Chuyên đề nâng cao:
Thuật toán Pollard's rho – Tìm ước của một hợp số

Tác giả: Phan Thành Hưng (PTNK, ĐHQG TP. HCM)

I. Giới thiệu

Cho bài toán sau:

"Cho một hợp số n, hãy xác định một ước số D của n sao cho $1 < D < n$"

- **Lưu ý:**
  - Hợp số là số tự nhiên lớn hơn 1, có nhiều hơn hai ước (chia hết cho 1, chính nó và ít nhất một số khác).
  - Những số D thỏa mãn $1 < D < n$ và n chia hết cho D được gọi là **ước không tầm thường (non-trivial divisor)**. Từ giờ trở đi, mình sẽ gọi chung là D.

II. Chú thích

1. Độ phức tạp thời gian

Độ phức tạp thời gian, được viết dưới dạng Big-O. Đây là thước đo tốc độ của thuật toán. Nó cho biết khi số càng lớn thì máy tính sẽ mất thêm bao nhiêu bước tính toán để ra kết quả. Ký hiệu $O(\dots)$ giúp ta phân loại thuật toán nào "nhanh" hay "chậm".

Có thể nói, $O\big(f(n)\big)$ nghĩa là số vòng lặp sẽ tỉ lệ thuận với $f(n)$. Khi biểu diễn, ta có thể lược bỏ các hằng số và các thành phần bậc thấp không đáng kể.. Ví dụ: $O(3n+2) = O(n)$, $O(\pi n^2 + n) = O(n^2)$,...

2. Ký hiệu boolean cơ bản

Là cách máy tính hiểu đúng/sai dưới dạng con số 0 và 1.

- True (Đúng): Tương ứng với số 1.
- False (Sai): Tương ứng với số 0.

Phép toán logic trong bài:

- AND ($\wedge$): Chỉ đúng khi cả hai cùng đúng.
- OR ($\vee$): Chỉ cần một cái đúng là đúng.
- NOT ($\neg$): Đảo ngược ($0 \to 1$, $1 \to 0$).

Ứng dụng trong phần tới: Ta sẽ dùng các biểu thức toán học để ép máy tính thực hiện logic dạng "Nếu đúng thì làm A, nếu sai thì làm B" mà không cần nút lệnh điều kiện if-else chuyên dụng.

III. Thuật toán

1. Thuật toán vét cạn

Nếu $D > \sqrt{n}$ thì tồn tại một ước không tầm thường D' khác sao cho $D' = \frac{n}{D} < \sqrt{n}$. Do đó, luôn tồn tại giá trị $D \leq \sqrt{n}$, nên ta có thể dễ dàng tìm D bằng cách chỉ cần duyệt từ 2 tới $\sqrt{n}$. Ta có thể sử dụng tính năng table, đa câu lệnh, hoặc tốt nhất là dùng biểu thức sau:

$$\sum_{D=A}^{\text{Int}(\sqrt{n})+1} \left(\frac{0}{n - D \times \text{Int}\left(\frac{n}{D}\right)} \right)$$

Với A ban đầu có giá trị là 2. Sau đó, ta lần lượt thử nâng A lên cho tới khi biểu thức trên chạy được trên máy tính mà không báo lỗi **Math ERROR**. Thì giá trị lớn nhất của A mà biểu thức trên còn báo lỗi chính là giá trị D cần tìm.

- **Lưu ý:** Thực ra đây mới là một trong những cách làm nhanh, gọn và ổn định nhất để xác định ước của một số, mình sẽ viết một bài viết hướng dẫn cách tìm ước, phân tích thừa số nguyên tố bằng kỹ thuật trong biểu thức trên.

2. Thuật toán Pollard's rho

a. Nghịch lý ngày sinh nhật (Birthday Paradox)

"Hãy tưởng tượng bạn đang ở một bữa tiệc nhỏ, chỉ có khoảng 23 người. Chủ nhà bỗng nhiên cao hứng tuyên bố: "Tôi cá 2 triệu đồng là trong phòng này có ít nhất hai người trùng ngày sinh nhật với nhau!"

Bạn nhìn quanh căn phòng. Chỉ có 23 người, trong khi một năm có tận gần 367 ngày. Tỉ lệ để ai đó trùng ngày sinh với nhau chắc hẳn phải cực thấp. Bạn thầm nghĩ: "Kèo thơm! Xác suất trúng chắc chỉ tầm 6-7% là cùng." Bạn gật đầu cái rụp, hy vọng kiếm được tiền triệu dễ dàng.

