Render

```
ข้อนี้ให้ค่า $A,B,C,D,E,F,G,H$ และกำหนด $a_1=A,
a_2=B,a_3=C,a_4=D$ และ $a_k = F a_{k-1} + E a_{k-2} + G
a_{k-3} + H a_{k-4}$

จากนั้นให้ $Q\leq 200000$ คำถาม ในแต่ละคำถามให้หาค่า $a_N$
สำหรับ $N\leq 10^{18}$

### เคส $ N \leq 1000000$

ในเคสนี้สามารถคำนวณ $a_5, a_6, \dots, a_{1000000}$ ไว้ก่อน
โดยใช้ recurrence ที่โจทย์กำหนดและเก็บมาตอบคำถาม $Q$ ข้อ

การตอบคำถามจะใช้เวลาเพียง $\mathcal{O}(1)$ และการคำนวณ
แต่ละ $a_k = Fa_{k-1} + E a_{k-2} + Ga_{k-3} +H a_{k-4}$ ใช้
$\mathcal{O}(1)$ สำหรับแต่ละ $k$ เช่นกันซึ่งต้องทำ 1000000
ครั้ง จึงเร็วเพียงพอสำหรับเคสนี้

### เคส $Q \leq 2000$

สำหรับเคสนี้สามารถสังเกต

$\begin{bmatrix}\ a_k \\\ a_{k-1} \\\ a_{k-2} \\\ a_{k-3}
\end{bmatrix}= \begin{bmatrix} E & F & G & H \\ 1 &  0 &  0
&  0 \\ 0 &  1 &  0 &  0 \\ 0 &  0 &  1 &  0 \end{bmatrix}
\begin{bmatrix}\ a_{k-1} \\\ a_{k-2} \\\ a_{k-3} \\\ a_{k-4}
\end{bmatrix}$

ให้ $A = \begin{bmatrix} E & F & G & H \\ 1 &  0 &  0 &  0 \\
0 &  1 &  0 &  0 \\ 0 &  0 &  1 &  0 \end{bmatrix} $ จะได้ว่า
$\begin{bmatrix} \ a_k \\\ a_{k-1} \\\ a_{k-2} \\\ a_{k-3}
\end{bmatrix}= A \begin{bmatrix}\ a_{k-1} \\\ a_{k-2} \\\
a_{k-3} \\\ a_{k-4} \end{bmatrix}$

สังเกตว่า

 $\begin{bmatrix} \ a_N \\\ a_{N-1} \\\ a_{N-2} \\\ a_{N-3}
\end{bmatrix}= A\begin{bmatrix}\ a_{N-1} \\\ a_{N-2} \\\
a_{N-3} \\\ a_{N-4} \end{bmatrix} = A^2 \begin{bmatrix}\
a_{N-2} \\\ a_{N-3} \\\ a_{N-4} \\\ a_{N-5} \end{bmatrix} =
\dots  = A^{N-4} \begin{bmatrix} \ a_{4}\\\ a_{3}\\\ a_{2}\\\
a_{1}\end{bmatrix} = A^{N-4} \begin{bmatrix} \ D \\\ C\\\ B
\\\ A \end{bmatrix}$

หากใช้ Matrix Exponentiation จะทำให้คำนวณ $A^{N-4}$ ได้ใน
เวลา $\mathcal{O}(M \log N)$ เมื่อ $M$ คือเวลาที่ใช้ในการทำ
Matrix Multiplication หนึ่งรอบ หากใช้วิธีปกติจะเป็น
$\mathcal{O}(r^3)$ สำหรับ Matrix ขนาด $r\times r$ ($r=4$)

เมื่อทำสำหรับทุกคำถามจะเป็น $\mathcal{O}(Q r^3 \log N)$ ซึ่ง
เร็วพอสำหรับ $Q \leq 2000$ แต่อาจช้าไปสำหรับเคสที่ใหญ่กว่านั้น

#### Matrix Multiplication

วิธี Matrix Exponentiation เริ่มจากการสังเกตว่าหาก $N =
(b_{\lfloor \log N \rfloor } b_{\lfloor \log N \rfloor -1} \dots
b_0)_2$ นั้นคือ $N$ มี Binary Representation (การแทนแบบ
```

