiv 1 \pmod{p}$$

Do đó, hoặc $x_{s-1} \equiv 1 \pmod{p}$ hoặc $x_{s-1} \equiv -1 \pmod{p}$.

Nếu $x_{s-1} \equiv -1$ ta dừng lại, còn nếu ngược lại ta tiếp tục với x_{s-2} .

Sau một số hữu hạn bước

- hoặc ta có một chỉ số k , $0 \leq k \leq s - 1$ sao cho $x_k \equiv -1 \pmod{p}$
- hoặc với $k = 0$ ta vẫn có $x_k \equiv 1 \pmod{p}$

Ta có mệnh đề $Q(p, a)$ như sau:

*Nếu p là số nguyên tố lẻ và $p - 1 = 2^s * m$ thì với mọi a : $0 < a < p-1$*

- hoặc $x_k = a^{2^k * m} \equiv 1 \pmod{p}$ với mọi $k = 0, 1, 2, \dots, s$
- hoặc tồn tại k : $0 \leq k \leq s$ sao cho $x_k = a^{2^k * m} \equiv -1 \pmod{p}$

2. Về thuật toán RSA

Trong mật mã học, **RSA** là một thuật toán mật mã hóa khóa công khai. Đây là thuật toán đầu tiên phù hợp với việc tạo ra chữ ký điện tử đồng thời với việc mã hóa. Nó đánh dấu một sự tiến bộ vượt bậc của lĩnh vực mật mã học trong việc sử dụng khóa công cộng. RSA đang được sử dụng phổ biến trong thương mại điện tử và được cho là đảm bảo an toàn với điều kiện độ dài khóa đủ lớn.

Lịch sử

Thuật toán được Ron Rivest, Adi Shamir và Len Adleman mô tả lần đầu tiên vào năm 1977 tại Học viện Công nghệ Massachusetts (MIT). Tên của thuật toán lấy từ 3 chữ cái đầu của tên 3 tác giả.

Trước đó, vào năm 1973, Clifford Cocks, một nhà toán học người Anh làm việc tại GCHQ, đã mô tả một thuật toán tương tự. Với khả năng tính toán tại thời điểm đó thì thuật toán này không khả thi và chưa bao giờ được thực nghiệm. Tuy nhiên, phát minh này chỉ được công bố vào năm 1997 vì được xếp vào loại tuyệt mật.

Thuật toán RSA có hai khóa: khóa công khai (hay khóa công cộng) và khóa bí mật (hay khóa cá nhân). Mỗi khóa là những số cố định sử dụng trong quá trình mã hóa và giải mã. Khóa công khai được công bố rộng rãi cho mọi người và được dùng để mã hóa. Những thông tin được mã hóa bằng khóa công khai chỉ có thể được giải mã bằng khóa bí mật tương ứng. Nói cách khác, mọi người đều có thể mã hóa nhưng chỉ có người biết khóa cá nhân (bí mật) mới có thể giải mã được.

Ta có thể mô phỏng trực quan một hệ mật mã khóa công khai như sau: Bình muốn gửi cho An một thông tin mật mà Bình muốn duy nhất An có thể đọc được. Để làm được điều này, An gửi cho Bình một chiếc hộp có khóa đã mở sẵn và giữ lại chìa khóa. Bình nhận chiếc hộp, cho vào đó một tờ giấy viết thư bình thường và khóa lại (như loại khóa thông thường chỉ cần sập chốt lại, sau khi sập chốt khóa ngay cả Bình cũng không thể mở lại được-không đọc lại hay sửa thông tin trong thư được nữa). Sau đó Bình gửi chiếc hộp lại cho An. An mở hộp với chìa khóa của mình và đọc thông tin trong thư. Trong ví dụ này, chiếc hộp với khóa mở đóng vai trò khóa công khai, chiếc chìa khóa chính là khóa bí mật.

