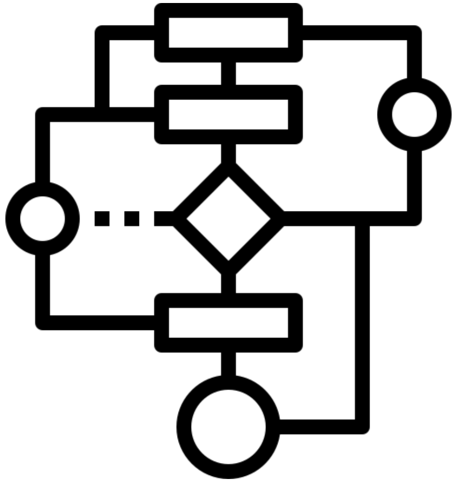


CS 310: Algorithms

Lecture 17

Instructor: Naveed Anwar Bhatti

Administrivia

Assignment 3 will be released today

Quiz 4 - Solution

In the class, we discussed that if weights on edges of a graph ***G*** are not distinct, then ***G*** may have more than one MST's. Make a small graph (**3 vertices**) and show that with different sorted orders of edge weights (depending on how we break ties) Kruskal's algorithm produces different MST's.

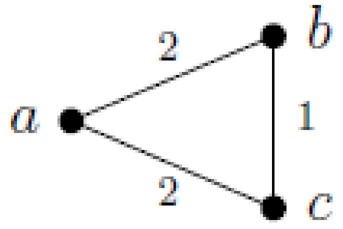
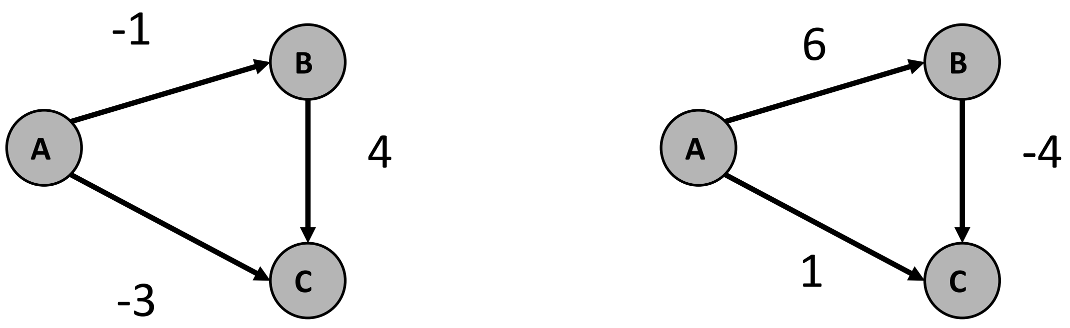


Figure 4: The sorted order $(b, c), (a, b), (a, c)$ produces a MST $\{(b, c), (a, b)\}$, while the sorted order $(b, c), (a, c), (a, b)$ produces a MST $\{(b, c), (a, c)\}$

Quiz 4 - Solution

In the class, we discussed that Dijkstra, in some scenarios, does handle negative edges. Make a small example of directed weighted graph (**3 vertices**), where Dijkstra algorithm does produce correct shortest paths, even though there are negative weights.



Quiz 4 - Solution

Consider the following divide-and-conquer approach to computing MST of a graph $\boldsymbol{G = (V,E,w)}$. Suppose $|\boldsymbol{V}| = \boldsymbol{n} = \mathbf{2}^{\boldsymbol{k}}$ for some integer $\boldsymbol{k}$. We partition $\boldsymbol{V}$ into $\boldsymbol{V_1}$ and $\boldsymbol{V_2}$ such that $|\boldsymbol{V_1}| = |\boldsymbol{V_2}| = \frac{\boldsymbol{n}}{\mathbf{2}}$. Let $\boldsymbol{G_1}$ and $\boldsymbol{G_2}$ be the subgraphs induced by $\boldsymbol{V_1}$ and $\boldsymbol{V_1}$ respectively. We recursively compute a MST's in $\boldsymbol{T_1}$ of $\boldsymbol{G_1}$ and $\boldsymbol{T_2}$ of $\boldsymbol{G_2}$. Now we add the lightest edge $\boldsymbol{e = (u, v)}$ crossing the cut $[\boldsymbol{V_1}, \boldsymbol{V_2}]$ in $\boldsymbol{G}$ and use it to unite $\boldsymbol{T_1}$ and $\boldsymbol{T_2}$. Either prove that $\boldsymbol{T} = \boldsymbol{T_1}$ U $\boldsymbol{T_2}$ U [$\{(\boldsymbol{u}, \boldsymbol{v})\}$ is a MST of $\boldsymbol{G}$ or give a small counter example on **n = 4** vertices on which this algorithm fails.

