



Report

BigInt và các hàm MulMod, PowMod, GCD, RSA và primality test

Lớp: Nhập môn mã hóa - mật mã

Giảng viên: Nguyễn Đình Thúc

Thành viên nhóm:

Nguyễn Trung Thành: 18120565

Phạm Xuân Thành: 18120567

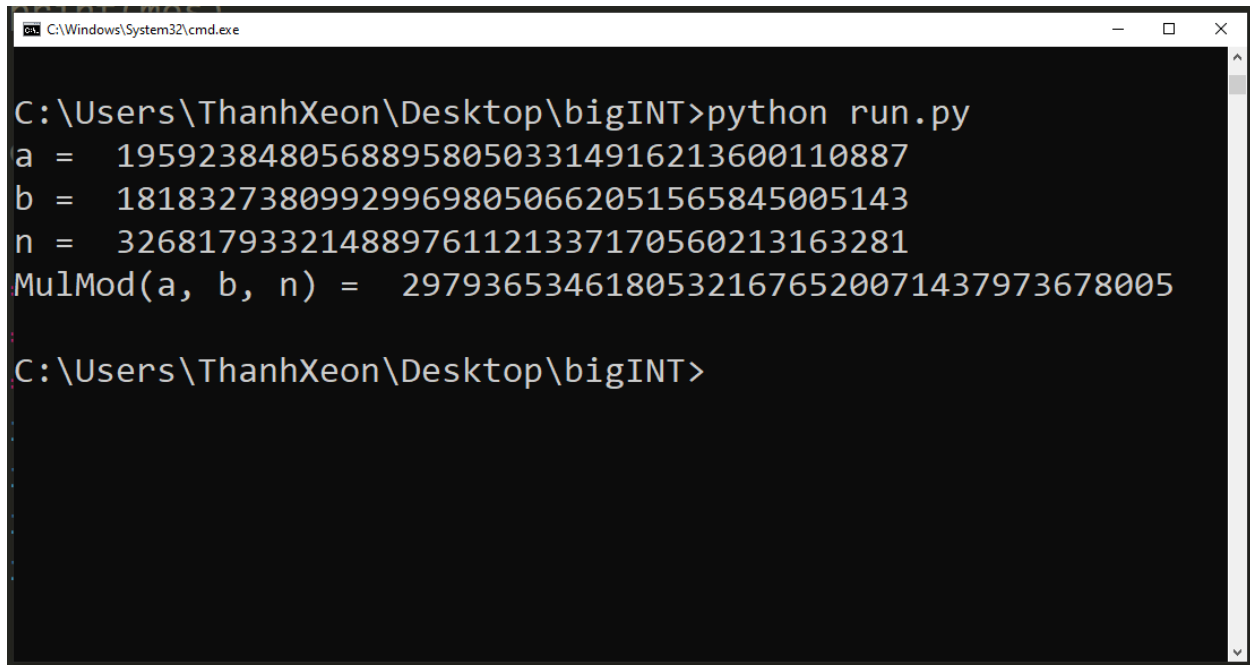
1. *Class::BigInt*

- *Ý tưởng*: sử dụng 16 biến longlong để lưu trữ tối đa 1024bit số nguyên dương.
- *Ngôn ngữ*: Sử dụng ngôn ngữ python trên toàn bộ đồ án.

a) *BigInt::MulMod(self, other)*

$$a * b = \sum_{i=0}^n (a * b[i] * 2^i) \% n$$
, với n là số lượng bit của b và b[i] là bit tại vị trí i.

