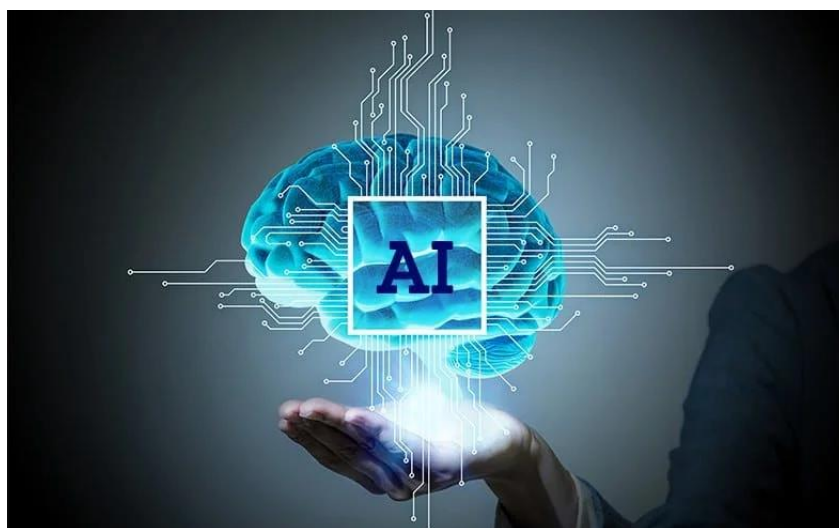


**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
KHOA TOÁN - TIN HỌC**

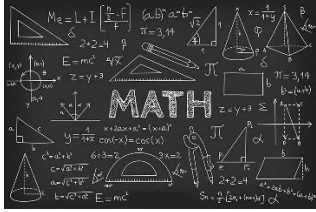


BÀI BÁO CÁO THỰC HÀNH TUẦN 1



MÔN HỌC: Phân Tích Thuật Toán
Sinh Viên: Trần Công Hiếu - 21110294
Lớp: 21TTH

TP.HCM, ngày 07 tháng 04 năm 2024



TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA TOÁN – TIN HỌC



BÀI BÁO CÁO THỰC HÀNH TUẦN 1

HK1 - NĂM HỌC: 2024-2025

MÔN: PHÂN TÍCH THUẬT TOÁN

SINH VIÊN: TRẦN CÔNG HIẾU

MSSV: 21110294

LỚP: 21TTH

[illegible]

Giảng viên Bộ môn

Bài 1.

1. Viết chương trình để biểu diễn một số thập phân N sang dạng biểu diễn nhị phân có độ phức tạp thuật toán là $O(\log_2 N)$.

- Input: Số thập phân N .
- Output: Dạng biểu diễn nhị phân của N .

Bài làm

```
1 N = int(input("Nhập N: "))
2 sum = 0
3 count = -1
4 temp = N
5
6 while(N!=0):
7     count = count + 1
8     sum = sum + (N%2)*pow(10, count)
9     N = N//2
10 print("Số", temp, "chuyển về nhị phân là:", sum)
```

Ý tưởng: Để chuyển từ hệ cơ số 10 sang hệ cơ số 2, ta thực hiện lần lượt các bước sau:

- Bước 1: Ta lấy số N chia cho 2 và lưu lại phần dư, ta lấy thương chia tiếp cho 2 cho đến khi thương bằng 0.
- Bước 2: Kết quả phép chuyển, ta lấy tất cả các số dư và ghi từ dưới lên.

Chi tiết:

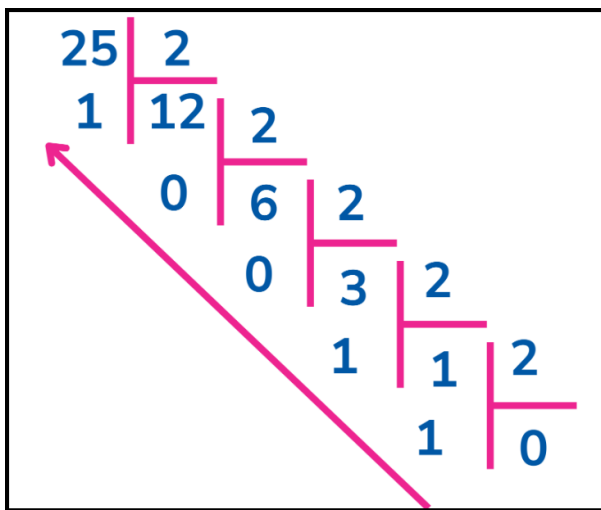
```
1 N = int(input("Nhập N: "))
2 sum = 0
3 count = -1
4
```

“ $N = \text{int}(\text{input}(\text{"Nhập } N: \text{"}))$ ”: Đầu tiên ta gọi hàm $\text{input}()$ để thông báo nhập giá trị cho N . Và dùng hàm $\text{int}()$ để chuyển đổi chuỗi nhập vào thành một số nguyên. Bởi trong vòng lặp while phía dưới sẽ có thao tác chia lấy nguyên, nên nếu chỉ gọi hàm $\text{input}()$ cho N thì sẽ dẫn đến lỗi định dạng kiểu dữ liệu.

“ $\text{sum} = 0$ ”: Khởi gán biến sum bằng 0 để lưu lại kết quả của N sau khi được chuyển đổi về hệ nhị phân.