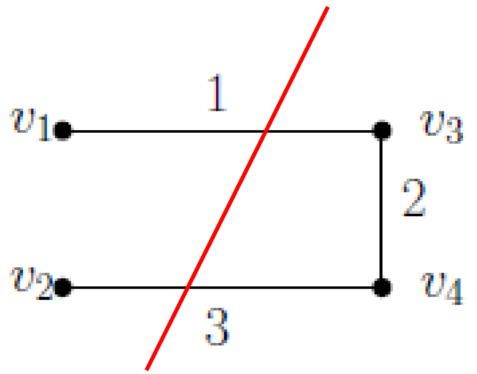
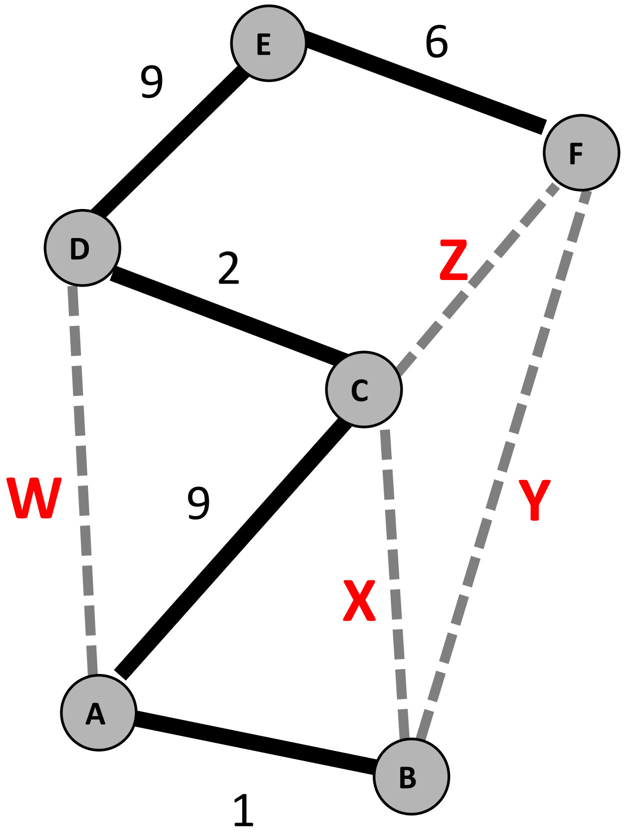
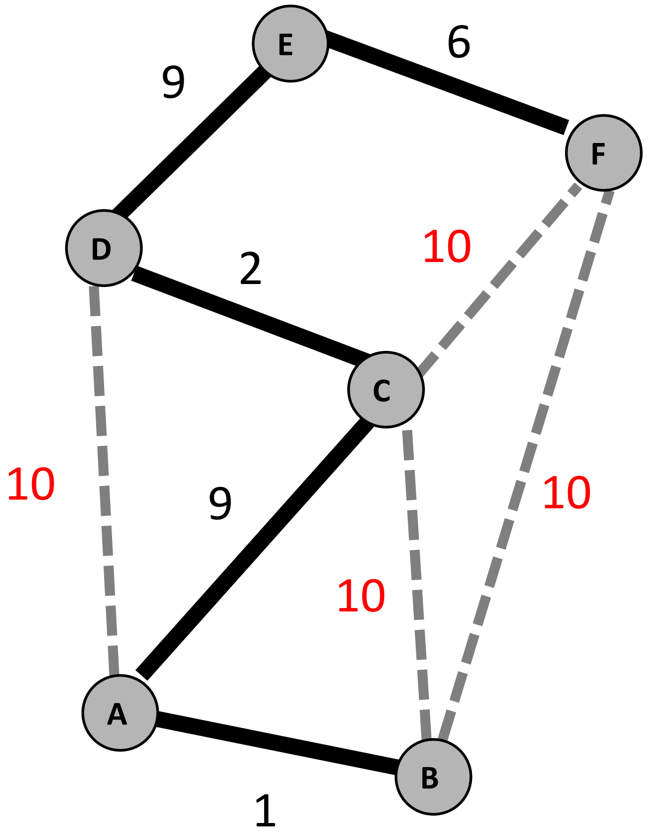


