Table of Contents

Tác giả: Nguyễn *RR* Thành Trung, Nguyễn Mạnh Quân

Mở đầu

Thông thường, để đạt được độ phức tạp thuật toán như mong muốn, cách làm thường là tìm ra một thuật toán ban đầu làm cơ sở, rồi từ đó dùng các kỹ năng để giảm độ phức tạp của thuật toán. Trong bài viết này, tôi xin giới thiệu với bạn đọc một kỹ năng khá thông dụng: nhân ma trận.

Trước khi đọc bài viết này, nếu bạn chưa có khái niệm gì về ma trận, bạn có thể tham khảo định nghĩa về ma trận trong một tài liệu khác.

Trước hết, tôi xin nhắc lại tóm tắt khái niệm về phép nhân ma trận:

- Cho 2 ma trận: A kích thước $M * N$ và B kích thước $N * P$. (chú ý số cột của ma trận A phải bằng số hàng của ma trận B thì mới có thể thực hiện phép nhân).
- Kết quả phép nhân ma trận A và B là ma trận C kích thước $M * P$, với mỗi phần tử của ma trận C được tính theo công thức: $C(i,j) = \sum A(i,k) * B(k,j)$

Để thực hiện phép nhân ma trận trên máy tính, ta có thể thực hiện thuật toán với độ phức tạp $O(M * N * P)$ như sau:

```
for i:=1 to M do
for j:=1 to P do
begin
 C[i,j]:=0;
 for k:=1 to N do
  C[i,j]:=C[i,j]+A[i,k] * B[k,j];
end;
```

Đối với phép nhân các ma trận vuông kích thước $N * N$, có thuật toán nhân ma trận Strassen với độ phức tạp $O(N^{\log 7})$ theo tư tưởng chia nhỏ ma trận (tương tự cách nhân nhanh 2 số lớn)). Tuy nhiên cài đặt rất phức tạp và trên thực tế với giá trị N thường gặp, cách này không chạy nhanh hơn nhân ma trận thông thường $O(N^3)$.

Cần chú ý thêm là phép nhân ma trận không có tính giao hoán (do có thể thực hiện nhân 2 ma trận A kích thước $M * N$ và ma trận B kích thước $N * P$ nhưng không thể thực hiện phép nhân $B * A$ nếu $P \neq M$). Tuy nhiên, nhân ma trận lại có tính kết hợp:

$(A * B) * C = A * (B * C)$

Ví dụ 1

Chúng ta hãy cùng xem xét một ví dụ kinh điển nhất trong ứng dụng của phép nhân ma trận.

Bài toán

Dãy Fibonacci được định nghĩa như sau:

```
F(0) = 1
F(1) = 1
...
F(i) = F(i-1) + F(i-2), i >= 2
```

Yêu cầu: Cho N ($N \leq 10^9$), tính $F(N)$.

Bạn có thể nộp bài thử ở VNOI - LATGACH4.

Phân tích

Hiển nhiên cách làm thông thường là tính lần lượt các $F(j)$. Tuy nhiên, cách làm này hoàn toàn không hiệu quả với N lên đến 10^9, và ta cần một cách tiếp cận khác.

Ta xét các lớp số:

- Lớp 1: $F(1), F(2)$
- Lớp 2: $F(2), F(3)$
- ...
- Lớp i: $F(i), F(i+1)$

Ta hình dung mỗi lớp là một ma trận 1x2. Tiếp đó, ta sẽ biến đổi từ lớp $i-1$ đến lớp i. Sau mỗi lần biến đổi như vậy, ta tính thêm được một giá trị $F(i+1)$. Để thực hiện phép biến đổi này, chú ý là các số ở lớp sau chỉ phụ thuộc vào lớp ngay trước nó theo các phép cộng, ta tìm được cách biến đổi bằng nhân ma trận:

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} * \begin{pmatrix} F_{i-1} \\ F_i \end{pmatrix} = \begin{pmatrix} F_i \\ F_{i+1} \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^T * \begin{pmatrix} F_0 \\ F_1 \end{pmatrix} = \begin{pmatrix} F_T \\ F_{T+1} \end{pmatrix}$$

Chắc hẳn đọc đến đây bạn đọc sẽ thắc mắc, làm thế nào để tìm được ma trận trên? Để tìm được ma trận này, ta làm như sau:

Ta có:

- $F(i) = 0 * F(i-1) + 1 * F(i)$, do đó hàng đầu của ma trận là $[0, 1]$
- $F(i+1) = 1 * F(i-1) + 1 * F(i)$, do đó hàng thứ 2 của ma trận là $[1, 1]$

Bây giờ ta sẽ cần tìm cách tăng tốc việc tính $[0, 1; 1, 1]^T$ (*). Việc tính nhanh (*) cũng hoàn toàn tương tự việc ta tính a^T với a là số nguyên. Sau đây là đoạn code minh hoạ. Trong đoạn code này, để bạn đọc dễ hiểu, tôi bỏ qua yếu tố về tính toán số lớn, và thực hiện các phép tính với kiểu số 32-bit.

```
type
    matrix=array[0..1,0..1] of longint;
const
    a: matrix=((0,1),(1,1));

//Định nghĩa phép nhân 2 ma trận
operator * (a,b:matrix) c:matrix;
var
    i,j,k:longint;
begin
    fillchar(c,sizeof(c),0);
    for i:=0 to 1 do
    for j:=0 to 1 do
        for k:=0 to 1 do
            c[i,j]:=c[i,j]+a[i,k]*b[k,j];
end;

//Tính a^n
function power(n:longint):matrix;
var
    temp:matrix;
begin
    if n=1 then exit(a);
    temp:=power(n div 2);
    temp:=temp*temp;
    if n mod 2=1 then temp:=temp*a;
    exit(temp);
end;
```

Ví dụ 2

Bây giờ chúng ta sẽ cùng xem xét một ví dụ tổng quát hơn của ví dụ 1.

Bài toán: SPOJ - SEQ

Cho số nguyên N ($N \leq 100$) và 2 dãy số độ dài N: $a_1, a_2, \ldots, a_N$; $b_1, b_2, \ldots, b_N$. Dãy số c được định nghĩa như sau:

- $c_i = a_i$ với $i \leq N$
- $c_i = c_{i-1} * b_1 + c_{i-2} * b_2 + \ldots + c_{i-N} * b_N$

Yêu cầu: Tính c_k với $k \leq 10^9$.

Phân tích

Cũng như trong ví dụ 1, ta xét các lớp số:

- Lớp 1: $c_1, c_2, \ldots, c_N$
- Lớp 2: $c_2, c_3, \ldots, c_{N+1}$
- ...
- Lớp i: $c_i, c_{i+1}, \ldots, c_{i+N-1}$

Ta cũng sẽ áp dụng phép nhân ma trận để biến đổi từ lớp i sang lớp $i + 1$ như sau:

$$\begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & & & & \\ 0 & 0 & 0 & \dots & 1 \\ b_N & b_{N-1} & b_{N-2} & \dots & b_1 \end{pmatrix} * \begin{pmatrix} a_i \\ \vdots \\ a_{i+N-1} \end{pmatrix} = \begin{pmatrix} a_{i+1} \\ \vdots \\ a_{i+N} \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & & & & \\ 0 & 0 & 0 & \dots & 1 \\ b_N & b_{N-1} & b_{N-2} & \dots & b_1 \end{pmatrix}^T * \begin{pmatrix} a_1 \\ \vdots \\ a_N \end{pmatrix} = \begin{pmatrix} a_{T+1} \\ \vdots \\ a_{N+T} \end{pmatrix}$$

Để xây dựng ma trận vuông như trên, ta thực hiện tương tự như trong ví dụ trước: Phân tích a_{i+1} đến a_{i+N} dưới dạng $a_i, \dots, a_{i+N-1}$:

- $a_{i+1} = 0 * a_i + 1 * a_{i+1} + \dots + 0 * a_{i+N-1}$ nên hàng 1 là $0, 1, 0, \dots, 0$
- ...
- $a_{i+N-1} = 0 * a_i + 0 * a_{i+1} + \dots + 1 * a_{i+N-1}$ nên hàng $N-1$ là $0, 0, 0, \dots, 1$
- $a_{i+N} = b_N * a_i + b_{N-1} * a_{i+1} + \dots + b_1 * a_{i+N-1}$ nên hàng N là $b_N, b_{N-1}, \dots, b_1$

Từ đó, ta thu được cách làm như trong ví dụ 1. Cài đặt cụ thể xin nhường lại cho bạn đọc.