Tốc độ chạy khá nhanh, độ phức tạp thuật toán $O(\log_2 n)$



```
C:\Windows\System32\cmd.exe

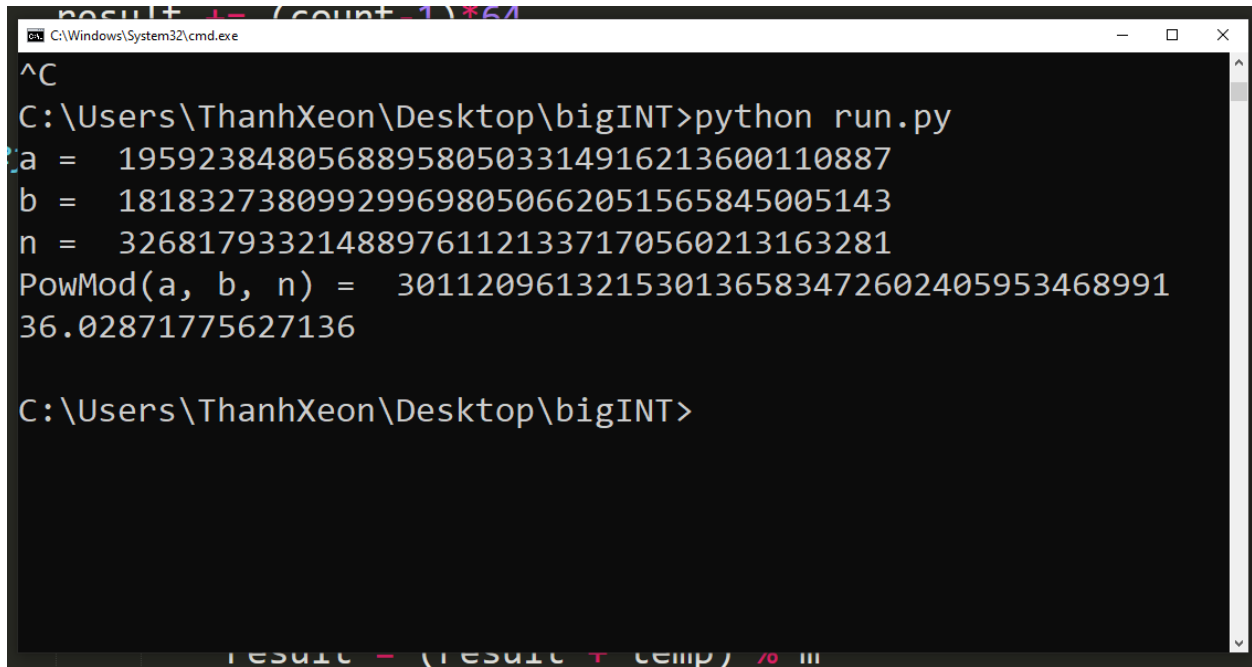
C:\Users\ThanhXeon\Desktop\bigINT>python run.py
a = 195923848056889580503314916213600110887
b = 181832738099299698050662051565845005143
n = 326817933214889761121337170560213163281
MulMod(a, b, n) = 29793653461805321676520071437973678005

C:\Users\ThanhXeon\Desktop\bigINT>
```

b) BigInt::PowMod(self, other)

$res = a \mid a^b = \prod_{i=1}^n (res * b[i] * a^{2^i}) \% n$, với $b[i] = 1$

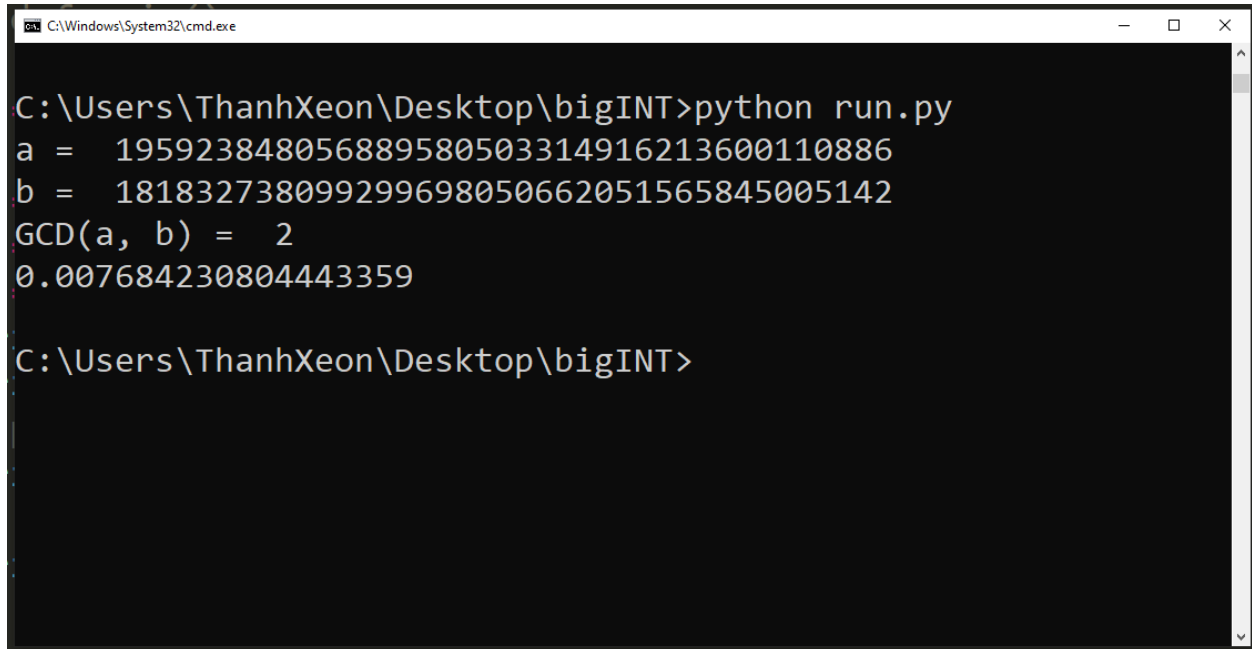
Tốc độ chạy vẫn chưa thực sự tốt, độ phức tạp $O(\log_2^2 n)$



```
result = (count - 1) * 64
C:\Windows\System32\cmd.exe
^C
C:\Users\ThanhXeon\Desktop\bigINT>python run.py
a = 195923848056889580503314916213600110887
b = 181832738099299698050662051565845005143
n = 326817933214889761121337170560213163281
PowMod(a, b, n) = 301120961321530136583472602405953468991
36.02871775627136
C:\Users\ThanhXeon\Desktop\bigINT>
```

