Chapter 52: Matrix Exponentiation

Section 52.1: Matrix Exponentiation to Solve Example Problems

Find f(n): nth Fibonacci number. The problem is quite easy when **n** is relatively small. We can use simple recursion, `f(n) = f(n-1) + f(n-2)`, or we can use dynamic programming approach to avoid the calculation of same function over and over again. But what will you do if the problem says, **Given 0 < n < 10^9, find f(n) mod 999983?** Dynamic programming will fail, so how do we tackle this problem?

First let's see how matrix exponentiation can help to represent recursive relation.

Prerequisites:

- Given two matrices, know how to find their product. Further, given the product matrix of two matrices, and one of them, know how to find the other matrix.
- Given a matrix of size **d X d**, know how to find its nth power in **O(d3log(n))**.

Patterns:

At first we need a recursive relation and we want to find a matrix **M** which can lead us to the desired state from a set of already known states. Let's assume that, we know the **k** states of a given recurrence relation and we want to find the **(k+1)th** state. Let **M** be a **k X k** matrix, and we build a matrix **A:[k X 1]** from the known states of the recurrence relation, now we want to get a matrix **B:[k X 1]** which will represent the set of next states, i. e. **M X A = B** as shown below:

```
       |  f(n)  |     | f(n+1) |
       | f(n-1) |     |  f(n)  |
   M X | f(n-2) |  =  | f(n-1) |
       | ...... |     | ...... |
       | f(n-k) |     |f(n-k+1)|
```

So, if we can design **M** accordingly, our job will be done! The matrix will then be used to represent the recurrence relation.

Type 1:
Let's start with the simplest one, `f(n) = f(n-1) + f(n-2)`
We get, `f(n+1) = f(n) + f(n-1)`.
Let's assume, we know `f(n)` and `f(n-1)`; We want to find out `f(n+1)`.
From the situation stated above, matrix **A** and matrix **B** can be formed as shown below:

```
 Matrix A          Matrix B

|  f(n)  |         | f(n+1) |
| f(n-1) |         |  f(n)  |
```

[Note: Matrix **A** will be always designed in such a way that, every state on which `f(n+1)` depends, will be present]
Now, we need to design a **2X2** matrix **M** such that, it satisfies **M X A = B** as stated above.
The first element of **B** is `f(n+1)` which is actually `f(n) + f(n-1)`. To get this, from matrix **A**, we need, **1 X f(n)** and **1 X f(n-1)**. So the first row of **M** will be **[1 1]**.

```
| 1   1 |  X  |  f(n)  |  =  | f(n+1) |
| ----- |     | f(n-1) |     | ------ |
```

[Note: ----- means we are not concerned about this value.]
Similarly, 2nd item of **B** is `f(n)` which can be got by simply taking **1 X f(n)** from **A**, so the 2nd row of **M** is [1 0].

```
| ----- |   X   |  f(n)   |   =   | ------ |
| 1   0 |       | f(n-1)  |       |  f(n)  |
```

Then we get our desired **2 X 2** matrix **M**.

```
| 1   1 |   X   |  f(n)   |   =   | f(n+1) |
| 1   0 |       | f(n-1)  |       |  f(n)  |
```

These matrices are simply derived using matrix multiplication.

Type 2:

Let's make it a little complex: find `f(n) = a X f(n-1) + b X f(n-2)`, where **a** and **b** are constants.
This tells us, `f(n+1) = a X f(n) + b X f(n-1)`.
By this far, this should be clear that the dimension of the matrices will be equal to the number of dependencies, i.e. in this particular example, again 2. So for **A** and **B**, we can build two matrices of size **2 X 1**:

```
Matrix A                Matrix B
|  f(n)  |             | f(n+1) |
| f(n-1) |             |  f(n)  |
```

Now for `f(n+1) = a X f(n) + b X f(n-1)`, we need [a, b] in the first row of objective matrix **M**. And for the 2nd item in **B**, i.e. `f(n)` we already have that in matrix **A**, so we just take that, which leads, the 2nd row of the matrix M to [1 0]. This time we get:

```
| a   b |   X   |  f(n)   |   =   | f(n+1) |
| 1   0 |       | f(n-1)  |       |  f(n)  |
```

Pretty simple, eh?