Chú ý rằng ta hoàn toàn có thể thay thế phép nhân và phép cộng trong định nghĩa phép nhân ma trận, chỉ cần đảm bảo giữ nguyên tính chất kết hợp. Cụ thể hơn, thay vì $C(i,j) = \sum A(i,k) * B(k,j)$, ta có thể định nghĩa phép nhân ma trận: $C(i,j) = min(A(i,k) + B(k,j))$. Từ đó, ta có thể thu được một lớp các bài toán khác.

Sau đây là một ví dụ minh hoạ cho nhóm các bài toán này

Ví dụ 3

Bài toán

Cho đồ thị có hướng N đỉnh $(N \leq 100)$. Tính ma trận $C(k)$ kích thước $N * N$, với $C(k)[i,j]$ là độ dài đường đi ngắn nhất từ i đến j đi qua đúng k cạnh

Phân tích

Xét ma trận A là ma trận kề của đồ thị đã cho. Ta có:

- $A = C(1)$
- $C(2)[i,j] = \min A[i,u] + A[u,j]$ với u chạy từ 1 đến N
- $C(k)[i,j] = \min C(k-1)[i,u] + A[u,j]$ với u chạy từ 1 đến N

Như vậy, nếu ta thay phép nhân và phép cộng trong việc nhân ma trận thông thường lần lượt bởi phép cộng và phép lấy min, ta thu được một phép ”nhân ma trận” mới, tạm dùng ký hiệu `x`, thì:

```
C1 = A
C2 = C1 x C1 = A x C1
C3 = C1 x C2 = A x C2
C4 = C1 x C3 = A x C3
...
Ck = C1 x C(k-1) = A x C(k-1)
```

Do đó, $C(k) = A^k$

Như vậy, bài toán được đưa về bài toán tính lũy thừa của một ma trận, ta hoàn toàn có thể giải tương tự các ví dụ trước. Cài đặt phép nhân ma trận mới này hoàn toàn không phức tạp hơn cài đặt phép nhân ma trận thông thường. Việc cài đặt xin nhường lại cho bạn đọc.

Ví dụ 4

VOJ - THBAC

Bài toán

Người ta mới tìm ra một loại vi khuẩn mới. Chúng sống thành N bầy ($N \leq 100$), đánh số từ 0 đến $N-1$. Ban đầu, mỗi bầy này chỉ có một con vi khuẩn. Tuy nhiên, mỗi giây, số lượng vi khuẩn trong các bầy lại có sự thay đổi. Ví dụ:

- một bầy có thể bị chết đi
- số lượng vi khuẩn trong một bầy có thể tăng lên
- một bầy có thể di chuyển vị trí.

Các thay đổi này tuân theo một số quy luật cho trước. Tại mỗi giây chỉ xảy ra đúng một quy luật. Các quy luật này được thực hiện nối tiếp nhau và theo chu kỳ. Có nghĩa là, nếu đánh số các quy luật từ 0 đến $M-1$, tại giây thứ S thì quy luật được áp dụng sẽ là $(S-1) \ mod \ M$ (M \le 1000)

Nhiệm vụ của bạn là tìm xem, với một bộ các quy luật cho trước, sau T đơn vị thời gian (T \le 10^{18}), mỗi bầy có bao nhiêu vi khuẩn.

Các loại quy luật có thể có:

- `A i 0` : Tất cả các vi khuẩn thuộc bầy i chết.
- `B i k` : Số vi khuẩn trong bầy i tăng lên k lần.
- `C i j` : số vi khuẩn bầy thứ i tăng lên một số lượng bằng với số vi khuẩn bầy j.
- `D i j` : Các vi khuẩn thuộc bầy j di chuyển toàn bộ sang bầy i.
- `E i j` : Các vi khuẩn thuộc bầy i và bầy j đổi vị trí cho nhau.
- `F 0 0` : Vị trí các vi khuẩn di chuyển trên vòng tròn.