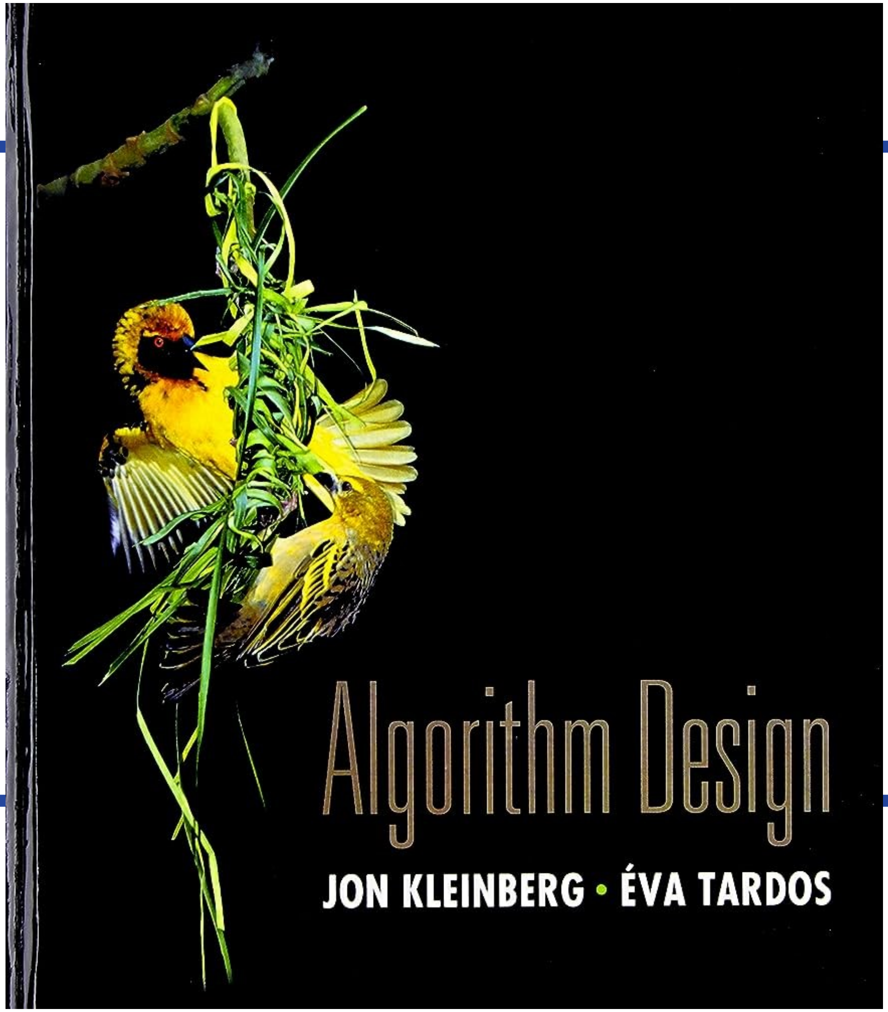
Figure 6: If we partition it as $V_1 = \{v_1, v_2\}$ and $V_2 = \{v_2, v_3\}$, then left part is not even connected, so there is no spanning tree of it, while the whole graph is connected.

Quiz 4 - Solution

In the provided graph, bold lines represent the edges of the minimum spanning tree (MST). Determine the minimum possible values for the non-MST edges labeled **W**, **X**, **Y**, and **Z**.







Chapter 6:
Dynamic Programming

Section :
Dynamic Programming

So far…

Greedy Algorithms

- Solve a problem step by step, picking the **best choice at each step**.
- Only think about what seems best right now, not the whole problem.

Divide and Conquer

- Divide the problem into **smaller subproblems**.
- Solve each small part on its own.
- Put the answers of the small parts together to solve the whole problem.

Dynamic Programming

- Divide the problem into **overlapping subproblems**.
- Solve and store the solution to each subproblem so it doesn't need to be recomputed.
- Build up the solution of the main problem using the stored solutions of the smaller subproblems.