Nhưng khi chủ nhà bắt đầu hỏi từng người... Bùm! Có một anh chàng và một cô nàng cùng hô lên "Ngày 02 tháng 02". Và đó là cách mà bạn làm mất 2 triệu :<"

Vậy điều gì khiến bạn sập bẫy, có phải chăng bạn đã quá ngây thơ và dại khờ? Có lẽ một phần là vậy thật, nhưng, sai lầm lớn nhất với những người giỏi như bạn là việc bạn chỉ nghĩ tới việc "có ai trùng ngày sinh với **MÌNH** không?". Nếu đúng là như vậy, bạn đúng, tỉ lệ rất thấp. Tuy nhiên, thực tế thì ta đang xét tới việc có một cặp (bất kì cặp nào) có trùng ngày sinh là được. Do đó, với 23 người, số lượng cặp lên tới 253 cặp. Lúc này, bằng giác quan cũng thấy, tỉ lệ xuất hiện một cặp thực sự trùng nhau đã tăng đáng kể.

Vậy cụ thể là như nào? Ta sẽ đi tìm xác suất không có cặp nào có cùng ngày sinh.

- Người thứ nhất có 365 lựa chọn ngày sinh, tỉ lệ là $365/365$
- Người thứ hai có 364 lựa chọn, tỉ lệ là $364/365$
- Người thứ ba có 363 lựa chọn, tỉ lệ là $363/365$
- …

Cứ như thế tới người cuối cùng. Để chúng xảy ra đồng thời, ta nhân các tỉ lệ lại với nhau:

$$\frac{365}{365} \times \frac{364}{365} \times \ldots \times \frac{343}{365} \approx 0.49$$

Nghĩa là chỉ có khoảng 49% cơ hội để không có cặp nào có ngày sinh trùng nhau, hay có tới khoảng 51% cơ hội để ít nhất một cặp trùng nhau!
Và với một bữa tiệc có 36 người thì xác suất này lên tới 83%, 50 người thì xác suất này là 97%, một bữa tiệc 67 người thì xác suất này là 99%.

Tổng quát lên, ta có xác suất để trong số k người thì có ít nhất một cặp trùng ngày sinh trong một năm có S ngày là:

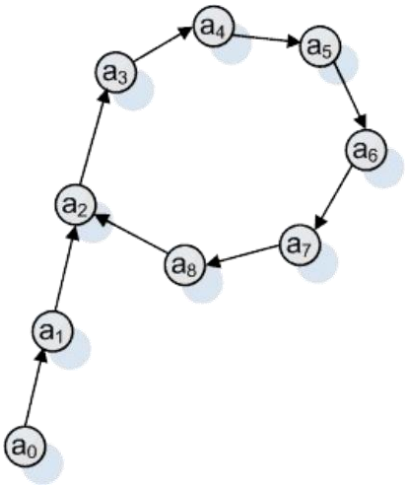
$$1 - \prod_{i=0}^{k-1}\left(1 - \frac{i}{S}\right)$$

Người ta đã xấp xỉ giá trị trên bằng $1 - e^{-\frac{k^2}{2S}}$ bằng một vài "kỹ thuật thú vị".
(*) Để xác suất tồn tại một cặp trùng ngày sinh là khoảng 50% thì số người cần là:

$$1 - e^{-\frac{k^2}{2S}} \approx 0.5 \Leftrightarrow k \approx \sqrt{-2S \cdot \ln 0.5} \approx \sqrt{S}$$

b. Thuật toán tìm chu trình Floyd (Rùa và Thỏ)

Xét một đồ thị hàm số (functional graph, trong lý thuyết đồ thị) có hướng có hình dạng giống chữ rho (ρ) như hình bên phải và có đúng một chu trình.

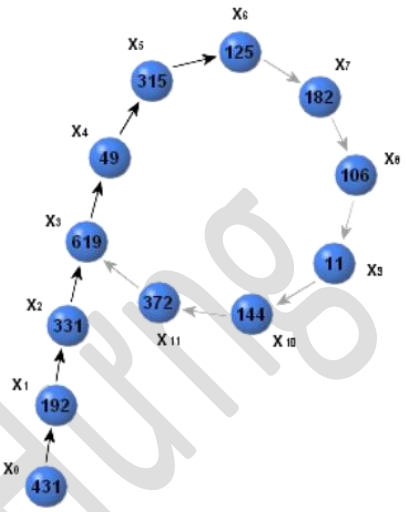


Ban đầu, có một chú rùa và chú thỏ ở vị trí a_0. Sau mỗi đơn vị thời gian, rùa sẽ di chuyển 1 ô, còn thỏ sẽ di chuyển 2 ô theo hướng mũi tên. Ta nhận thấy rằng, sau một khoảng thời gian, cả hai sẽ cùng tiến vào một vòng lặp, và đến một lúc nào đó, chúng sẽ cùng ở chung một ô trong cùng thời điểm trong chu trình.