“ $\text{count} = -1$ ”: Khởi gán biến count bằng -1, biến này có tác dụng đếm số số dư thu được sau mỗi lần N chia 2 lấy nguyên (trong ví dụ dưới thì số số dư thu được là 5 lần lượt là 1 1 0 0 1). Nhằm biểu diễn hệ số mũ của lũy thừa 10 giúp cho việc lấy kết quả ghi được từ dưới lên. Và vì ngay sau khi thỏa điều kiện của vòng lặp while là count tăng thêm 1 đơn vị nên khởi gán ban đầu là -1 để khi cộng 1 sẽ là 0 và khi mũ 0 lên sẽ đẩy số dư đầu tiên (trong ví dụ là 1, số dư của phép chia 25 cho 2) về vị trí đơn vị trong kết quả biểu diễn dạng nhị phân của N .

ví dụ:



```

5   while(N!=0):
6       count += 1
7       sum = sum + (N%2)*pow(10, count)
8       N = N//2
9   print(sum)
10

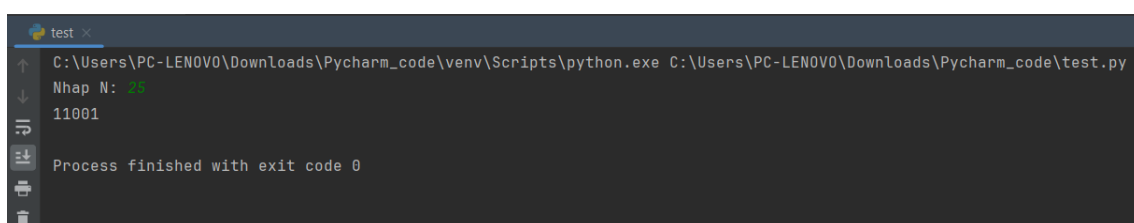
```

“while(N!= 0):”: Tạo vòng lặp while với điều kiện tiếp tục là khi N khác 0, nói cách khác thì đúng với ý tưởng nêu trên, sau khi N chia 2 lấy nguyên nhiều lần thì sẽ dừng đến khi nào N bằng 0. Trong vòng lặp while ta sẽ thực hiện lần lượt các bước sau:

- “count += 1”: Tăng biến đếm count lên 1 đơn vị để cho việc lũy thừa cơ số 10.
- “sum = sum + (N%2)*pow(10, count)”: Cập nhật giá trị biến sum mới bằng chính nó trước đó cộng thêm 1 biểu thức. Biểu thức đó là số dư (1 hoặc 0) nhận được sau mỗi lần lấy N chia 2 rồi nhân cho 10 mũ count, điều này giúp các số dư ấy lần lượt trở nằm ở vị trí đơn vị, chục, trăm, ngàn,... như ta mong muốn chính là đảo ngược của các số dư. Như trong ví dụ số dư lần lượt là 1 0 0 1 1 thì nhờ biểu thức này, sum sẽ thu được kết quả biểu diễn của N dưới dạng nhị phân là 11001.
- “N = N//2”: Cập nhật lại giá trị của N bằng giá trị của N chia lấy nguyên cho 2.

Kết quả:

Với $N = 25$, ta thu được kết quả chuyển đổi qua hệ nhị phân là 11001 như hình bên dưới.



```

test
C:\Users\PC-LENOVO\Downloads\Pycharm_code\venv\Scripts\python.exe C:\Users\PC-LENOVO\Downloads\Pycharm_code\test.py
Nhap N: 25
11001
Process finished with exit code 0

```

Hướng khác tiếp cận bài toán.

Khi nói đến số thập phân, ta nghĩ đến số nguyên (âm và dương), số có phần nguyên và phần thập phân (số thực). Và vì độ phức tạp thuật toán là $O(\log_2 N)$ nên có thể coi N nguyên dương. Xét trường hợp N nguyên nhưng có cả nguyên âm thì ta có thể làm như sau:

• Khi N nguyên âm.

Ý tưởng:

1. Xác định giá trị tuyệt đối của N .
2. Chuyển giá trị N vừa lấy trị tuyệt đối về hệ nhị phân.
3. Thêm các bit 0 vào trước kết quả trên để đủ 8 bit.
4. Đảo ngược bit và thêm 1.

Bài làm.

```
1      N = int(input("Nhập N (N<0) : "))
2      sum = 0
3      count = -1
4
5      N = N*(-1)
6
7      while(N!=0):
8          count = count + 1
9          sum = sum + (N%2)*pow(10, count)
10         N = N//2
11
12     print(sum)
```

Tương tự với trường hợp N dương chỉ khác ở phần đầu chương trình, ta chuyển đổi N nguyên âm về nguyên dương bằng cách nhân vào giá trị -1. Lúc này, sum là kết quả của $|N|$ sau khi chuyển về hệ nhị phân.

```
14     result = '0'*(8-len(str(sum))) + str(sum)
15     #print(type(result))
16
17     print(result)
```

Tiếp đến ta thêm các bit 0 vào trước kết quả trên để đủ 8 bit, muốn làm như thế ta chỉ cần lấy '0' nhân với số bit 0 còn thiếu để đủ 8 bit bằng biểu thức $8 - \text{len}(\text{str}(\text{sum}))$ và cộng cho $\text{str}(\text{sum})$.

```
19     for i in range(len(result)):
20         if result[i] == '1':
21             result = result[:i] + '0' + result[i+1:]
22         else:
23             result = result[:i] + '1' + result[i+1:]
24
25     print(result)
26
```

