Now we know that fib(0) = 0 and fib(1) = 1 we can substitute these values in the above tree diagram , then fib(2) = fib(1) + fib(0) = 0 + 1 = 1.
Now fib(3) = fib(2) + fib(1) = 1 + 1 = 2
fib(4) = fib(3) + fib(2) = 2 + 1 = 3
fib(5) = fib(4) + fib(3) = 3 + 2 = 5

Q3: given two integers a and b. Find the value of a^b using recursion.
Input 1: a = 3 , b = 4
Output 1: 81

Solution :
Code: LP_Code3.java

Output:

```
Required answer is 81
```

Approach :

- Create a recursive function say power(int a, int b)
- Base condition : if(b == 0) return 1 because we know anything to the base 0 = 1.
- Otherwise, return (a * power(a , b - 1))

Efficient approach:
Lets see a better approach on how we can improve the time complexity of our solution, lets define a function func which will take two numbers a and b as input and will return a^b. Now this function can we written in this way func(a, b) =func(a, b / 2) ×func(a, b / 2); Like humans do in real life for example a=2, b=4, so for calculating 2^4 = we can find (2^2), take square of it in case b is even. In case b is odd, for example a=2, b=5, so for calculating 2^5 = we can find (2^2), take square of it and multiply it with 2 like (2^5)= (2) * (2^2) * (2^2).

power(a, b) =power(a , b / 2) ×power(a , b / 2); // if b is even

func(a, b) = a ×power(a , (b-1) / 2) ×power(a , (b-1) / 2); // if b is odd

The generated recursion tree when above recurrence relation is implemented to calculate power(3 , 4) is shown below.

As you can observe, func(2,2,), func(2,1) are being computed multiple times. To avoid these redundant computations, we store the result in a result variable while making the recursive call and use this variable value subsequently.

Code : LP_Code4.java

Output :

```
Required answer is 81
```

Approach :

- This function has a base case: b = 0.
- This function has two recursive calls. Only one of them is made in any given call.
- Let's first consider the case when b is even: In that case, the recursive call is made with b / 2. I will store the value return by power(a , b / 2) in result variable and will return square of result variable i.e (result * result).