Phân tích

Cách làm đơn giản nhất là chúng ta mô phỏng lại số lượng vi khuẩn trong mỗi bầy qua từng đơn vị thời gian. Cách làm này có độ phức tạp \mathcal{O}(T * N * k) với \mathcal{O}(k) là độ phức tạp cho xử lý số lớn. Cách này không thể chạy được với T lớn.

Ta hình dung số lượng vi khuẩn trong mỗi bầy trong một đơn vị thời gian là một dãy số. Như vậy, mỗi quy luật cho trước thực chất là một phép biến đổi từ một dãy số thành một dãy số mới, và ta hoàn toàn có thể thực hiện biến đổi này bằng một phép nhân ma trận.

Cụ thể hơn, ta coi số lượng vi khuẩn trong N bầy tại một thời điểm xác định là một ma trận 1 * N, và mỗi phép biến đổi là một ma trận N * N. Khi áp dụng mỗi phép biến đổi, ta nhân hai ma trận nói trên với nhau.

Bây giờ, xét trường hợp N = 4, tôi xin lần lượt mô tả các ma trận tương ứng với các phép biến đổi:

- Biến đổi: `A 2 0`

```
1 0 0 0
0 1 0 0
0 0 0 0
0 0 0 1
```

- Biến đổi: `B 2 6`

```
1 0 0 0
0 1 0 0
0 0 6 0
0 0 0 1
```

- Biến đổi: `C 1 3`

```
1 0 0 0
0 1 0 0
0 0 1 0
0 1 0 1
```

- Biến đổi: `D 1 3`

```
1 0 0 0
0 1 0 0
0 0 1 0
0 1 0 0
```

- Biến đổi: `E 1 3`

```
1 0 0 0
0 0 0 1
0 0 1 0
0 1 0 0
```

- Biến đổi: `F 0 0`

```
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0
```

Cũng như các bài toán trước, ta sẽ cố gắng áp dụng việc tính toán lũy thừa, kết hợp với phép nhân ma trận để giảm độ phức tạp từ T xuống $\log{T}$. Tuy nhiên, có thể thấy việc sử dụng phép lũy thừa trong bài toán này phần nào phức tạp hơn bởi các ma trận được cho không giống nhau. Để giải quyết vấn đề này, ta làm như sau:

Gọi $X_1, X_2, ..., X_m$ là các ma trận tương ứng với các phép biến đổi được cho.

Đặt $X = X_1 * X_2 * ... * X_m$.

Đặt $S = [1, 1, ..., 1]$ (dãy số lượng vi khuẩn tại thời điểm đầu tiên).

Như vậy, $Y = S * X^t * X_1 * X_2 * ... * X_r$ là ma trận thể hiện số lượng vi khuẩn tại thời điểm $M * t + r$.

Như vậy, thuật toán đến đây đã rõ. Ta phân tích $T = M * t + r$, nhờ đó, ta có thể giải quyết bài toán trong $\mathcal{O}(N^3 * M)$ cho bước tính ma trận X và $\mathcal{O}(N^3 * (\log{T/M} + M)$ cho bước tính Y. Bài toán được giải quyết.

Phép toán kết hợp và độ phức tạp tính toán

Nhân tổ hợp dãy ma trận

Trong phần Mở Đầu ta đã có thuật toán nhân hai ma trận A kích cỡ $M * N$ và B kích cỡ $N * P$ cần độ phức tạp $O(M * N * P)$. Giả sử ta có thêm ma trận C có kích cỡ $P * Q$ và ta cần tính tích $A * B * C$. Xét hai cách thực hiện phép nhân này:

- *Cách 1*: $(A * B) * C$ thực hiện nhân A và B rồi nhân với C cần độ phức tạp $O(M * N * P) + O(M * P * Q) = O(M * P * (N + Q))$
- *Cách 2*: $A * (B * C)$ thực hiện nhân B và C rồi nhân với A cần độ phức tạp $O(N * P * Q) + O(M * N * Q) = O(N * Q * (M + P))$

Như vậy là hai cách thực hiện khác nhau cần hai độ phức tạp khác nhau. Chọn $M = N = 500, P = 1000, Q = 2$, cách 1 sẽ cần tới $500 * 1000 * (500 + 2) = 251 * 10^6$ phép tính, trong khi cách 2 chỉ cần $500 * 2 * (500 + 1000) = 1.5 * 10^6$ phép tính, nghĩa là cách 1 chậm hơn cách 2 tới gần 200 lần.