ข้อนี้ให้ค่า A,B,C,D,E,F,G,H และกำหนด $a_1=A, a_2=B,a_3=C,a_4=D$ และ $a_k = F a_{k-1} + E a_{k-2} + G a_{k-3} + H a_{k-4}$

จากนั้นให้ $Q\leq 200000$ คำถาม ในแต่ละคำถามให้หาค่า a_N สำหรับ $N\leq 10^{18}$

เคส $N \leq 1000000$

ในเคสนี้สามารถคำนวณ $a_5, a_6, \dots, a_{1000000}$ ไว้ก่อนโดยใช้ recurrence ที่โจทย์กำหนดและเก็บมาตอบคำถาม Q ข้อ

การตอบคำถามจะใช้เวลาเพียง $\mathcal{O}(1)$ และการคำนวณแต่ละ $a_k = Fa_{k-1} + E a_{k-2} + Ga_{k-3} +H a_{k-4}$ ใช้ $\mathcal{O}(1)$ สำหรับแต่ละ k เช่นกันซึ่งต้องทำ 1000000 ครั้ง จึงเร็วเพียงพอสำหรับเคสนี้

เคส $Q \leq 2000$

สำหรับเคสนี้สามารถสังเกต

$$\begin{bmatrix} a_k \\ a_{k-1} \\ a_{k-2} \\ a_{k-3} \end{bmatrix}= \begin{bmatrix} E & F & G & H \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_{k-1} \\ a_{k-2} \\ a_{k-3} \\ a_{k-4} \end{bmatrix}$$

ให้ $A = \begin{bmatrix} E & F & G & H \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ จะได้ว่า $\begin{bmatrix} a_k \\ a_{k-1} \\ a_{k-2} \\ a_{k-3} \end{bmatrix}= A \begin{bmatrix} a_{k-1} \\ a_{k-2} \\ a_{k-3} \\ a_{k-4} \end{bmatrix}$

สังเกตว่า

$$\begin{bmatrix} a_N \\ a_{N-1} \\ a_{N-2} \\ a_{N-3} \end{bmatrix}= A\begin{bmatrix} a_{N-1} \\ a_{N-2} \\ a_{N-3} \\ a_{N-4} \end{bmatrix} = A^2 \begin{bmatrix} a_{N-2} \\ a_{N-3} \\ a_{N-4} \\ a_{N-5} \end{bmatrix} = \dots = A^{N-4} \begin{bmatrix} a_{4}\\ a_{3}\\ a_{2}\\ a_{1}\end{bmatrix} = A^{N-4} \begin{bmatrix} D \\ C\\ B \\ A \end{bmatrix}$$

หากใช้ Matrix Exponentiation จะทำให้คำนวณ A^{N-4} ได้ในเวลา $\mathcal{O}(M \log N)$ เมื่อ M คือเวลาที่ใช้ในการทำ Matrix Multiplication หนึ่งรอบ หากใช้วิธีปกติจะเป็น $\mathcal{O}(r^3)$ สำหรับ Matrix ขนาด $r\times r$ ($r=4$)

เมื่อทำสำหรับทุกคำถามจะเป็น $\mathcal{O}(Q r^3 \log N)$ ซึ่งเร็วพอสำหรับ $Q \leq 2000$ แต่อาจช้าไปสำหรับเคสที่ใหญ่กว่านั้น

Matrix Multiplication

วิธี Matrix Exponentiation เริ่มจากการสังเกตว่าหาก $N = (b_{\lfloor \log N \rfloor } b_{\lfloor \log N \rfloor -1} \dots b_0)_2$ นั้นคือ N มี Binary Representation (การ

ฐานสอง) เป็น $b_{\lfloor \log N \rfloor } b_{\lfloor \log N \rfloor -1} \dots b_0$ จะได้ว่า $N = 2^{\lfloor \log N \rfloor}{b_{\lfloor \log N \rfloor }} + 2^{\lfloor \log N \rfloor -1}{b_{\lfloor \log N \rfloor -1}} + \dots + 2^0{b_0}$

ซึ่งทำให้ได้ว่า $A^N = A^{2^{\lfloor \log N \rfloor}{b_{\lfloor \log N \rfloor }} }A^{2^{\lfloor \log N \rfloor -1}{b_{\lfloor \log N \rfloor -1}}} \dots A^{2^0{b_0}} $