c)GCD(a, b, other)

Áp dụng thuật giải thầy đưa

A screenshot of a Windows command prompt window. The title bar at the top reads "C:\Windows\System32\cmd.exe". The command prompt shows the following text:
C:\Users\ThanhXeon\Desktop\bigINT>python run.py
a = 195923848056889580503314916213600110886
b = 181832738099299698050662051565845005142
GCD(a, b) = 2
0.007684230804443359

C:\Users\ThanhXeon\Desktop\bigINT>
The window has a black background with white text. The standard Windows window controls (minimize, maximize, close) are visible in the top right corner of the title bar.

2. RSA

a) *Genkey(p, q)*

Tạo ra e, d, n . Với e là publish key d là private key

b) *Encrypt(pubKey, plaintext)*

Mã hóa *message*: $c = m^e \bmod n$

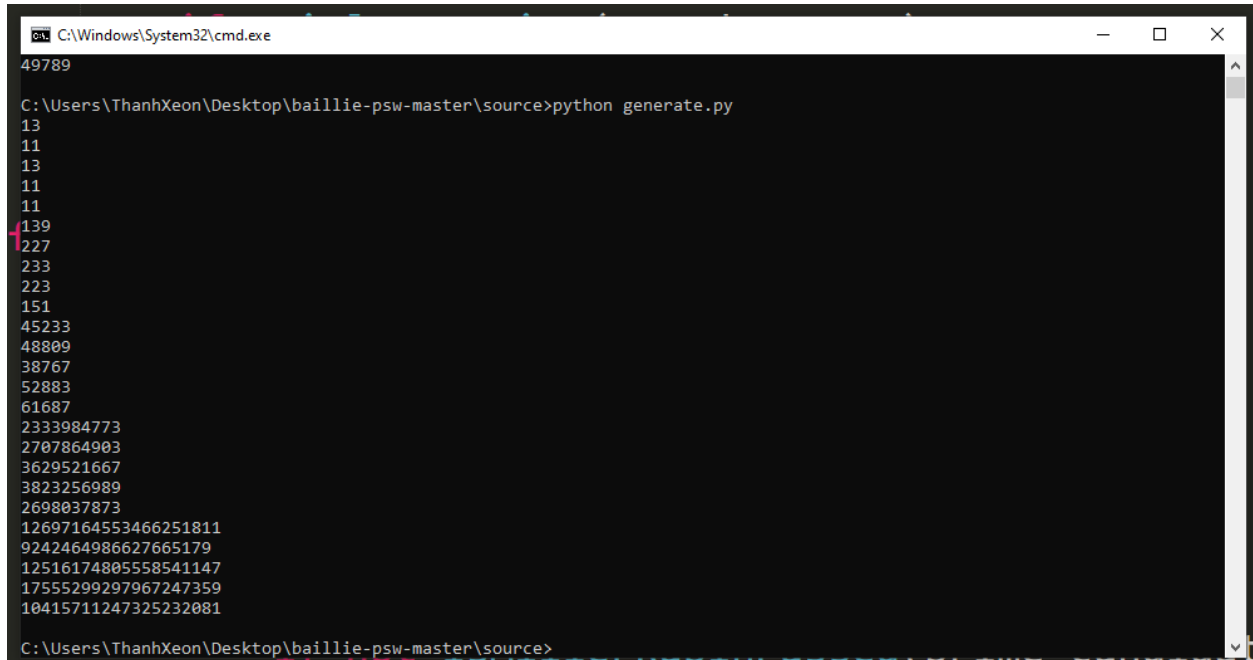
c) *FastDecrypt(prvKey, p, q, ciphers)*

Giải mã dựa trên đồng dư trung hoa và định lý fermat nhỏ

```
C:\Windows\System32\cmd.exe
C:\Users\ThanhXeon\Desktop\bigINT>
C:\Users\ThanhXeon\Desktop\bigINT>python run.py
57786 60520