Tạo khóa

Giả sử An và Bình cần trao đổi thông tin bí mật thông qua một kênh không an toàn (ví dụ như Internet). Với thuật toán RSA, An đầu tiên cần tạo ra cho mình cặp khóa gồm khóa công khai và khóa bí mật theo các bước sau:

1. Chọn 2 số nguyên tố lớn p và q với , lựa chọn ngẫu nhiên và độc lập.
2. Tính: $n = p * q$
3. Tính: giá trị hàm số Euler : $\phi(n) = (p - 1)(q - 1)$
4. Chọn một số tự nhiên e sao cho $1 < e < \phi(n)$ và nguyên tố cùng nhau với $\phi(n)$
5. Tính: d sao cho $d * e \equiv 1 \pmod{\phi(n)}$

Khóa công khai bao gồm:

- n , module
- e , số mũ công khai (cũng gọi là *số mũ mã hóa*).

Khóa bí mật bao gồm:

- n , module, xuất hiện cả trong khóa công khai và khóa bí mật, và
- d , số mũ bí mật (cũng gọi là *số mũ giải mã*).

Mã hóa

Giả sử Bình muốn gửi đoạn thông tin M cho An. Đầu tiên Bình chuyển M thành một số $m < n$ theo một hàm có thể đảo ngược (từ m có thể xác định lại M) được thỏa thuận trước. Quá trình này trong thực tế đã được xử lý bởi 1 bộ mã mà thường là ASCII hoặc thông dụng hơn là Unicode

Lúc này Bình có m và biết n cũng như e do An gửi. Bình sẽ tính c là bản mã hóa của m theo công thức:

Hàm trên có thể tính dễ dàng sử dụng phương pháp tính hàm mũ (theo môđun) bằng (thuật toán bình phương và nhân) Cuối cùng Bình gửi c cho An.

Giải mã

An nhận c từ Bình và biết khóa bí mật d . An có thể tìm được m từ c theo công thức sau:

$$m = c^d \bmod n$$

Biết m , An tìm lại M theo phương pháp đã thỏa thuận trước. Quá trình giải mã hoạt động vì ta có

$$c \equiv (m^e)^d \pmod{n}.$$

Do $ed \equiv 1 \pmod{p-1}$ và $ed \equiv 1 \pmod{q-1}$, (theo Định lý Fermat bé) nên:

$$m^{ed} \equiv m \pmod{p}$$

Và

$$m^{ed} \equiv m \pmod{q}$$

Do p và q là hai số nguyên tố cùng nhau, áp dụng định lý phần dư Trung Hoa, ta có:

$$m^{ed} \equiv m \pmod{pq}$$

hay

$$c^d = m \pmod{n}$$

II. Cụ thể hơn về chương trình, chạy input và output:

1. Xây dựng chương trình

Xét cấu trúc tổng thể của chương trình, ta có các bước thực hiện như sau:

Module Miller_Rabin:

1. Thiết lập hàm modular_power function tính module lũy thừa $(x^y) \% p$ bằng thuật toán nhân lũy thừa

```
def modular_power(x, y, p):
    result = 1;

    # Cập nhật giá trị của x nếu x >= p
    x = x % p;
    while (y > 0):

        # Nếu y lẻ, nhân x vào biến result, tính result mod p
        if (y & 1):
            result = (result * x) % p
        # Nếu y chẵn, tính x^2 mod p
        y = int(y/2)
        x = (x * x) % p
```

- Thiết lập hàm millerTest chạy thuật toán Miller-Rabin cho 1 số thử a (chọn ngẫu nhiên trong đoạn $[2, n-2]$ ($n > 4$) và 1 số lẻ d. Hàm này kiểm tra liệu $a^d \% n$ có nhận giá trị 1 hay $n - 1$ không. Nếu có thì n có thể là 1 số nguyên tố

```
def miillerTest(d, n):

    # Chọn số a bất kì thuộc [2,n-2]
    # Hàm isPrime ở dưới cần đảm bảo n > 4
    a = 2 + random.randint(1, n - 4)

    # Compute a^d % n
    x = modular_power(a, d, n)

    if (x == 1 or x == n - 1):
        return True;

    # Tiếp tục bình phương x cho đến khi 1 trong 3 điều kiện sau không thỏa mãn:
    # (i) d != n-1
    # (ii) (x^2) % n != 1
    # (iii) (x^2) % n != n-1
    while (d != n - 1):
        x = (x * x) % n
        d *= 2;

        if (x == 1):
            return False;
        if (x == n - 1):
            return True;

    # Gặp được số phức, trả về false
    return False;
```