เช่นถ้า $N= 13 = 1101_2 $ ก็จะได้ $A^{13} = A^{2^3 \times 1} A^{2^2 \times 1} A^{2^1 \times 0} A^{2^0 \times 1} = A^8 A^4 A^1$

สังเกตว่า A^{2^i} คำนวณได้จาก $A^{2^{i}} = A^{2^{i-1}} A^{2^{i-1}} $ ดังนั้นการคำนวณ $A^{2^{i}}$ สำหรับทุก i ตั้งแต่ 1 ถึง $\lfloor \log N \rfloor$ จะใช้เวลา $\mathcal{O}(r^3 \log N) $ ดังนั้นการคำนวณ A^N เพียงต้องเลือกเฉพาะค่า i ที่ $b_i = 1$ ใน Binary Represenation ของ N มาคูณกัน ซึ่งใช้เวลา $\mathcal{O}(r^3 \log N)$ เช่นกัน

ตัวอย่างการเขียน Matrix Exponentiation

เริ่มด้วย Function MUL(X,Y,R) สำหรับการทำ Matrix Multiplication เพื่อแก้ค่า R ให้เป็น $X Y$

```cpp
void mul(int X[4][4], int Y[4][4], int R[4][4]) {
  int M[4][4] = {};
  for (int r = 0; r < 4; r++)
    for (int c = 0; c < 4; c++)
      for (int k = 0; k < 4; k++)
        M[r][c] += X[r][k] * Y[k][c];

  for (int r = 0; r < 4; r++)
    for (int c = 0; c < 4; c++)
      R[r][c] = M[r][c];
}
```

ในการคำนวณ A^N เราจะไล่ตั้งแต่ $i=0$ ถึง $i=\lfloor \log N \rfloor $ และหา A^{2^i} โดย สำหรับ $i=0$ จะเอาค่าจาก A มาโดยตรง ส่วนสำหรับ $i \geq 1$ จำใช้ $A^{2^i} = A^{2^{i-1}} A^{2^{i-1}}$ ดังที่อธิบายไว้ ส่วนการคำนวณ ผลลัพท์จะเริ่มจากตั้ง $C = I$ (เมทริกซ์เอกลักษณ์) และแก้เป็น $C = A^{2^i}$ เมื่อเลขตัว

แทนแบบฐานสอง) เป็น $b_{\lfloor \log N \rfloor} b_{\lfloor \log N \rfloor -1} \dots b_0$ จะได้ว่า $N = 2^{\lfloor \log N \rfloor} b_{\lfloor \log N \rfloor} + 2^{\lfloor \log N \rfloor -1} b_{\lfloor \log N \rfloor -1} + \dots + 2^0 b_0$

ซึ่งทำให้ได้ว่า $A^N = A^{2^{\lfloor \log N \rfloor} b_{\lfloor \log N \rfloor}} A^{2^{\lfloor \log N \rfloor -1} b_{\lfloor \log N \rfloor -1}} \dots A^{2^0 b_0}$

เช่นถ้า $N = 13 = 1101_2$ ก็จะได้ $A^{13} = A^{2^3 \times 1} A^{2^2 \times 1} A^{2^1 \times 0} A^{2^0 \times 1} = A^8 A^4 A^1$

สังเกตว่า A^{2^i} คำนวณได้จาก $A^{2^i} = A^{2^{i-1}} A^{2^{i-1}}$ ดังนั้นการคำนวณ A^{2^i} สำหรับทุก i ตั้งแต่ 1 ถึง $\lfloor \log N \rfloor$ จะใช้เวลา $\mathcal{O}(r^3 \log N)$ ดังนั้นการคำนวณ A^N เพียงต้องเลือกเฉพาะค่า i ที่ $b_i = 1$ ใน Binary Represenation ของ N มาคูณกัน ซึ่งใช้เวลา $\mathcal{O}(r^3 \log N)$ เช่นกัน

ตัวอย่างการเขียน Matrix Exponentiation

เริ่มด้วย Function MUL(X,Y,R) สำหรับการทำ Matrix Multiplication เพื่อแก้ค่า R ให้เป็น XY

```
void mul(int X[4][4], int Y[4][4], int R[4][4]) {
  int M[4][4] = {};
  for (int r = 0; r < 4; r++)
    for (int c = 0; c < 4; c++)
      for (int k = 0; k < 4; k++)
        M[r][c] += X[r][k] * Y[k][c];

  for (int r = 0; r < 4; r++)
    for (int c = 0; c < 4; c++)
      R[r][c] = M[r][c];
}
```