Fibonacci Sequence

$$0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2 \end{cases}$$

Recursive F_n computation

Implementing the recursive definition of F_n

Algorithm Recursive F_n computation

```
function FIB1(n)
    if n = 0 then
        return 0
    else if n = 1 then
        return 1
    else
        return FIB1(n − 1) + FIB1(n − 2)
```

How much time will it take?

Recursive F_n computation

Implementing the recursive definition of F_n

Algorithm Recursive F_n computation

function $\textsc{Fib1}(n)$
  if $n = 0$ **then**
    return 0
  else if $n = 1$ **then**
    return 1
  else
    return $\textsc{Fib1}(n-1) + \textsc{Fib1}(n-2)$

n

(n-1) (n-2)

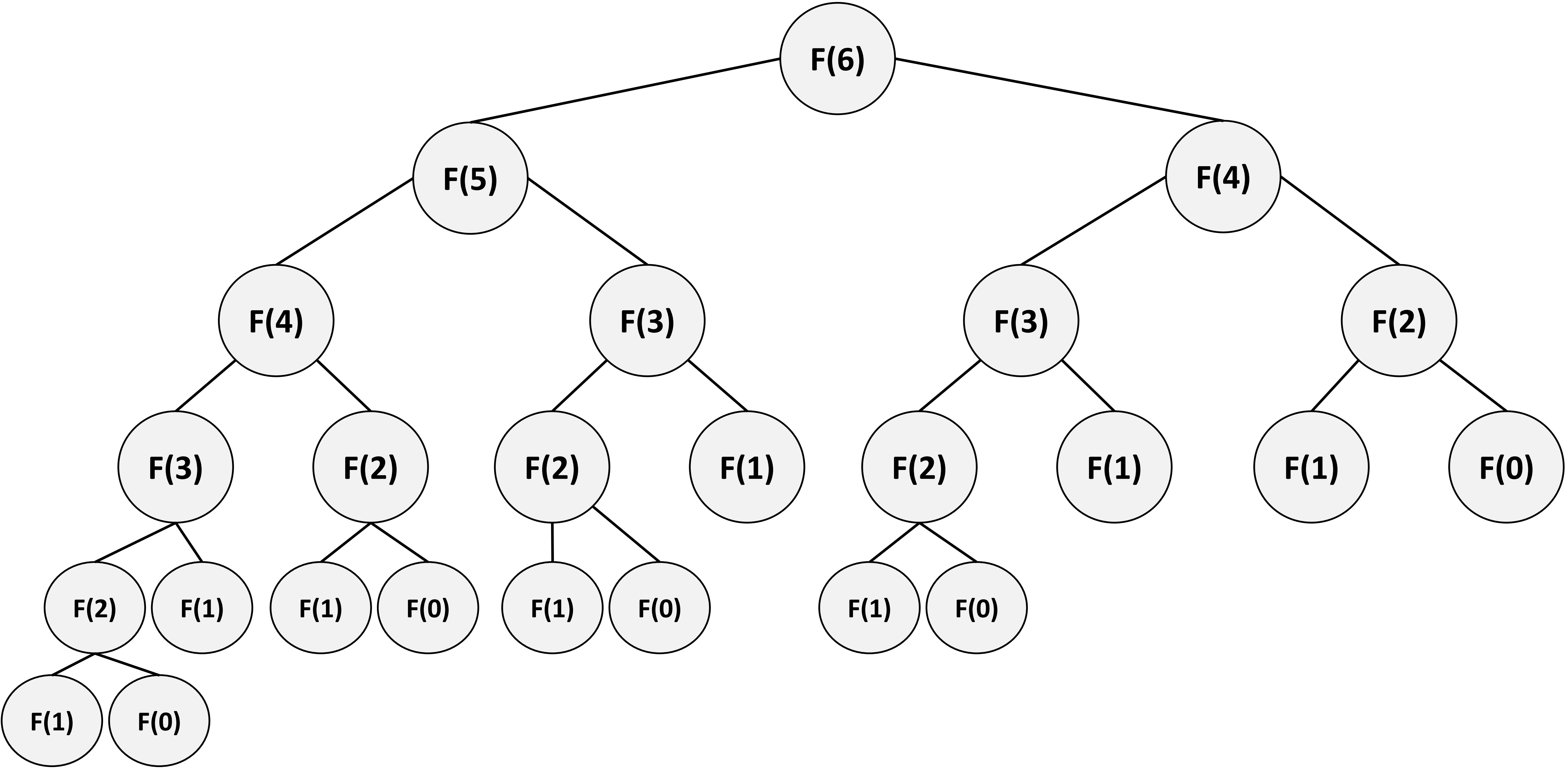
(n-1-1) (n-1-2) (n-2-1) (n-2-2)

How much time will it take?