Ở bước này, ta dùng vòng lặp for để đảo ngược các bit, 0 thành 1 và 1 thành 0.


```

27     new_result = list(result)
28
29     if result[-1] == '1':
30         i = len(result) - 1
31         while i >= 0 and result[i] == '1':
32             new_result[i] = '0'
33             i -= 1
34         if i >= 0:
35             new_result[i] = '1'
36         else:
37             chuoimoi = ['1'] + new_result
38
39     else:
40         new_result[-1] = '1'
41
42     new_result = ''.join(new_result)
43
44     print(new_result)
45

```

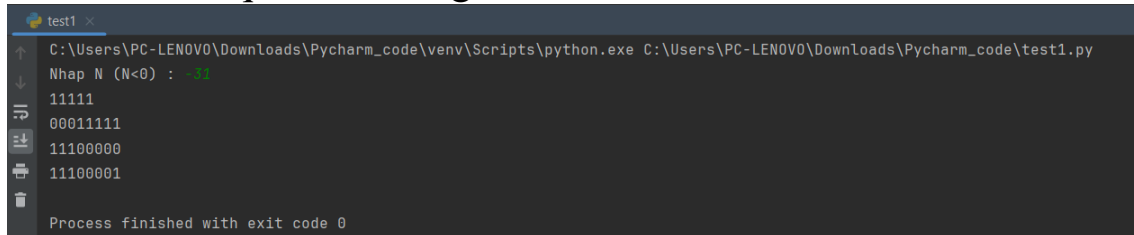
Cuối cùng là thao tác cộng 1 vào bit cuối cùng, nếu bit cuối cùng là 1 thì khi cộng phải nhớ 1. Đoạn mã này kiểm tra nếu bit cuối là 1 thì sẽ thực hiện cộng liên tục thông qua việc duyệt từ phải sang trái kết quả hệ nhị phân, nếu là 1 thì chuyển về 0, cho đến khi gặp 0 thì dừng lại và cập nhật nó thành 1. Ngược lại, nếu bit cuối cùng là 0 thì ta cập nhật lại thành 1.

Vì ban đầu new_result là một list nên tới bước cuối ta dùng phương thức join() để nối các phần tử trong list và ngăn cách nhau bằng một chuỗi rỗng. Mục đích là chuyển từ list về các ký tự thành một chuỗi để in ra kết quả.

Kết quả.

Với $N = -31$, chương trình sẽ lấy trị tuyệt đối của N , tức lúc này $N = 31$, sau đó chuyển về hệ nhị phân ta được 11111 và

thêm các bit 0 ở đầu sao cho đủ 8 bit (00011111). Đảo ngược bit ta thu được 11100000. Cuối cùng là cộng thêm 1 vào bit cuối và thu được kết quả cuối cùng là 11100001 như hình bên dưới.



```
test1 x
C:\Users\PC-LENOVO\Downloads\Pycharm_code\venv\Scripts\python.exe C:\Users\PC-LENOVO\Downloads\Pycharm_code\test1.py
Nhap N (N<0) : 1111
1111
00011111
11100000
11100001
Process finished with exit code 0
```

Độ phức tạp thuật toán.

Nhìn chung chương trình có 3 đoạn chương trình có thời gian thực hiện lần lượt là $O(\log_2(N))$, $O(1)$, $O(1)$. Do vậy, tổng thời gian thực hiện cả chương trình là $O(\log_2(N + 1 + 1))$ và độ phức tạp thuật toán lúc này vẫn là $O(\log_2(N))$.

Nhận xét.

Ta có thể bắt gặp từ khóa “Biểu diễn bù 1” hay “Biểu diễn bù 2” khi làm bài toán biểu diễn số thập phân sang dạng biểu diễn nhị phân. Và ở chương trình trên thì chúng ta đã dùng “Biểu diễn bù 2” bởi sự rõ rệt lớn nhất giữa 2 biểu diễn này là nằm ở số 0. “Biểu diễn bù 2” sẽ xử lý số 0 thỏa tính nhất quán là chỉ có một biểu diễn cho số 0, xử lý dấu chấm động và hơn nữa là giảm thiểu số lượng quy tắc đặc biệt so với “Biểu diễn bù 1”.

• Khi N thực dương.

Nếu như số thập phân không chỉ dừng lại ở số nguyên dương mà ở số thực dương thì liệu ta có thể ép độ phức tạp thuật toán vẫn có thể là $O(\log_2 N)$?

```

1 usage
2 def float_to_binary(num):
3     if num >= 1 or num <= -1:
4         int_part = bin(int(abs(num)))[2:] # Chuyển phần nguyên sang nhị phân
5     else:
6         int_part = '0'
7
8     decimal_part = ''
9     frac = abs(num) - abs(int(num)) # Phần thập phân
10    while frac != 0:
11        frac *= 2
12        if frac >= 1:
13            decimal_part += '1'
14            frac -= 1
15        else:
16            decimal_part += '0'
17
18    return int_part + '.' + decimal_part
19
20 N = float(input("Nhập N thực dương: "))
21 binary_representation = float_to_binary(N)
22 print("Số thực", N, "được biểu diễn nhị phân là:", binary_representation)
23

```

Thuật toán trong chương trình trên chuyển đổi số thực dương về biểu diễn nhị phân bằng cách định nghĩa hàm `float_to_binary()` với đối số đầu vào là `num`, chính là số N cần chuyển về biểu diễn nhị phân.