ในการคำนวณ A^N เราจะไล่ตั้งแต่ $i = 0$ ถึง $i = \lfloor \log N \rfloor$ และหา A^{2^i} โดย สำหรับ $i = 0$ จะเอาค่าจาก A มาโดยตรง ส่วนสำหรับ $i \geq 1$ จำใช้ $A^{2^i} = A^{2^{i-1}} A^{2^{i-1}}$ ดังที่อธิบายไว้ ส่วนการคำนวณ ผลลัพท์จะเริ่มจากตั้ง $C = I$ (เมทริกซ์เอกลักษณ์) และแก้เป็น $C = A^{2^i}$ เมื่อเลขตัวที่ $b_i = 1$ ซึ่งสามารถตรวจสอบโดยใช้ bit operation เพราะ $b_i = 1$ เมื่อ $(1LL << i) \& N \neq 0$ ($(1LL << i)$ คือการ shift 1 มาด้านซ้าย i รอบ ถ้า $\&$ กับ N และได้ค่าที่ไม่ใช่ 0 แสดงว่า bit นี้ใน N เป็น 1)

```
void exp(int A[4][4], long long N, int result[4][4])
  int A_pow_2[65][4][4];
  int C[4][4] = {1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
  for (int b = 0; (1LL << b) <= N; b++) {
    if (b == 0) {
      for (int r = 0; r < 4; r++)
        for (int c = 0; c < 4; c++)
          A_pow_2[b][r][c] = A[r][c];
    } else {
      mul(A_pow_2[b - 1], A_pow_2[b - 1], A_pow_2[b])
    }

    if ((1LL << b) & N)
      mul(A_pow_2[b], C, C);
  }

  for (int r = 0; r < 4; r++)
    for (int c = 0; c < 4; c++)
      result[r][c] = C[r][c];
  return;
}
```

เคส $Q \leq 200000$

สำหรับเคสสุดท้ายสังเกตว่าในแต่ละ Test Case ค่า A, B, C, D, E, F, G, H จะไม่เปลี่ยน ดังนั้นเราสามารถคำนวณ $A^0, A^1, \ldots, A^{\lfloor \log 10^{18} \rfloor}$ ไว้ล่วงหน้า โดยใช้เวลา $\mathcal{O}(r^3 \log N)$

ตอนคำนวณคำตอบแต่ละ N เมื่อให้ $N - 4 = (b_{\lfloor \log N \rfloor} b_{\lfloor \log N \rfloor - 1} \ldots b_0)_2$ เราสามารถคำนวณเป็น $(A^{2^{\lfloor \log N \rfloor}} b_{\lfloor \log N \rfloor} (A^{2^{\lfloor \log N \rfloor - 1}} b_{\lfloor \log N \rfloor - 1} (\ldots (A^{2^0} b_0 \begin{bmatrix} D \\ C \\ B \\ A \end{bmatrix}) \ldots)))$ นั่นคือ ในการคูณจะคูณด้านหลังสุดก่อนเพื่อให้เป็นการคูณ Matrix $r \times r$ กับ Vector $r \times 1$ ซึ่งทำให้เวลาในการคูณเหลือ $\mathcal{O}(r^2)$ แทนที่จะเป็น $\mathcal{O}(r^3)$ ในแต่ละรอบการคูณ

การตอบคำถามสำหรับ N ค่าหนึ่งจึงใช้เวลา $\mathcal{O}(r^2 \log N)$

ดังนั้นเวลาทั้งหมดที่ใช้คือ $\mathcal{O}(r^3 \log N) + \mathcal{O}(Qr^2 \log N)$ ซึ่งเร็วเพียงพอสำหรับข้อนี้

Home Tasks Learn About

PROGRAMMING.IN.TH

โปรแกรมมิ่งอินทีเอช ศูนย์รวมของโจทย์และเนื้อหาสำหรับ การเขียนโปรแกรมเพื่อการแข่งขัน และวิทยาการคอมพิวเตอร์

ค้นหาโจทย์

© 2019-2023 the PROGRAMMING.IN.TH team
We are open source on GitHub
สามารถใช้งานเว็บเก่าได้ที่ legacy.programming.in.th

System

Powered by Vercel