public Key 7 3497327027
Private Key 1998404983 3497327027
'Day chua ma hoa: chay ngay di!
da ma hoa ['%T\x8fiK', 'sc±`K', '\x88_yhL', ',\x84GyL', 'v²M
\xa0K', 'IWCvC', '\x9e~axB', '\x88_yhL', ',\x84GyL', 'v²M\xa
t0K', '~½\x94{D', '-\x87\x9dφH', '%\x81YwC']
3.436830997467041
chay ngay di!
C:\Users\ThanhXeon\Desktop\bigINT>
iC:\Users\ThanhXeon\Desktop\bigINT>
```

3. *Generate prime numbers*

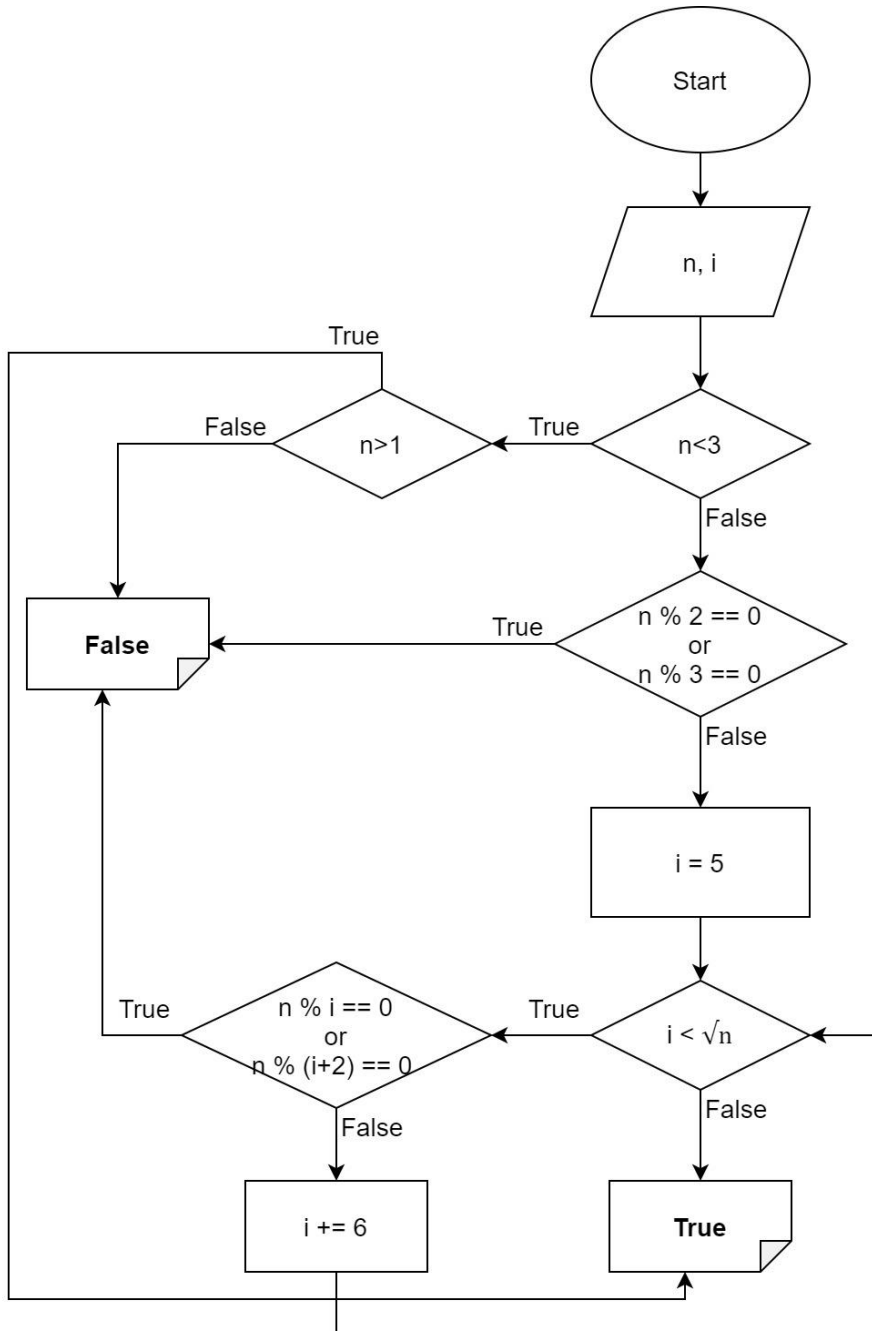
Tạo ra n số nguyên tố dựa trên miler rabin test, trong khoảng k bit cho trước. *Thuật toán có độ tin tưởng cao nhưng vẫn có xác suất không chính xác nên cần kiểm lại với baillie test.*



```
C:\Windows\System32\cmd.exe
49789
C:\Users\ThanhXeon\Desktop\baillie-psw-master\source>python generate.py
13
11
13
11
11
11
139
227
233
223
151
45233
48809
38767
52883
61687
2333984773
2707864903
3629521667
3823256989
2698037873
12697164553466251811
9242464986627665179
12516174805558541147
17555299297967247359
10415711247325232081
C:\Users\ThanhXeon\Desktop\baillie-psw-master\source>
```