Chúng hoạt động tuân tự việc tách phần thực thành 2 phần (phần nguyên và phần thập phân).

Với phần nguyên ta chuyển bình thường về dạng nhị phân, có thể dùng hàm đã viết ở trên đối với n nguyên dương hoặc dùng hàm có sẵn trong thư viện python là `bin()`. Độ phức tạp của chúng lúc này là $O(\log_2 n)$

Với phần thập phân, ta dùng vòng lặp `while` để kiểm tra. Thì xét về tổng quan, trong trường hợp này, số lần lặp không phụ thuộc vào giá trị tuyệt đối của phần thập phân m , mà phụ thuộc vào số bit cần để biểu diễn phần thập phân đó. Và trong trường hợp này, giả sử m có độ dài là L bits, thì số lần lặp cần thiết là khoảng $O(L)$. Hơn nữa, Khi phân tích độ phức tạp của một thuật toán, chúng ta thường muốn biểu thị độ phức tạp tăng lên tương

ứng với kích thước của dữ liệu đầu vào. Trong trường hợp này, dữ liệu đầu vào là phần thập phân của số thực, và kích thước của nó có thể được biểu diễn bằng số bit cần thiết để biểu diễn phần thập phân đó. Điều này gợi ý rằng độ phức tạp có thể được biểu diễn bằng một hàm tăng logarit với kích thước của dữ liệu đầu vào.

Khi chúng ta nói " $O(\log_2 m)$ " trong phân tích độ phức tạp, chúng ta đang ám chỉ rằng độ phức tạp tăng tuyến tính với số lượng bit cần thiết để biểu diễn phần thập phân của số thực. Điều này phản ánh rằng số lượng lần lặp trong quá trình chuyển đổi phần thập phân sang nhị phân tăng theo cách tuyến tính với số bit này.

Tóm lại, có thể thấy độ phức tạp của thuật toán trên là $O(\log_2(nm))$, có dạng gần tương tự $O(\log_2 N)$ chứ không giống. Và kết quả thu được là:

```
C:\Users\PC-LENOV0\Downloads\Pycharm_code\venv\Scripts\python.exe C:\Users\PC-LENOV0\Downloads\Pycharm_code\test1.py
Nhập N thực dương: 25.56125
Số thực 25.56125 được biểu diễn nhị phân là: 11001.100011111010111000010100011110101110000101001
Process finished with exit code 0
```

• Khi N thực âm.

Ta có thể dựa trên ý tưởng “Chuyển đổi số thực sang nhị phân theo chuẩn IEEE 754” và đoạn mã trên để tiếp tục thực hiện bước cuối để đưa phần nguyên và phần lẻ đã tính thành dạng IEEE 754. Hoặc viết một chương trình cũng đi theo hướng chuẩn IEEE 754 nhưng quá trình xử lý thao tác với phần thập phân sẽ khác.

2. Giả sử các em đã xây dựng thuật toán trên. Hãy viết một chương trình để đếm số phép gán và số phép so sánh mà chương trình trên đã dùng để biểu diễn một số thập phân N .

- **Input:** $binary(N)$, $N = 100, 200, 300, 400, \dots, 1000$.
- **Output:** $Gan(N)$, $Sosanh(N)$. Vẽ $\log_2 N$, $Gan(N)$ và $Sosanh(N)$ trên cùng một đồ thị để so sánh.

Ý tưởng:

Tương tự như chương trình trên, nhưng ta đưa tất cả bước chuyển đổi sang hệ nhị phân vào hàm. Khởi tạo các biến đếm số lần gán, số lần so sánh. Sau đó hàm trả về giá trị lần lượt là $\log_2 N$, count_gan và count_sosanh.

Bài làm.

```
1  import math
2  import numpy as np
3  import matplotlib.pyplot as plt
   1 usage
4  def N_to_Binary(N):
5      print("- Voi N = ", N, "thì:")
6
7      sum = 0
8      count = -1
9      log_2_N = math.log2(N)
10
```

Đầu tiên ta import các thư viện cần thiết cho chương trình, ở đây có 3 thư viện chính là math để tính $\log_2 N$ thuận tiện cho việc tính để vẽ biểu đồ, numpy as np để tính toán cũng như gọi mảng để lưu các giá trị trả về từ hàm và công cụ vẽ là matplotlib.pyplot viết tắt là plt.

“def N_to_Binary(N):”: Định nghĩa hàm N_to_Binary() để chuyển số N đầu vào thành biểu diễn nhị phân, với đối số đầu vào là N , phía dưới ta sẽ truyền vào các giá trị cho N từ mảng.

“sum = 0”: Khởi gán biến sum bằng 0 để lưu lại kết quả của N sau khi được chuyển đổi về hệ nhị phân.

“count = -1”: Khởi gán biến count bằng -1, biến này có tác dụng đếm số số dư thu được sau mỗi lần N chia 2 lấy nguyên (trong ví dụ dưới thì số số dư thu được là 5 lần lượt là 1 1 0 0 1).

Nhằm biểu diễn hệ số mũ của lũy thừa 10 giúp cho việc lấy kết quả ghi được từ dưới lên. Và vì ngay sau khi thỏa điều kiện của vòng lặp while là count tăng thêm 1 đơn vị nên khởi gán ban đầu là -1 để khi cộng 1 sẽ là 0 và khi mũ 0 lên sẽ đẩy số dư đầu tiên (trong ví dụ là 1, số dư của phép chia 25 cho 2) về vị trí đơn vị trong kết quả biểu diễn dạng nhị phân của N .