- Thiết lập hàm isPrime cho phép chạy thuật toán Miller-Rabin với độ chính xác k.

Xử lý trước trường hợp $n \leq 4$

```
def isPrime(n, k):

    # Loại bỏ trước 1 số trường hợp đặc biệt của input,
    # đồng thời là để đảm bảo điều kiện thực thi cho hàm millerTest ở trên (n > 4)

    if (n <= 1 or n == 4):
        return False;
    if (n <= 3):
        return True;
```

Hàm này sau đó tìm số r thỏa mãn $n = 2^u * r + 1$, với r lẻ, $r \geq 1$

```

# Tìm r thỏa mãn n = 2^d * r + 1 với r >= 1
d = n - 1;
while (d % 2 == 0):
    d //= 2;

# Chạy phép thử Miller-Rabin k lần đối với r, n (lúc này d chính là r)
for i in range(k):
    if (miillerTest(d, n) == False):
        return False;

return True;

```

Sau đó, hàm millerTest được chạy k lần trong hàm isPrime the Miller-Rabin với số thử a và số r vừa tìm được. Hàm trả về False với bất kì trường hợp nào không thỏa mãn với số thử a; trả về True nếu tất cả các số thử a đều thỏa mãn. Điều này chứng tỏ n có thể là số nguyên tố

4. (Kiểm thử) Chạy thử đoạn code để tìm ra các số nguyên tố ≤ 2000

```

# Xóa dấu comment ở phần này nếu muốn kiểm thử lại thuật toán
'''print("All primes smaller than 2000: ");
for n in range(1,2000):
    if (isPrime(n, k)):
        print(n , end=" ")'''

```

Chương trình thực hiện toàn bộ yêu cầu bài toán:

1. Nhập module Miller_Rabin, trong đó quan trọng nhất là hàm isPrime dùng để xét tính nguyên tố & tạo lập tập số nguyên tố
2. Khai báo biến prime dùng để lưu các số nguyên tố sinh ra, định dạng tập hợp, khởi tạo giá trị rỗng

```

prime = set()

public_key = None
private_key = None
n = None

```

3. Khai báo biến toàn cục public_key, private_key, and n, khởi tạo giá trị ban đầu None.
4. Sử dụng hàm primefiller() làm đầy tập prime với hàm isPrime, số n được lựa chọn cùng độ chính xác k cho trước (ở đây chọn n=250, k=10)

```

def primefiller(n, k):
    global prime
    for i in range(n):
        if (isPrime(i, k)):
            prime.add(i)

```

5. Chọn ra 2 số nguyên tố p,q khác nhau bằng cách sử dụng hàm pickrandomprime 2 lần

```
def pickrandomprime():
    global prime
    k = random.randint(0, len(prime) - 1)
    it = iter(prime)
    for _ in range(k):
        next(it)

    ret = next(it)
    prime.remove(ret)
    return ret

#Tạo lập khóa công khai và khóa bí mật
#Các bước thực hiện sẽ ghi ở bên dưới

def setkeys():
    global public_key, private_key, n
    # Step 1: Chọn ra 2 số nguyên tố p và q khác nhau

    prime1 = pickrandomprime()
    prime2 = pickrandomprime()
```