4. Thuật toán *Simple primality test*

- *Ý tưởng*: Dựa trên định nghĩa của số nguyên tố, và lược bỏ đi một số bước.



- **Độ phức tạp thuật toán:** $O(\sqrt{n})$
- **Tính chất:** vết cạn, tuyến tính tăng khi n càng lớn
- **Ưu điểm:** Chính xác 100%
- **Khuyết điểm:**
 - + Thực thi quá chậm khi kiểm tra số nguyên lớn.
 - + Tính chất vết cạn nên rất tốn thời gian và tài nguyên.
- **Nhận xét:**
 - + Phù hợp với các số nguyên tố nhỏ.
 - + Độ chính xác tuyệt đối, dễ cài đặt.

5. Thuật toán Miller-Rabin

- **Ý tưởng:**

- + Dựa trên định lý Fermat nhỏ.

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

- + Độ chính xác tăng dần theo số lần lặp lại (k).

- **Độ phức tạp thuật toán:** $O(k \times \log^3 n)$

- **Tính chất:**

- + Mang tính chất xác suất dựa vào các định lý toán học (Fermat)

- + Tồn tại các trường hợp Hợp số có thể vượt qua test (pseudoprime). Có thể giảm thiểu trường hợp này bằng cách lặp lại nhiều lần với số k lớn.

- + Chọn ngẫu nhiên số a trong không gian mẫu thích hợp để áp dụng định lý Fermat.

- **Ưu điểm:**

- + Kiểm tra nhanh hơn *Simple primality test* ở những số nguyên lớn.

- + Có thể thay đổi độ tin cậy dựa vào số lần thử.

- + Nhanh chóng và hiệu quả.

- + Chắc chắn xác định được hợp số

- + Độ chính xác tin cậy dựa trên số lần lặp với công thức: $(\frac{1}{4})^k$

- **Khuyết điểm:**

- + Tồn tại trường hợp thỏa nhưng không phải là số nguyên tố. Bởi vì tính chất ngẫu nhiên trong quá trình chọn cơ sở cho định lý Fermat

- + Khó cài đặt .

- + Ví dụ cho trường hợp tồn tại số giả nguyên tố vượt qua thuật toán Miller-Rabin nhưng không thỏa thuật toán Baillie-PSW.