“ $\log_2 N = \text{math.log2}(N)$ ”: Sử dụng hàm log trong thư viện math để tính $\log_2 N$.

```
11         # Gan(N), Sosanh(N)
12         count_gan = 3
13         count_sosanh = 1
14
15         while (N != 0):
16             count_sosanh += 1
17             count += 1
18             sum = sum + (N % 2) * pow(10, count)
19             N = N // 2
20             count_gan += 3
21
22         print(sum)
23         print("So phép so sánh: ", count_sosanh)
24         print("so phép gan: ", count_gan, "\n")
25
26         return np.array([log_2_N, count_gan, count_sosanh])
27
```

“ $\text{count_gan} = 3$ ”: Khởi tạo biến đếm số lần gán là count_gan với giá trị ban đầu là 3, vì ở bài toán 1 trên, ta cho count_gan bằng 3 vì tính cả input cho N nhưng ở bài này thì ta truyền vào hàm giá trị, hơn nữa còn có thêm cả “ $\log_2 N = \text{math.log2}(N)$ ” nhưng vì ta tính cho mục đích vẽ nên sẽ không tính vào việc đếm gán. Hoặc tùy quy ước để chỉnh sửa giá trị ban đầu của biến đếm count_gan.

“count_sosanh = 1”: Khởi tạo biến count_sosanh để đếm số lần so sánh và cho giá trị ban đầu là 1. Bởi vì nếu $N = 0$ thì nó vẫn so sánh 1 lần với điều kiện vòng lặp while, còn nếu N ban đầu là giá trị khác 0 thì khi nó chia đến khi bằng 0 thì nó cũng so sánh với điều kiện while nhưng vì bằng 0 nên nếu không đếm vì nó không đi vào để đếm lần so sánh đó thì sẽ bị sót. Do đó, ban đầu count_sosanh sẽ là 1 thay vì 0 như mọi biến đếm hay thấy.

“while($N \neq 0$):”: Tạo vòng lặp while với điều kiện tiếp tục là khi N khác 0, nói cách khác thì đúng với ý tưởng nêu trên, sau khi N chia 2 lấy nguyên nhiều lần thì sẽ dừng đến khi nào N bằng 0. Trong vòng lặp while ta sẽ thực hiện lần lượt các bước sau:

- “count_sosanh += 1”: Cộng 1 lần so sánh và lưu vào biến count_sosanh.
- “count += 1”: Tăng biến đếm count lên 1 đơn vị để cho việc lũy thừa cơ số 10.
- “sum = sum + ($N \% 2$)*pow(10, count)”: Cập nhật giá trị biến sum mới bằng chính nó trước đó cộng thêm 1 biểu thức. Biểu thức đó là số dư (1 hoặc 0) nhận được sau mỗi lần lấy N chia 2 rồi nhân cho 10 mũ count, điều này giúp các số dư ấy lần lượt trở nằm ở vị trí đơn vị, chục, trăm, ngàn,... như ta mong muốn chính là đảo ngược của các số dư. Như trong ví dụ số dư lần lượt là 1 0 0 1 1 thì nhờ biểu thức này, sum sẽ thu được kết quả biểu diễn của N dưới dạng nhị phân là 11001.
- “ $N = N // 2$ ”: Cập nhật lại giá trị của N bằng giá trị của N chia lấy nguyên cho 2.
- “count_gan += 3”: Cộng thêm 3 vào biến đếm số lần gán của biến sum, N và count.

Và lần lượt in các kết quả lần lượt là hệ biểu diễn nhị phân của N , số phép so sánh và số phép gán ứng với từng N truyền vào. Cuối cùng thì hàm trả về mảng 3 giá trị thuận tiện cho việc vẽ đồ thị.

```

28     N = np.array([100, 200, 300, 400, 500, 600, 700, 800, 900, 1000])
29     y = []
30
31     for i in N:
32         y.append(N_to_Binary(i))
33
34     y_new = np.array(y)

```

“N = np.array([...])”: Khởi tạo mảng N với các giá trị lần lượt là 100, 200, ..., 1000.

“y = []”: Khởi tạo list y để lưu các giá trị trả về từ hàm, dùng để lấy các giá trị đó vẽ.

“for i in N:”: Dùng vòng lặp for duyệt qua từng phần tử trong N.

“y.append(N_to_Binary(i))”: Gọi phương thức append() để thêm vào list giá trị trả về của hàm ứng với i lần lượt là từng phần tử trong i.

Cuối cùng, ép y từ list về mảng thông qua phương thức array từ thư viện numpy, gọi tắt là np lúc import. Và gán vào y_new, lúc này y_new sẽ là 1 mảng chứa các giá trị trả về tương ứng với từng giá trị N.

```

36     plt.plot(*args: N, y_new[:,0], label='log_2_N')
37     plt.plot(*args: N, y_new[:,1], label='count_gan')
38     plt.plot(*args: N, y_new[:,2], label='count_sosanh')
39
40     plt.title('Biểu đồ ba đường')
41     plt.xlabel('Trục X')
42     plt.ylabel('Trục Y')
43     plt.legend()
44
45     plt.show()
46

```

“plt.plot(N, y_new[:,0], label='log_2_N')”: Đây là lệnh để vẽ đường cho dữ liệu được lưu trong mảng y_new theo trục X là

N, và trục Y là cột đầu tiên của mảng y_new. Label của đường này được đặt là 'log_2_N'.