6. Gán giá trị cho public_key, private_key theo lý thuyết (e: public; d: private)

```
n = prime1 * prime2
fi = (prime1 - 1) * (prime2 - 1)

#Step 3: Chọn số e nguyên tố cùng nhau với  $\Phi(n)$ , lấy e làm khóa công khai
e = 2
while True:
    if math.gcd(e, fi) == 1:
        break
    e += 1

public_key = e

d = 2
while True:
    if (d * e) % fi == 1:
        break
    d += 1

private_key = d
```

7. Mã hóa văn bản bằng hàm encrypt.

```
def encrypt(message):
    global public_key, n
    e = public_key
    encrypted_text = 1
    while e > 0:
        encrypted_text *= message
        encrypted_text %= n
        e -= 1
    return encrypted_text
```

8. Giải mã văn bản bằng hàm decrypt


```
def decrypt(encrypted_text):
    global private_key, n
    d = private_key
    decrypted = 1
    while d > 0:
        decrypted *= encrypted_text
        decrypted %= n
        d -= 1
    return decrypted
```

9. Sử dụng hàm encrypt, mã hóa thông tin văn bản bằng hàm encoder thông qua giá trị Unicode của kí tự
10. Trả lại giá trị Unicode của kí tự đã được mã hóa thông qua hàm decoder

```
def encoder(message):
    encoded = []
    # Gọi hàm mã hóa
    for letter in message:
        encoded.append(encrypt(ord(letter)))
    return encoded

def decoder(encoded):
    s = ''
    # Gọi hàm giải mã, xếp lại các kí tự tạo thành chuỗi
    for num in encoded:
        s += chr(decrypt(num))
    return s
```

11. Sử dụng hàm isPrime của giải thuật Miller_Rabin kết hợp với 1 vòng lặp while để tìm ra số nguyên tố gần nhất lớn hơn số tự nhiên n cho trước. Đặt tên cho hàm thực hiện tác vụ này là find_next_prime

```
def find_next_prime(n):
    next_prime = n + 1
    while not isPrime(next_prime, 10):
        next_prime += 1
    return next_prime
```

12. Sử dụng hàm isPrime của giải thuật Miller_Rabin kết hợp 1 vòng lặp for để sinh các số nguyên tố nhỏ hơn số tự nhiên n cho trước. Đặt tên cho hàm thực hiện tác vụ này là return_all_primes

```
def return_all_primes(n):
    k = 10
    primes = []
    for i in range(2, n):
        if (isPrime(i, k)):
            primes.append(i)
    return primes
```

13. Thiết kế Menu cho các tùy chọn & tác vụ:

- Tìm số nguyên tố tiếp theo lớn hơn n
- Mã hóa & giải mã file sử dụng giải thuật RSA
- Trả về các số nguyên tố nhỏ hơn n với giải thuật Miller-Rabin

- Trả về kết quả thuật toán RSA Test cho input đã cài sẵn (đồng thời phục vụ kiểm thử)
- Trả về kết quả thuật toán RSA cho input nhập từ Command Prompt
- Thoát khỏi chương trình
- In ra kết quả trên màn hình Command Prompt

2. Một số nguyên tắc tạo lập chương trình “tốt” đã dùng:

- Giải quyết bài toán theo hướng top-down:
 - Chia nhỏ bài toán lớn là xây dựng thuật toán mã hóa thành các bài toán nhỏ:
 - Giải thuật Miller_Rabin:
 - Cách tính lũy thừa module
 - Xây dựng phương thức kiểm tra số nguyên tố
 - Xây dựng hàm cho phép phương thức trên lặp đi lặp lại k lần (k là độ chính xác của thuật toán)
 - Sinh số nguyên tố bằng giải thuật Miller_Rabin
 - Chuyển kí tự thành mã ASCII/Unicode sau đó mã hóa
 - Xử lý đầu vào:
 - Nhập tay
 - Dạng file
- Đặt tên biến, hàm có nguyên tắc, dễ hiểu, phù hợp với mục đích của biến – hàm
- Có chú thích tại mỗi bước của hàm/ khối hàm/ đầu và cuối chương trình, chỉ rõ input/output cũng như mục đích của từng bước lập trình
- Tái sử dụng tài nguyên sẵn có:
 - Đối với xử lý output của giải thuật Miller_Rabin: Sử dụng hàm bubbleSort có sẵn trong Bài thực hành số 5 để sắp xếp output
 - Đối với yêu cầu sinh số nguyên tố lớn hơn n: sử dụng hàm isPrime của Miller_Rabin (vấn đề này trong mã nguồn em đã có chú thích cụ thể)

