Memoization

CS 115

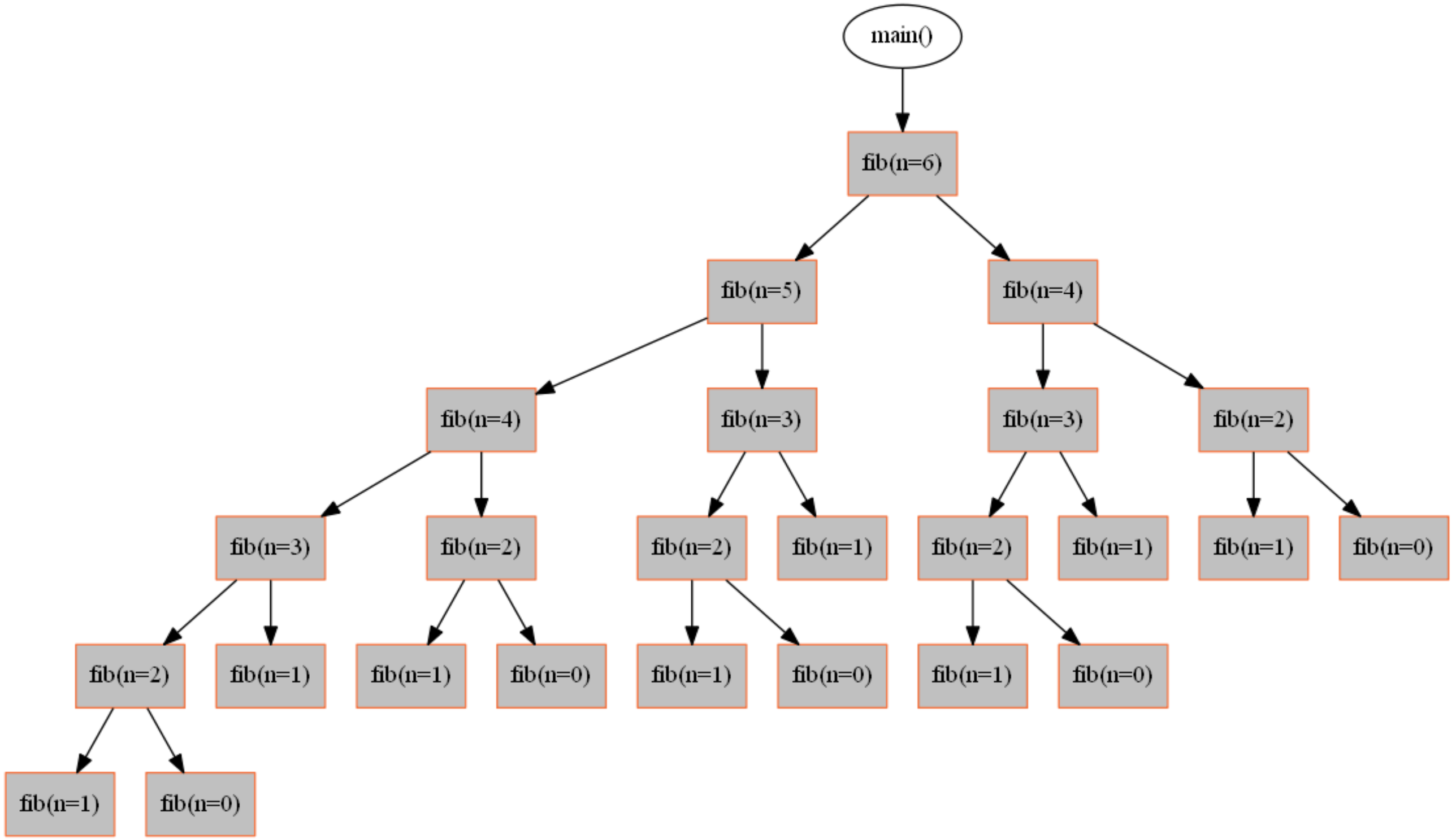
Recursion

Recursion is a programming technique where a function calls itself repeatedly till one/more termination conditions are met.