“plt.plot(N, y_new[:,1], label='count_gan')”: Tương tự như trên, lệnh này vẽ đường cho dữ liệu trong cột thứ hai của y_new với trục X là N. Label của đường này là 'count_gan'.

“plt.plot(N, y_new[:,2], label='count_sosanh')”: Lệnh này vẽ đường cho dữ liệu trong cột thứ ba của y_new với trục X là N. Label của đường này là 'count_sosanh'.

“plt.title('Biểu đồ ba đường')”: Thiết lập tiêu đề cho biểu đồ là 'Biểu đồ ba đường'.

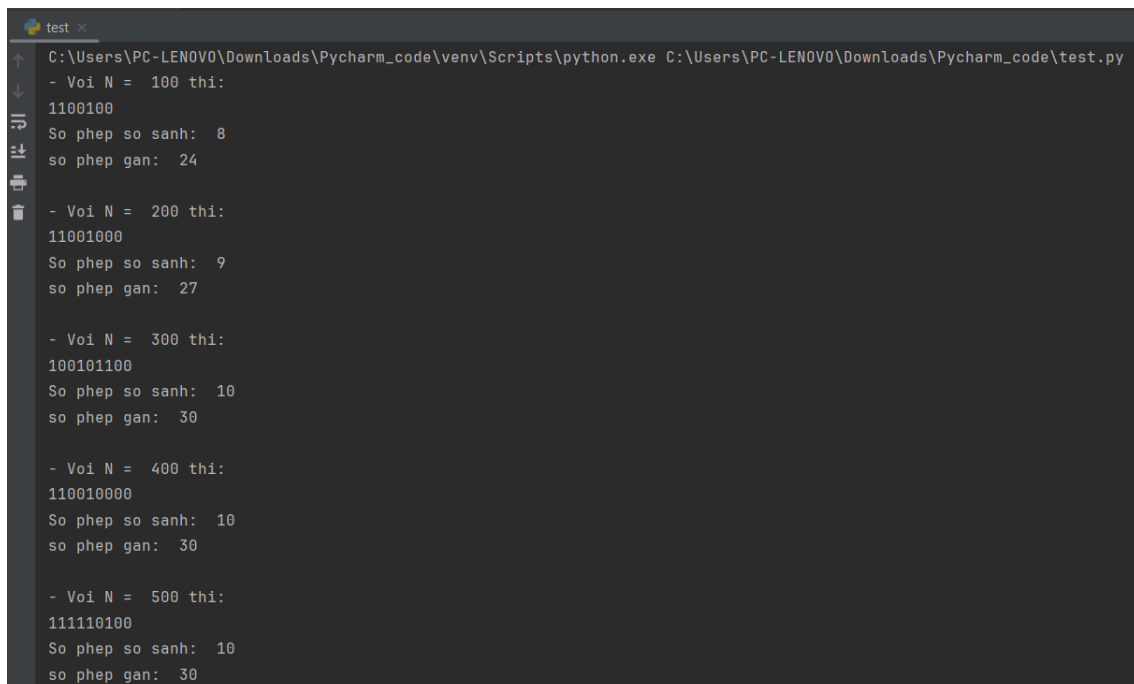
“plt.xlabel('Trục X')”: Đặt nhãn cho trục X là 'Trục X'.

“plt.ylabel('Trục Y')”: Đặt nhãn cho trục Y là 'Trục Y'.

“plt.legend()”: Hiện thị chú thích (legend) trên biểu đồ, cho phép người đọc hiểu được ý nghĩa của từng đường.

“plt.show()”: Lệnh này hiện thị biểu đồ đã được vẽ.