Khi độ dài của dãy ma trận tăng lên, sự khác biệt có thể còn lớn hơn nữa. Ví dụ trên đã cho thấy rằng trong một số trường hợp thứ tự thực hiện phép nhân ma trận có ý nghĩa rất lớn đối với việc tìm lời giải của các bài toán.

Trong thực tế, bài toán xác định thứ tự nhân ma trận hiệu quả nhất là một bài toán rất phổ biến, bạn có thể tìm đọc chi tiết thêm ở Mục 3.5 Phép Nhân Tổ Hợp dãy Ma Trận trong ebook của thầy Lê Minh Hoàng.

Giải thuật Freivalds kiểm tra tích hai ma trận

Giải thuật Freivalds là một ví dụ điển hình về việc áp dụng thứ tự thực hiện phép nhân ma trận để giảm độ phức tạp tính toán của phép nhân một dãy ma trận. Bài toán đặt ra là cho ba ma trận vuông A, B, C có kích cỡ $N * N$ với $N \le 1000$. Ta cần kiểm tra xem C có phải là tích của A và B, nói cách khác ta cần kiểm tra $A*B = C$ có phải là mệnh đề đúng hay không (đây chính là bài VMATRIX - VNOI Marathon 2014).

Phân tích

Cách làm thông thường là nhân trực tiếp hai ma trận A, B rồi so sánh kết quả với C. Như phân tích trong phần Mở Đầu độ phức tạp của cách làm này là $O(N^3)$, với $N = 1000$ thì cách làm này không đủ nhanh. Giải thuật Freivalds thực hiện việc kiểm tra thông qua thuật toán xác suất kiểu Monte Carlo với k lần thử cho xác suất kết luận sai là xấp xỉ $1 / 2^k$, mỗi lần thử có độ phức tạp $O(N^2)$. Các bước cơ bản của một phép thử Freivalds như sau:

1. Sinh ngẫu nhiên một ma trận v kích cỡ $N * 1$ với các phần tử chỉ nhận giá trị 0 hoặc 1.
2. Tính hiệu $P = A * B * v - C * v$. Dễ thấy rằng P là ma trận kích cỡ $N * 1$.
3. Trả về `True` nếu P chỉ gồm phần tử 0 (bằng với vector 0) và `False` nếu ngược lại.

Ta thực hiện k lần thử, nếu gặp phép thử trả về `False` thì ta kết luận là $A * B \neq C$. Ngược lại nếu sau k phép thử mà luôn thấy `True` thì ta kết luận $A * B = C$. Vì xác suất lỗi giảm theo hàm mũ của k nên thông thường chỉ cần chọn k vừa đủ là sẽ thu được xác suất đúng rất cao ($k = 5$ với bài **VMATRIX** ở trên). Một nhận xét quan trọng khác là cận trên của đánh giá xác suất kiểm tra lỗi không phụ thuộc vào kích cỡ N của ma trận được cho mà chỉ phụ thuộc vào số lần thực hiện phép thử.

Xét bước thứ 2, ta thấy rằng phép thử Freivalds chỉ có ý nghĩa nếu như ta có thể thực hiện phép nhân $A * B * v$ trong thời gian $O(N^2)$ (vì phép nhân $C * v$ đã đạt sẵn $O(N^2)$ rồi). Thay vì thực hiện tuần tự từ trái qua phải sẽ cần $O(N^3)$, ta thực hiện theo thứ tự $A * (B * v)$. Vì kết quả của phép nhân B và v là một ma trận $N * 1$ nên độ phức tạp tổng cộng sẽ là $O(N^2)$. Trên tất cả các phép thử, độ phức tạp là $O(k * N^2)$.

Bài tập áp dụng

- HackerEarth - PK and interesting language
- HackerEarth - Long walks from Office to Home Sweet Home
- HackerEarth - Tiles
- HackerEarth - ABCD Strings
- HackerEarth - Mehta and the difficult task
- HackerEarth - Mehta and the Evil Strings
- VOJ - PA06ANT
- VOJ - CONNECTE

Last edited by **Thanh Trung Nguyen**, 2016-11-04 16:06:29