Fibonacci Finding (easy)

Fibonacci Numbers

Fibonacci Numbers are

$$0, 1, 1, 2, 3, 5, 8, 13 \ldots$$

Fibonacci numbers are generated using the following recurrence relation

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ &\vdots \\ F_i &= F_{i-1} + F_{i-2} \; for \; i \geq 2 \end{aligned}$$

It is interesting to note that the n^{th} Fibonacci number grows so fast that F_{47} exceeds the 32-bit signed integer range.

The fastest way to accurately compute Fibonacci numbers is by using a matrix-exponentiation method.

$$\begin{pmatrix} F_{k+2} \\ F_{k+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{k+1} \\ F_k \end{pmatrix}$$

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

We need to calculate M^N to calculate the Nth fibonacci number. We can calculate it in $O(log(N))$ using fast exponentiation.

Simple Recursive Solution

```
fibonacci(n)
    if(n=0)
        return 0
    if (n=1)
        return 1
    return (fibonacci(n-1)+fibonacci(n-2))
```

Time complexity:
$T(n) = T(n-1) + T(n-2)$
$T(n) = O(2^n)$

Optimization using Dynamic Programming

There are only n overlapping subproblems which can be stored to reduce the time complexity to $O(n)$

Go to Top

```
//Initialize all elements in dp to -1
fibonacci(n)
    if(dp[n]!=-1)
        return dp[n]
    if(n=0)
        dp[n]=0
    else if (n=1)
        dp[n]=1
    else
        dp[n]=fibonacci(n-1)+fibonacci(n-2)
    return dp[n]
```

Time Complexity = $O(n)$
Space Complexity = $O(n)$

Using Matrix Exponentiation

```
//Calculating A^p in O(log(P))
Matrix_pow ( Matrix A,int p )
    if(p=1)
        return A
    if(p%2=1)
        return A*Matrix_pow(A,p-1)
    Matrix B = Matrix_pow(a,p/2);
    return B * B

fibonacci(n)
    if(n=0)
        return 0;
    if(n=1)
        return 1;
    Matrix M[2][2]={{1,1},{1,0}}
    Matrix res=matrix_pow(M,n-1);
    return res[0][0];
```

Time Complexity = $O(log(n))$

Related challenge for **Fibonacci Numbers**

Is Fibo

Success Rate: 78.62% **Max Score: 20** **Difficulty:**

Solve Challenge