Output:

Option 1:

```
Menu:
Enter your choice of input:
1. Find the next prime number greater than n:
2. Decode a file using the RSA method:
3. Return all prime numbers using the Miller-Rabin method:
4. RSA Test for built-in Input
5. RSA Test for manually typed in Input:
6. Exit
Enter your choice: 1
Enter the value of n: 2032455234
Next prime number greater than n: 2032455277
```

Option 2:

```

Menu:
Enter your choice of input:
1. Find the next prime number greater than n:
2. Encode and Decode a file using the RSA method:
3. Return all prime numbers using the Miller-Rabin method:
4. RSA Test for built-in Input
5. RSA Test for manually typed in Input:
6. Exit
Enter your choice: 2
Encoded message:
[249, 393, 718, 718, 947, 11, 461, 791, 550, 461, 718, 947, 539, 163, 461, 163,
947, 539, 393, 461, 951, 317, 447, 393, 539, 441, 415, 461, 354, 370, 579, 488,
461, 245, 110, 370, 948, 245, 718, 718, 550, 461, 245, 791, 245, 782, 447, 539,
163, 461, 370, 947, 461, 488, 393, 393, 461, 550, 947, 948, 461, 245, 163, 245,
447, 539, 461, 488, 947, 461, 393, 539, 393, 317, 163, 393, 370, 447, 110, 461,
245, 539, 441, 461, 791, 947, 370, 447, 270, 245, 370, 393, 441, 461, 300, 447,
370, 972, 461, 300, 972, 245, 370, 393, 270, 393, 317, 461, 550, 947, 948, 579,
317, 393, 461, 441, 947, 447, 539, 163, 415, 461, 292, 488, 461, 667, 944, 78, 5
50, 393, 245, 317, 78, 947, 718, 441, 488, 461, 791, 948, 488, 370, 461, 539, 94
7, 300, 461, 718, 393, 245, 317, 539, 461, 370, 947, 461, 947, 270, 393, 317, 11
0, 947, 791, 393, 461, 947, 373, 488, 370, 245, 110, 718, 393, 488, 461, 163, 31
7, 393, 245, 370, 393, 317, 461, 370, 972, 245, 539, 461, 393, 270, 393, 317, 46
1, 373, 393, 951, 947, 317, 393, 11, 461, 370, 947, 461, 393, 488, 370, 245, 373
, 718, 447, 488, 972, 461, 947, 948, 317, 461, 947, 300, 539, 461, 300, 393, 245
, 718, 370, 972, 11, 461, 488, 370, 317, 393, 539, 163, 370, 972, 461, 245, 539,
441, 461, 441, 447, 163, 539, 447, 370, 550, 415, 461, 354, 461, 552, 948, 488,
370, 461, 972, 947, 63, 393, 461, 370, 972, 245, 370, 461, 550, 947, 948, 461,
300, 447, 718, 718, 461, 373, 393, 461, 370, 972, 393, 461, 488, 245, 791, 393,
461, 300, 447, 370, 972, 447, 539, 461, 370, 972, 393, 461, 539, 393, 65, 370, 4
61, 667, 944, 461, 550, 393, 245, 317, 488, 11, 461, 947, 317, 461, 245, 370, 46
1, 718, 393, 245, 488, 370, 461, 370, 447, 718, 718, 461, 947, 948, 317, 461, 53
9, 393, 65, 370, 461, 317, 393, 948, 539, 447, 947, 539, 415, 461]
Decoded message:
Hello, my long gone friend. It's actually amazing to see you again so energetic
and motivated with whatever you're doing. Us 20-year-olds must now learn to over
come obstacles greater than ever before, to establish our own wealth, strength a
nd dignity. I just hope that you will be the same within the next 20 years, or a
t least till our next reunion.