Vì rùa đi chậm hơn thỏ nên ta sẽ chỉ xét rùa với trường hợp tệ nhất. Trước khi tiến vào chu trình, rùa đi nhiều nhất α bước, với α là số đỉnh không nằm trong chu trình. Và khi cả hai đã cùng nằm trong chu trình, vì rùa đi chậm hơn thỏ 1 bước nên cứ sau mỗi một đơn vị thời gian thì khoảng cách giữa rùa và thỏ trong chu trình giảm đi 1. Do đó, chúng sẽ gặp nhau trong tối đa là β bước (đơn vị thời gian) với β là số đỉnh nằm trong

chu trình. Do đó, tổng số bước là $O(\alpha + \beta) = O(n)$ với n là tổng số đỉnh trong đồ thị. Vậy độ phức tạp thời gian của thuật toán này là $O(n)$, và điều **quan trọng** nhất, độ phức tạp không gian thêm là $O(1)$ do ta chỉ khởi tạo một vài biến (rùa và thỏ).

c. THUẬT TOÁN CHÍNH (Pollard's rho)



Đầu tiên, chọn một hàm lặp, điển hình nhất là:

$$f(x) = (x^2 + c) \bmod n$$

Với c là một hằng số.

Tạo dãy: $x_0, x_1 = f(x_0), x_2 = f(x_1), \dots, x_i = f(x_{i-1})$.

Vì lấy mod n, dãy cuối cùng cũng sẽ lặp lại (chu kì), tạo thành một hình dạng đồ thị hình rho (ρ) giống như đồ thị trong phần **Thuật toán tìm chu trình Floyd (Rùa và Thỏ)** phía trên.

Xét tập S là các giá trị phân biệt được sinh bởi hàm lặp. Giả sử ta tìm được hai số $x, y \in S$ và $x \neq y$ sao cho:

$$x \equiv y \pmod D$$

Thì D là ước của $|x - y|$.

Khi đó, $g = GCD(|x - y|, n) = D$, ta có thể dễ dàng tìm ra được D nếu thỏa mãn điều kiện $1 < g < n$. Nếu $g = 1$, ta tiếp tục thử lại với x, y khác. Nếu $g = n$, ta thử lại với một tập hợp S khác, nghĩa là một hàm lặp khác.

Với D là ước của n, các giá trị trong S khi chia lấy dư cho D chỉ có tối đa D giá trị phân biệt. Theo **Nghịch lý ngày sinh nhật (Birthday Paradox)**, để xác suất tìm được một cặp x, y thỏa mãn $x \equiv y \pmod D$ vượt quá 50% thì số lần chạy thử cần là $\approx O(\sqrt{D}) \ll O(\sqrt{n})$. Do vậy, việc thuật toán tìm được giá trị D thỏa mãn là rất nhanh. Lúc này, ta chỉ cần mô phỏng lại **Thuật toán tìm chu trình Floyd (Rùa và Thỏ)** bằng cách:

- Khởi tạo một biến x, y, đều mang giá trị ngẫu nhiên ban đầu.
- Khởi tạo $D = 1$.
- Lặp cho tới khi $D \neq 1$:
  - $x \coloneqq f(x)$ // rùa
  - $y \coloneqq f(f(y))$ // thỏ
  - $D \coloneqq GCD(|x - y|, n)$
- Nếu $D = n \rightarrow$ thử lại hàm f với một hằng số c mới.
- Nếu $D \neq n$ thì ta đã tìm được D thỏa mãn.

Như đã phân tích, sau khoảng $O(\sqrt{D})$ lần lặp thì ta sẽ tìm được một giá trị ứng cử viên cho D, và qua thực nghiệm và phân tích cho thấy, số lần phải thử lại hàm f là rất ít, có

thể coi là hằng số trung bình $O(1)$ nếu các tham số được chọn ngẫu nhiên. Do vậy, **độ phức tạp trung bình** của thuật toán là $O(\sqrt{D})$ với D là ước không tầm thường nhỏ nhất của n, nghĩa là độ phức tạp trung bình tổng thể là $O(\sqrt[4]{n})$. Độ phức tạp không gian là $O(1)$, nhờ vào thuật toán tìm chu trình Floyd.

- **Lưu ý:** Với n là số nguyên tố, không tồn tại ước số không tầm thường, do vậy thuật toán không hoạt động hiệu quả được. Vì thế, ta cần phải đảm bảo n là hợp số trước khi chạy thuật toán Pollard's rho. Thông thường, người ta có thể sử dụng phép thử số nguyên tố Rabin - Miller để kiểm tra.