Kết quả.



```
test x
C:\Users\PC-LENOVO\Downloads\Pycharm_code\venv\Scripts\python.exe C:\Users\PC-LENOVO\Downloads\Pycharm_code\test.py
- Voi N = 100 thi:
1100100
So phiep so sanh: 8
so phiep gan: 24

- Voi N = 200 thi:
11001000
So phiep so sanh: 9
so phiep gan: 27

- Voi N = 300 thi:
100101100
So phiep so sanh: 10
so phiep gan: 30

- Voi N = 400 thi:
110010000
So phiep so sanh: 10
so phiep gan: 30

- Voi N = 500 thi:
111110100
So phiep so sanh: 10
so phiep gan: 30
```

```
test ×
↑
↓
so phep gan: 30
- Voi N = 600 thi:
1001011000
So phep so sanh: 11
so phep gan: 33

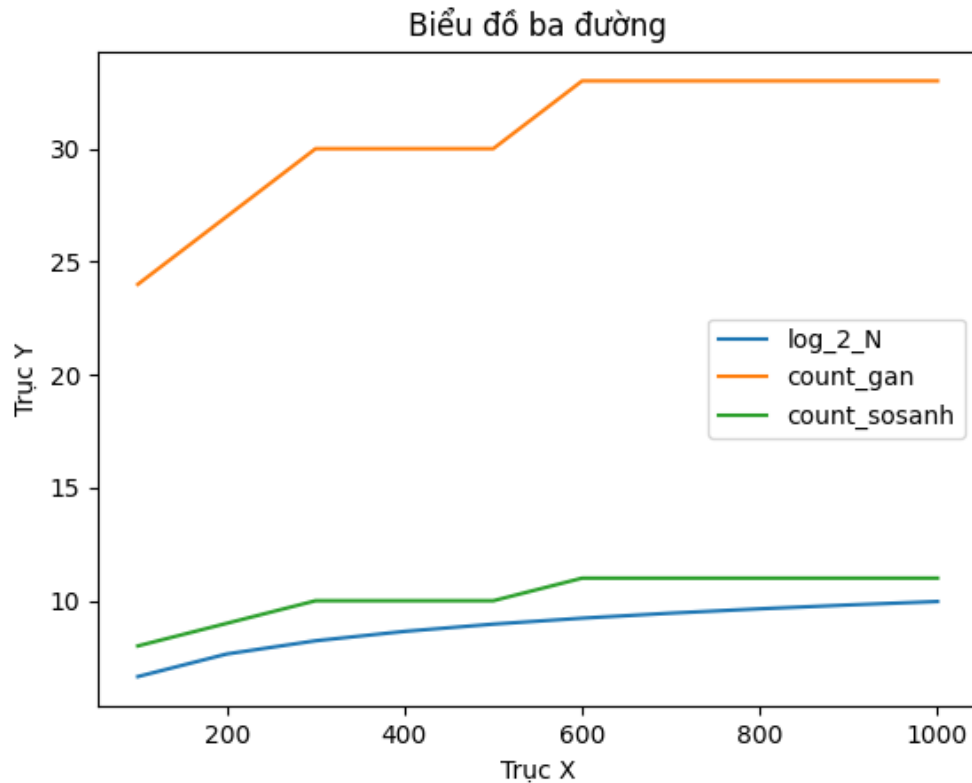
- Voi N = 700 thi:
1010111100
So phep so sanh: 11
so phep gan: 33

- Voi N = 800 thi:
1100100000
So phep so sanh: 11
so phep gan: 33

- Voi N = 900 thi:
1110000100
So phep so sanh: 11
so phep gan: 33

- Voi N = 1000 thi:
1111101000
So phep so sanh: 11
so phep gan: 33

Process finished with exit code 0
```



Từ đồ thị ta có thể rút ra một vài nhận xét. Tương quan giữa $\log_2 N$ và count_sosanh cho thấy sự gần gũi giữa hai đường này có mối tương quan giữa kích thước của dữ liệu (biểu diễn qua $\log_2 N$) và số lần so sánh (biểu diễn qua count_sosanh). Nếu $\log_2 N$ tăng lên, có thể số lần so sánh cũng tăng theo một cách tương tự hoặc có mối tương quan nào đó. Sự khác biệt giữa count_gan và count_sosanh thì lại cho ta thấy count_gan là một bội số của count_sosanh, điều này có thể chỉ ra rằng việc gán đang thực hiện một số lượng lớn các hoạt động so sánh, so với một thuật toán so sánh tiêu chuẩn. Điều này là do cách mà thuật toán gán được thiết kế một số lượng lớn các so sánh phức tạp.

Về hiệu suất và phức tạp thuật toán thì sự khác biệt trong số lần so sánh giữa các thuật toán (count_gan và count_sosanh) có thể gợi ý về hiệu suất và phức tạp của chúng. Thuật toán count_gan yêu cầu nhiều tài nguyên hơn hoặc có phức tạp hơn so với thuật toán count_sosanh, dựa trên số lượng so sánh được thực hiện.

Vì thế nên cân nhắc và tối ưu hóa khi thấy sự khác biệt giữa các đường biểu diễn, có thể cân nhắc tối ưu hóa hoặc điều chỉnh thuật toán để cải thiện hiệu suất hoặc giảm độ phức tạp của nó, đồng thời đảm bảo rằng kết quả đạt được vẫn đúng đắn và chính xác.

Bài 2. (Bonus): Cho một mảng A gồm N số tự nhiên nằm $[1 \dots k]$. Hãy thiết kế thuật toán để kiểm tra xem có bao nhiêu phần tử của A nằm trong $[a, b]$ có độ phức tạp là $O(N + k)$ và chứng minh thuật toán của các em đưa ra có độ phức tạp như trên.

- Input: Xét hai trường hợp
 - $N = 10, 20, 30, \dots, 10000$ (k cố định, $k = 100$).
 - $k = 10, 20, 30, \dots$ (N cố định, $N = 20000$).Tạo ngẫu nhiên mảng A với $[a, b]$ tương ứng.
- Output: Đếm số phép gán, số phép so sánh và so sánh với $O(N + k)$.

Bài làm.

```
1  import random
2  import numpy as np
   2 usages
3  def functionNk(N, k):
4      arr=[]
5      for i in range(N):
6          arr.append(random.randint(a=1,k))
7          # print(np.array(arr))
8      return np.array(arr)
9
```

“import random”: Gọi thư viện random để tạo ngẫu nhiên N phần tử trong mảng A thuộc trong đoạn $[1 \dots k]$.

“import numpy as np”: Gọi thư viện numpy cho việc thao tác với mảng và gọi tắt là np.