```

Option 3:

```

Menu:
Enter your choice of input:
1. Find the next prime number greater than n:
2. Encode and Decode a file using the RSA method:
3. Return all prime numbers using the Miller-Rabin method:
4. RSA Test for built-in Input
5. RSA Test for manually typed in Input:
6. Exit
Enter your choice: 3
Enter the value of n: 234
All prime numbers:
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163
, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233]

```

Option 4:

```

Menu:
Enter your choice of input:
1. Find the next prime number greater than n:
2. Encode and Decode a file using the RSA method:
3. Return all prime numbers using the Miller-Rabin method:
4. RSA Test for built-in Input
5. RSA Test for manually typed in Input:
6. Exit
Enter your choice: 4
Initial message:
My dear friend, I will always be there for you

The encoded message(encrypted by public key)
5064; 2619; 3494; 5326; 6098; 6143; 8988; 3494; 213; 8988; 2136; 6098; 6493; 532
6; 3007; 3494; 4355; 3494; 7806; 2136; 8002; 8002; 3494; 6143; 8002; 7806; 6143;
2619; 285; 3494; 9604; 6098; 3494; 6994; 6817; 6098; 8988; 6098; 3494; 213; 522
6; 8988; 3494; 2619; 5226; 2677

The decoded message(decrypted by public key)
My dear friend, I will always be there for you

```

Option 5:

```

3. Return all prime numbers using the Miller-Rabin method:
4. RSA Test for built-in Input
5. RSA Test for manually typed in Input:
6. Exit
Enter your choice: 5
Enter the message
Hello and welcome to the programming world!

Initial message:
Hello and welcome to the programming world!

The encoded message(encrypted by public key)
884; 6098; 8002; 8002; 5226; 3494; 6143; 6493; 5326; 3494; 7806; 6098; 8002; 153
0; 5226; 1063; 6098; 3494; 6994; 5226; 3494; 6994; 6817; 6098; 3494; 6279; 8988;
5226; 5404; 8988; 6143; 1063; 1063; 2136; 6493; 5404; 3494; 7806; 5226; 8988; 8
002; 5326; 326

The decoded message(decrypted by public key)
Hello and welcome to the programming world!

Menu:

```

Option 6:

```

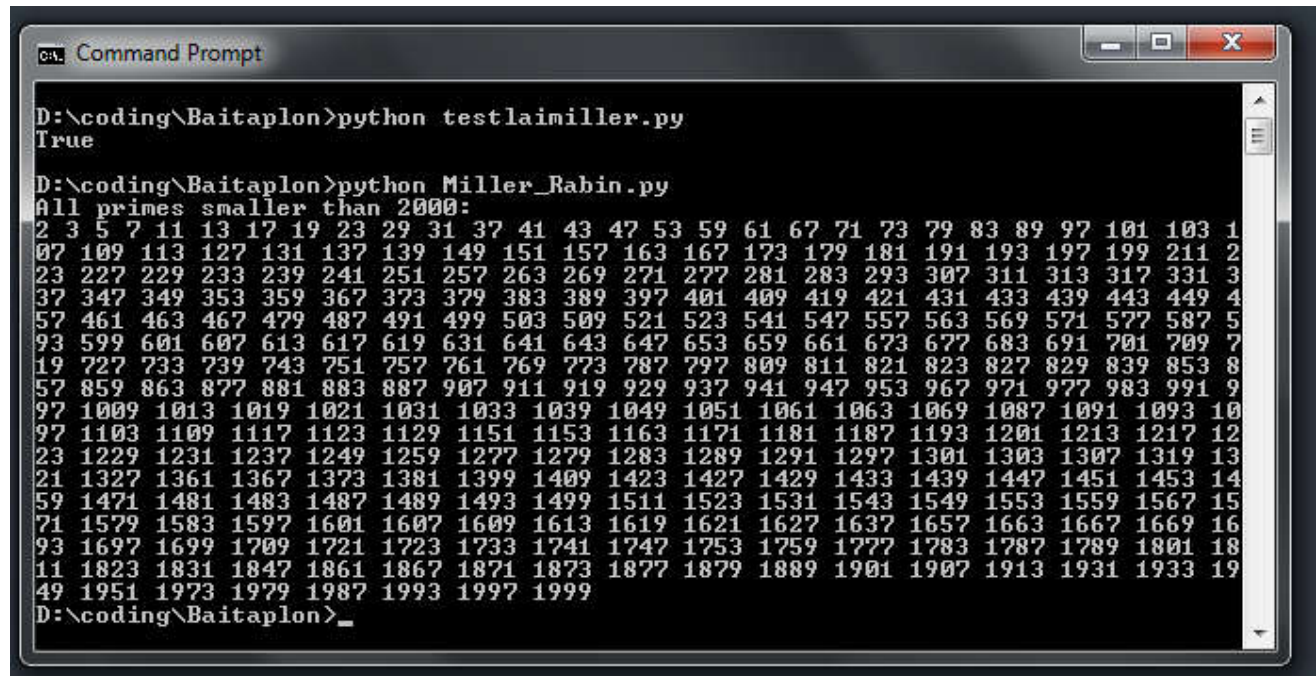
Menu:
Enter your choice of input:
1. Find the next prime number greater than n:
2. Encode and Decode a file using the RSA method:
3. Return all prime numbers using the Miller-Rabin method:
4. RSA Test for built-in Input
5. RSA Test for manually typed in Input:
6. Exit
Enter your choice: 6
Exiting...

```

3. Kiểm thử:

Ở đây do khối lượng tính toán tương đối lớn nên em sử dụng phương pháp Unit Test để kiểm thử 1 số hàm quan trọng:

Kiểm tra tính chính xác của module Miller_Rabin:

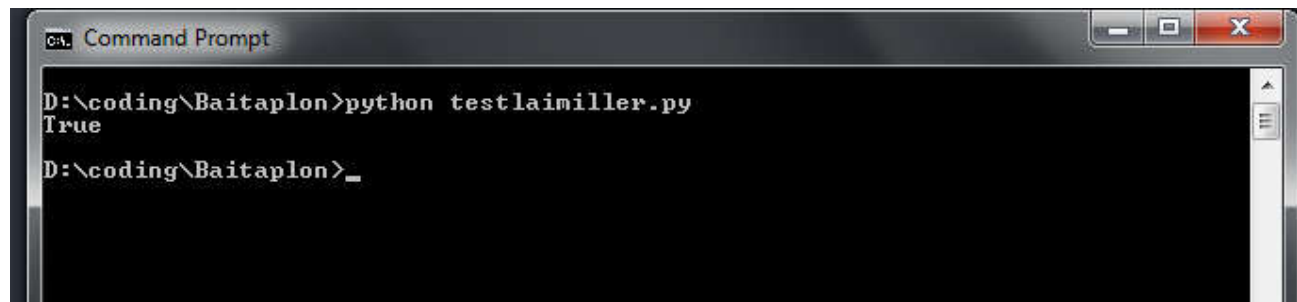


```
D:\coding\Baitaplon>python testlaimiller.py
True

D:\coding\Baitaplon>python Miller_Rabin.py
All primes smaller than 2000:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 1
07 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 2
23 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 3
37 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 4
57 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 5
93 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 7
19 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 8
57 859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 9
97 1009 1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 10
97 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217 12
23 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291 1297 1301 1303 1307 1319 13
21 1327 1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 14
59 1471 1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 15
71 1579 1583 1597 1601 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 16
93 1697 1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 18
11 1823 1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 19
49 1951 1973 1979 1987 1993 1997 1999
D:\coding\Baitaplon>
```