16581017526609192179

273113379123154359827225664964346213231

284147076998903336725870303827362431751

```
Select C:\Windows\System32\cmd.exe
C:\Users\ThanhXeon\Desktop\baillie-psw-master\source>C:\Users\ThanhXeon\AppData\Local\Microsoft\WindowsApps\python.exe p
rimality_test.py
=== PRIMALITY TEST ===
1. Simple
2. Miller Rabin
3. Baillie PSW
4. Simple with file test
5. Miller Rabin with file test
6. Baillie PSW with file test
Your choice:2
Input candidate:284147076998903336725870303827362431751
Input k:50
It's a prime number.
Reliability: 100.0 %
Count compare: 12855
Count assign: 15802

C:\Users\ThanhXeon\Desktop\baillie-psw-master\source>C:\Users\ThanhXeon\AppData\Local\Microsoft\WindowsApps\python.exe p
rimality_test.py
=== PRIMALITY TEST ===
1. Simple
2. Miller Rabin
3. Baillie PSW
4. Simple with file test
5. Miller Rabin with file test
6. Baillie PSW with file test
Your choice:3
Input candidate:284147076998903336725870303827362431751
It's not a prime number.
Count compare: 545
Count assign: 829

C:\Users\ThanhXeon\Desktop\baillie-psw-master\source>
```

- **Nhận xét:** Độ phức tạp thấp hơn Simple primality test . Không nên sử dụng cho các số nguyên nhỏ, nên cho các số nguyên tố lớn.

6. Thuật toán Baillie-PSW

- Ý tưởng:

+ Dựa trên định lý Fermat nhỏ.

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

+ Loại bỏ các hợp trong khi sử dụng các thuật toán Kiểm tra số nguyên tố mạnh trên một cơ sở nhất định.

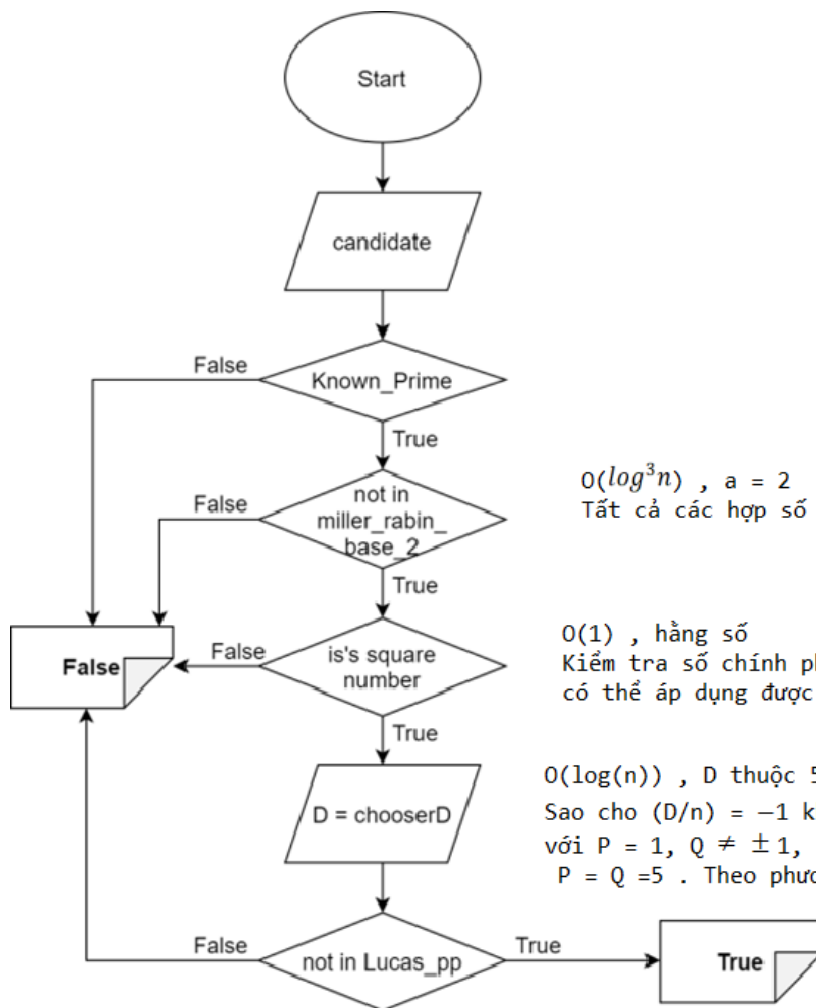
+ Kiểm tra thêm một lần nữa bằng thuật toán Strong Lucas test. Sử dụng Lucas primality test để kiểm tra tính nguyên tố.