Type 3:

If you've survived through to this stage, you've grown much older, now let's face a bit complex relation: find `f(n) = a X f(n-1) + c X f(n-3)`?
Ooops! A few minutes ago, all we saw were contiguous states, but here, the state **f(n-2)** is missing. Now?

Actually this is not a problem anymore, we can convert the relation as follows: `f(n) = a X f(n-1) + 0 X f(n-2) + c X f(n-3)`, deducing `f(n+1) = a X f(n) + 0 X f(n-1) + c X f(n-2)`. Now, we see that, this is actually a form described in Type 2. So here the objective matrix **M** will be **3 X 3**, and the elements are:

```
| a 0 c |       |  f(n)   |       | f(n+1) |
| 1 0 0 |   X   | f(n-1)  |   =   |  f(n)  |
| 0 1 0 |       | f(n-2)  |       | f(n-1) |
```

These are calculated in the same way as type 2, if you find it difficult, try it on pen and paper.

Type 4:

Life is getting complex as hell, and Mr, Problem now asks you to find `f(n) = f(n-1) + f(n-2) + c` where **c** is any constant.
Now this is a new one and all we have seen in past, after the multiplication, each state in **A** transforms to its next

state in **B**.

```
f(n) = f(n-1) + f(n-2) + c
f(n+1) = f(n) + f(n-1) + c
f(n+2) = f(n+1) + f(n) + c
.................... so on
```

So , normally we can't get it through previous fashion, but how about we add **c** as a state:

```
      |  f(n)   |   | f(n+1) |
  M X | f(n-1)  | = |  f(n)  |
      |    c    |   |    c   |
```

Now, its not much hard to design **M**. Here's how its done, but don't forget to verify:

```
  | 1 1 1 |     |  f(n)  |     | f(n+1) |
  | 1 0 0 |  X  | f(n-1) |  =  |  f(n)  |
  | 0 0 1 |     |    c   |     |    c   |
```

Type 5:

Let's put it altogether: find `f(n) = a X f(n-1) + c X f(n-3) + d X f(n-4) + e`. Let's leave it as an exercise for you. First try to find out the states and matrix **M**. And check if it matches with your solution. Also find matrix **A** and **B**.

```
| a 0 c d 1 |
| 1 0 0 0 0 |
| 0 1 0 0 0 |
| 0 0 1 0 0 |
| 0 0 0 0 1 |
```

Type 6:

Sometimes the recurrence is given like this:

```
f(n) = f(n-1)   -> if n is odd
f(n) = f(n-2)   -> if n is even
```

In short:

```
f(n) = (n&1) X f(n-1) + (!(n&1)) X f(n-2)
```

Here, we can split the functions in the basis of odd even and keep 2 different matrix for both of them and calculate them separately.

Type 7:

Feeling little too confident? Good for you. Sometimes we may need to maintain more than one recurrence, where they are interested. For example, let a recurrence re;atopm be:

```
g(n) = 2g(n-1) + 2g(n-2) + f(n)
```

Here, recurrence *g(n)* is dependent upon `f(n)` and this can be calculated in the same matrix but of increased dimensions. From these let's at first design the matrices **A** and **B**.

```
 Matrix A            Matrix B
|  g(n)  |          | g(n+1) |
| g(n-1) |          |  g(n)  |
| f(n+1) |          | f(n+2) |
|  f(n)  |          | f(n+1) |
```

Here, `g(n+1) = 2g(n-1) + f(n+1) and f(n+2) = 2f(n+1) + 2f(n)`. Now, using the processes stated above, we can find the objective matrix **M** to be:

```
| 2 2 1 0 |
| 1 0 0 0 |
| 0 0 2 2 |
| 0 0 1 0 |
```

So, these are the basic categories of recurrence relations which are used to solveby this simple technique.