Hàm

millerTest: (Kết quả của hàm millerTest(3,7))



```
D:\coding\Baitaplon>python testlaimiller.py
True

D:\coding\Baitaplon>
```

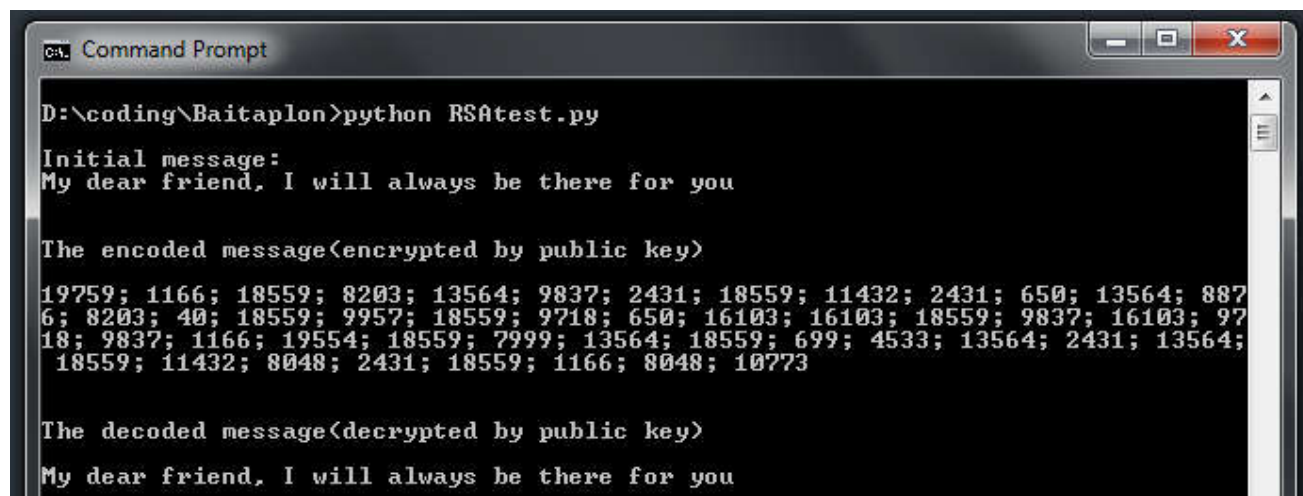
Hàm bubbleSort: kiểm thử với danh sách phần tử [64, 34, 25, 12, 22, 11, 90]



```
D:\coding\Baitaplon>python bubblesort.py

Sorted array:
11 12 22 25 34 64 90
D:\coding\Baitaplon>
```

Test output giải thuật RSA cho đoạn tin nhắn đã nhập sẵn:



```
Command Prompt
D:\coding\Baitaplon>python RSAtest.py
Initial message:
My dear friend, I will always be there for you

The encoded message<encrypted by public key>
19759; 1166; 18559; 8203; 13564; 9837; 2431; 18559; 11432; 2431; 650; 13564; 887
6; 8203; 40; 18559; 9957; 18559; 9718; 650; 16103; 16103; 18559; 9837; 16103; 97
18; 9837; 1166; 19554; 18559; 7999; 13564; 18559; 699; 4533; 13564; 2431; 13564;
18559; 11432; 8048; 2431; 18559; 1166; 8048; 10773

The decoded message<decrypted by public key>
My dear friend, I will always be there for you
```

4. Kết luận:

Chương trình em đã xây dựng trên đây chỉ là một phần rất căn bản của thuật toán RSA & Miller_Rabin. Thuật toán này hoàn toàn có thể nâng cấp lên khi dùng những thư viện mã hóa mạnh hơn của Python, hoặc đơn giản hơn là sử dụng những cải tiến khác mà em có đề cập trong mã nguồn cũng như trong bài làm lần này. Em rất mong nhận được sự đánh giá & góp ý cải thiện của thầy. Em cũng xin phép cảm ơn thầy đã hướng dẫn chúng em trong lần làm bài tập lớn này.