➢ Cài đặt vào CASIO

Ta cần xử lý điều kiện "lặp cho tới khi $D \neq 1$". Ta có thể tạo một biến F cho phép ta quản lý việc "có lặp tiếp hay không?". Nếu $F = 0$ thì nghĩa là $D \neq 1$ và ta cho dừng vòng lặp. Nhận thấy:

$$\begin{cases} F = 0, & nếu\ D - 1 \neq 0 \\ F = 1, & nếu\ D - 1 = 0 \end{cases}$$

Ta có thể dùng hàm cos để xác định giá trị F. Xét **đơn vị là radian**,

$$\begin{cases} \mathrm{F} = \mathrm{Int}(\cos x) = 0, & \Leftrightarrow x \neq 0, x \in \mathbb{N} \\ \mathrm{F} = \mathrm{Int}(\cos x) = 1, & \Leftrightarrow x = 0, x \in \mathbb{N} \end{cases}$$

Lúc này, nếu tiếp tục vòng lặp, giá trị x, y sẽ giữ nguyên để chạy tiếp, ta có thể xử lý chúng một cách đơn giản bằng phép gán:

$$\begin{cases} x := (1 - F)x + Fx \\ y := (1 - F)x + Fy \end{cases}$$

Tiếp theo, ta cần xử lý việc dừng thuật toán. Ta dừng thuật toán khi $1 < D < n$, nghĩa là $D \neq 1$ **VÀ** $D \neq n$. Theo Định luật De Morgan trong logic học và toán học tập hợp, điều đó tương đương với việc không phải ($D = 1$ hoặc $D = n$) xảy ra. Cụ thể: $\neg(D = 1) \wedge \neg(D = n) = \neg(D = 1 \vee D = n)$. Nếu xét chúng là một đại lượng mang giá trị 0/1 thì nghĩa là $(D = 1) + (D = n) = 0 \Leftrightarrow F + (D = n) = 0$. Nhận thấy rằng ta có thể cố tình ép máy tính gây lỗi để báo dừng bằng cách đưa biểu thức trên xuống mẫu số của một phân số để máy báo **Math ERROR**.

Để mô phỏng giá trị boolean của $D = n$, ta có thể dùng kỹ thuật hàm cos ở trên. Do đó, biểu thức giúp ta dừng vòng lặp là: $0 \div \big(F + \mathrm{Int}(\cos(D - n))\big)$.

Kết hợp những thứ ở trên, ta có một thuật toán hoàn chỉnh cài được vào máy tính Casio fx-580 VN X:

$$x = 0MFD + (1 - F) \times \text{RanInt\#}(2, M - 2) + Fx:$$
$$y = (1 - F)x + Fy:$$
$$C = (1 - F) \times \text{RanInt\#}(1, M - 1) + FC:$$
$$x = x^2 + C - M\text{Int}\left(\frac{x^2+C}{M}\right):$$
$$y = y^2 + C - M\text{Int}\left(\frac{y^2+C}{M}\right):$$
$$y = y^2 + C - M\text{Int}\left(\frac{y^2+C}{M}\right):$$
$$D = GCD(|x - y|, M):$$
$$F = \text{Int}(\cos(D - 1)):$$
$$0 \div \left(F + \text{Int}(\cos(D - M))\right)$$

Khởi tạo: $M = số\ cần\ phân\ tích,\ F = 0$, các biến còn lại để **nguyên không âm** tùy ý.
Cách dùng: Lặp đa câu lệnh tới khi gặp **Math ERROR** thì dừng, kết quả trong biến D.

IV. Kết luận & Tính khả thi của thuật toán trong thực tế

Mặc dù có độ phức tạp $O(\sqrt[4]{n})$, thuật toán này chỉ phù hợp với lập trình thực tế, việc bấm đa câu lệnh thủ công trên máy tính cầm tay vẫn là rất chậm, đặc biệt là trong phòng thi. Mình viết bài này nhằm mục đích học thuật, và cũng cho thấy rằng, ta hoàn toàn có thể thử thách giới hạn và cài một thuật toán phức tạp như Pollard's rho vào chiếc máy tính cầm tay nhỏ bé xinh xắn cute phô mai que dễ huông này :3

Bạn có thể tham khảo thêm bài viết này về cách cài đặt thuật toán Pollard's rho trên máy tính Casio fx-880 BTG một cách nhanh chóng:

- https://giaoduc.bitexedu.com/thcs/tai-lieu-thcs/toan-thcs/hsg-casio-thcs/bai-2-ly-thuyet-so

Tài liệu tham khảo:

- https://en.wikipedia.org/wiki/Pollard%27s_rho_algorithm
- https://cp-algorithms.com/algebra/factorization.html#pollards-rho-algorithm
- https://en.wikipedia.org/wiki/Birthday_problem
- https://cp-algorithms.com/others/tortoise_and_hare.html
- https://www.geeksforgeeks.org/dsa/floyds-cycle-finding-algorithm
- https://en.wikipedia.org/wiki/Cycle_detection#Floyd's_tortoise_and_hare

Nguồn trích dẫn hình ảnh:

- *File:Pollard Rho.png - Wikimedia Commons*
- *File:Rho-pollard.png - Wikimedia Commons*