“def functionNk(N, k):” Hàm tên functionNk với 2 đối số đầu vào là N và k để tạo mảng A có N phần tử ngẫu nhiên nằm trong đoạn từ [1...k]. Trong hàm ta tạo một list arr và lặp qua từng phần tử và append giá trị random nằm trong đoạn [1...k]. Hàm trả về một mảng bằng cách ép list thành mảng.

```
10 def cmpfuncN(a,b,k):
11     for i in range(10,10010,10):
12         print("- Khi N =",i)
13         arr = functionNk(i, k)
14         print("+ So sanh ", i)
15         print("+ Gan " len(arr[(arr>=a)&(arr<=b)]))
16
17
```

“def cmpfuncN(a, b, k):” Hàm tên cmpfuncN tức là compare function N (được hiểu là so sánh khi N thay đổi) với 3 đối số đầu vào là a, b (đoạn [a,b] để kiểm tra xem có bao nhiêu phần tử nằm trong đoạn này) và k. Hàm này mục đích kiểm tra khi N thay đổi (k cố định) thì có bao nhiêu phần tử được tạo ra ngẫu nhiên đó nằm trong đoạn [a,b]. Với từng N thay đổi thì tạo mảng tương ứng với N và đếm số lần so sánh và gán.

```
18 def cmpfunc(a,b,N):
19     for i in range(10,1000,10):
20         print("- Khi k =",i)
21         arr = functionNk(N, i)
22         # if i==10:
23         #     print(arr)
24         print("+ So sanh ", N)
25         print("+ Gan " len(arr[(arr>=a)&(arr<=b)]))
26
```

“def cmpfunck(a, b, N):” Hàm tên cmpfunck tức là compare function k (được hiểu là so sánh khi k thay đổi) với 3 đối số đầu vào là a, b (đoạn [a,b] để kiểm tra xem có bao nhiêu phần tử nằm trong đoạn này) và N. Hàm này mục đích kiểm tra khi k thay đổi (N cố định) thì có bao nhiêu phần tử được tạo ra ngẫu nhiên đó nằm trong đoạn [a,b]. Với việc khoảng giá trị của N phần tử được sinh ngẫu nhiên đó thì mảng đó có bao nhiêu phép so sánh và bao nhiêu phép gán.

```
28      #case 1
29      a=40
30      b=70
31      k=100
32      cmpfuncN(a,b,k)
33
34      #case 2
35      a=50
36      b=70
37      N=20000
38      cmpfunck(a,b,N)
```

Cuối cùng ta lần lượt thử với các trường hợp như bài toán yêu cầu.

Trường hợp 1 là ta khởi tạo đoạn [a,b] cho trước tương ứng là đoạn [40,70] và khoảng giá trị của N phần tử trong mảng là [1...100]. Ta thu được kết quả:

```
test1 ×
↑ - Khi N = 9930
↓ + So sanh 9930
↻ + Gan 3019
⇅ - Khi N = 9940
⇅ + So sanh 9940
⇅ + Gan 3113
⇅ - Khi N = 9950
⇅ + So sanh 9950
⇅ + Gan 3082
⇅ - Khi N = 9960
⇅ + So sanh 9960
⇅ + Gan 3087
⇅ - Khi N = 9970
⇅ + So sanh 9970
⇅ + Gan 3128
⇅ - Khi N = 9980
⇅ + So sanh 9980
⇅ + Gan 3082
⇅ - Khi N = 9990
⇅ + So sanh 9990
⇅ + Gan 3057
⇅ - Khi N = 10000
⇅ + So sanh 10000
⇅ + Gan 3061

Process finished with exit code 0
```

Trường hợp 2 là ta cũng khởi tạo đoạn $[a,b]$ cho trước tương ứng là đoạn $[40,70]$ và khoảng giá trị của N cố định ($N = 20000$) phần tử trong mảng là $[1\dots k]$. Ta thu được kết quả:

```
test1 x
- Khi k = 920
+ So sanh 20000
+ Gan 444
- Khi k = 930
+ So sanh 20000
+ Gan 450
- Khi k = 940
+ So sanh 20000
+ Gan 476
- Khi k = 950
+ So sanh 20000
+ Gan 436
- Khi k = 960
+ So sanh 20000
+ Gan 468
- Khi k = 970
+ So sanh 20000
+ Gan 424
- Khi k = 980
+ So sanh 20000
+ Gan 410
- Khi k = 990
+ So sanh 20000
+ Gan 450

Process finished with exit code 0
```

Nhận xét.

Tùy thuộc vào đoạn $[a,b]$ mà ta có thể thấy. Ở trường hợp 1 thì số lần so sánh cũng chính là số lượng phần tử (bởi ta sẽ duyệt hết mọi phần tử), nhưng số lần gán thì gần gấp 12 lần k . Ở trường hợp 2, khi $k = 990$ thì số lần gán là 450, $k = 980$ thì số lần gán là 410. Cảm giác nhìn kết quả đó ta thấy k gần như gấp đôi số lần

gán, tức biến gán cũng chính là biến đếm số số phần tử của A nằm trong đoạn $[a,b]$.

Có thể thấy, với từng hàm. Như hàm $\text{cmpfuncN}()$, giả sử khi $N = 10$ thì vòng lặp duyệt sẽ truyền 10 vào hàm $\text{functionNk}()$ thì với $i = 1, 2, \dots, 10$ thì có k, k, \dots, k lần nên độ phức tạp là $O(n + k)$. Tương tự với hàm $\text{cmpfuncn}()$, ta cũng dễ dàng tính được độ phức tạp là $O(n + k)$. Suy ra: độ phức tạp thuật toán là $O(n + k)$.

Ta thấy với trường hợp 1, $\text{sosanh} + \text{gan} = N + 12k$. Còn ở trường hợp 2 thì $\text{sosanh} + \text{gan} = N + \frac{1}{2}k$.