- Cài đặt :

➤ Bước 1: Kiểm tra số n với một thuật toán kiểm tra số nguyên tố mạnh trên cơ sở a . Trả về False nếu không thỏa. True thì tiếp tục.

➤ Bước 2: xây dựng các biến số (D,P,Q) cơ bản của thuật toán Lucas test.

➤ Bước 3: Áp dụng thuật toán Lucas test (lpsp). Trả về “Hợp số” nếu False. Trả về “có thể là số nguyên tố” nếu True.



$O(\log^3 n)$, $a = 2$

Tất cả các hợp số sẽ dừng lại ở trường hợp này.

$O(1)$, hằng số

Kiểm tra số chính phương, Mục đích để có thể áp dụng được ký hiệu Jacobi.

$O(\log(n))$, D thuộc $5, -7, 9, -11, \dots$

Sao cho $(D/n) = -1$ khi đó: $D = P^2 - 4Q$ với $P = 1$, $Q \neq \pm 1$, Nếu $Q = \pm 1$ thì $P = Q = 5$. Theo phương pháp A*.

$O(k \times \log^3 n)$

Áp dụng thuật giải Strong Lucas pseudoprime test từ D, P, Q từ bước trước

- **Độ phức tạp thuật toán:** $O(k \times \log^3 n)$

- **Tính chất:**
 - + Mang tính chất xác suất dựa vào các định lý toán học (Fermat)
 - + Tồn tại các trường hợp Hợp số có thể vượt qua test (pseudoprime). Có thể giảm thiểu trường hợp này bằng cách lặp lại nhiều lần với số k lớn.
 - + Sử dụng Lucas primality test để kiểm tra tính nguyên tố.

- **Ưu điểm:**
 - + Độ chính xác tuyệt đối với các số $n < 2^{64}$.
 - + Với các số $n > 2^{64}$, chưa phát hiện tồn tại số nào vượt qua Baillie- PSW.
 - + Độ tin cậy cao.

- **Khuyết điểm:**
 - + Phức tạp, khó cài đặt.

- **Nhận xét:**
 - + Độ phức tạp thấp hơn Simple primality test . Không nên sử dụng cho các số nguyên nhỏ, nên cho các số nguyên tố lớn.

7. Đánh giá mức độ hoàn thành:

STT	Họ và tên	Công Việc	Đánh giá
1	Nguyễn Trung Thành	Thuật toán Baillie-PSW, Simple test, Generate prime numbers, BigInt, RSA.	100%
2	Phạm Xuân Thành	Thuật toán Miler-Rabin, BigInt, RSA, Genkey, Encrypt, FastDecrypt.	100%

8. Tài liệu tham khảo:

Mã hóa thông tin – Tập 1

Mã hóa thông tin – Tập 2

[https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))

<https://arxiv.org/pdf/2006.14425.pdf>

https://www.researchgate.net/publication/317401953_Design_and_Implementation_Fast_Algorithm_of_RSA_Decryption_using_java

https://www.youtube.com/watch?v=NcPdiPrY_g8&ab_channel=JeffSuzuki

https://en.wikipedia.org/wiki/Lucas_primality_test

<http://mpqs.free.fr/LucasPseudoprimes.pdf>

https://en.wikipedia.org/wiki/Baillie%E2%80%93PSW_primality_test

<http://ntheory.org/data/spsps.txt?fbclid=IwAR3C1pK30RhR-94GJajTMPUrpsv6of7FINAsZRqJe2ChUHRAZTyTOWcnpkU>

<https://deepai.org/publication/taxonomy-and-practical-evaluation-of-primality-testing-algorithms?fbclid=IwAR2mxmwjYKBMsUQEMsKjmGBPYBhH7zMRJnM9TKt6dCPSv0X6in-hWXf3NVk>

https://en.wikipedia.org/wiki/Lucas_pseudoprime#Baillie-Wagstaff-Lucas_pseudoprimes