fastFibonacci.py

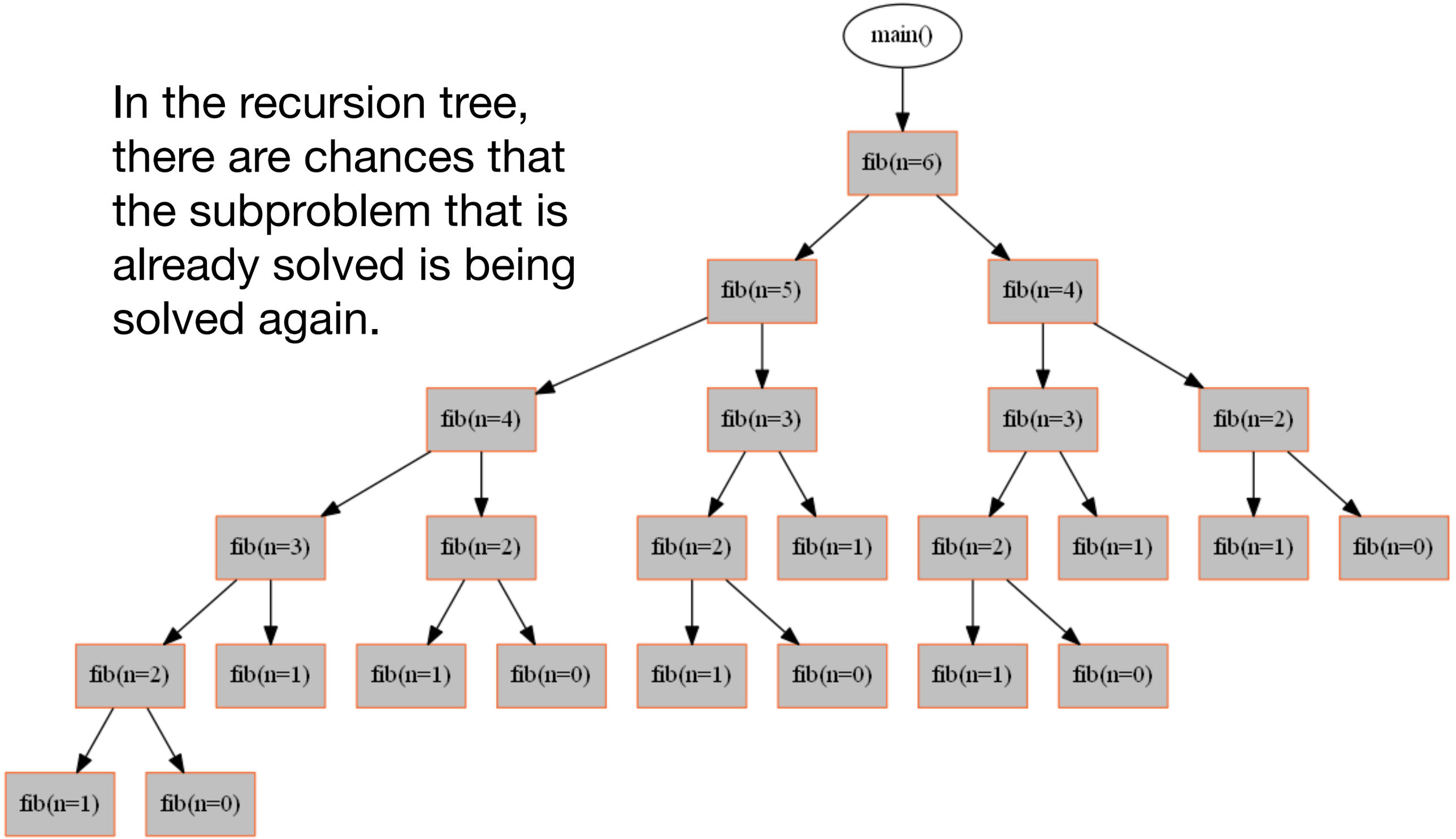
```
def Fibonacci(n):
    if n < 0:
        return "Incorrect input"
    elif n == 0:
        return 0
    elif n == 1 or n == 2:
        return 1
    else:
        return Fibonacci(n-1) + Fibonacci(n-2)
```

Problem


main()
fib(n=6)
fib(n=5)
fib(n=4)
fib(n=4)
fib(n=3)
fib(n=3)
fib(n=2)
fib(n=3)
fib(n=2)
fib(n=2)
fib(n=1)
fib(n=2)
fib(n=1)
fib(n=1)
fib(n=0)
fib(n=1)
fib(n=0)
fib(n=1)
fib(n=0)
fib(n=2)
fib(n=1)
fib(n=1)
fib(n=0)
fib(n=1)
fib(n=0)

Problem

In the recursion tree, there are chances that the subproblem that is already solved is being solved again.



Memoization

- Memoization is a technique of recording the intermediate results so that they can be used to avoid repeated calculation and speed up the programs
- Goal: Optimize recursive solutions.
- Dictionaries can help us here.

```
memo = {}
def fastFibonacci(n):
    if n in memo:
        return memo[n]
    if n < 0:
        return "Incorrect input"
    elif n == 0:
        memo[n] = 0
        return 0
elif n == 1 or n == 2:
    memo[n] = 1
    return 1
else:
    first_term = fastFibonacci(n-1)
    memo[n - 1] = first_term
    second_term = fastFibonacci(n-2)
    memo[n-2] = second_term
    return first_term + second_term
```