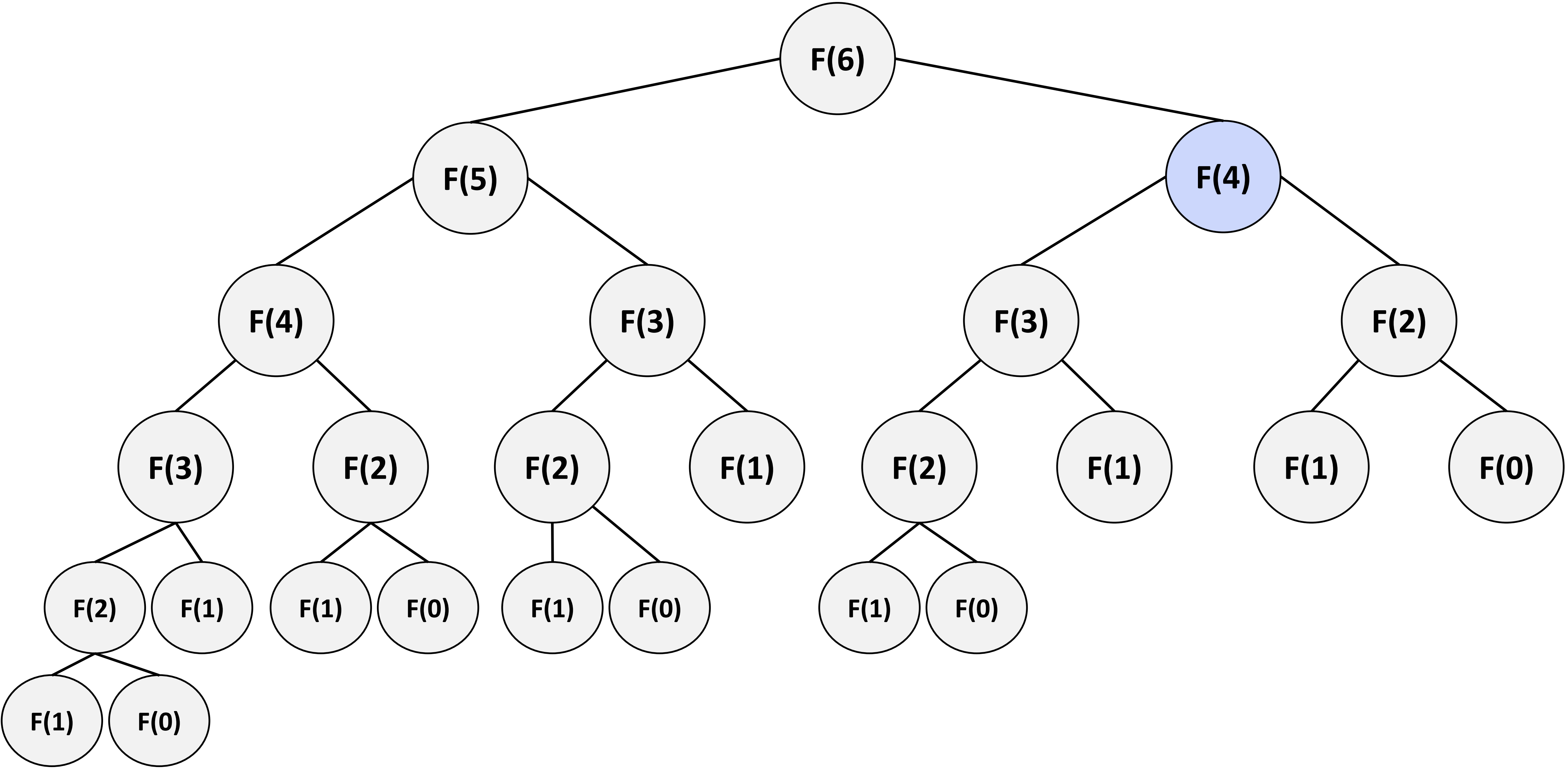
Time Complexity ≈ 2^n

Time Complexity = O(2^n)

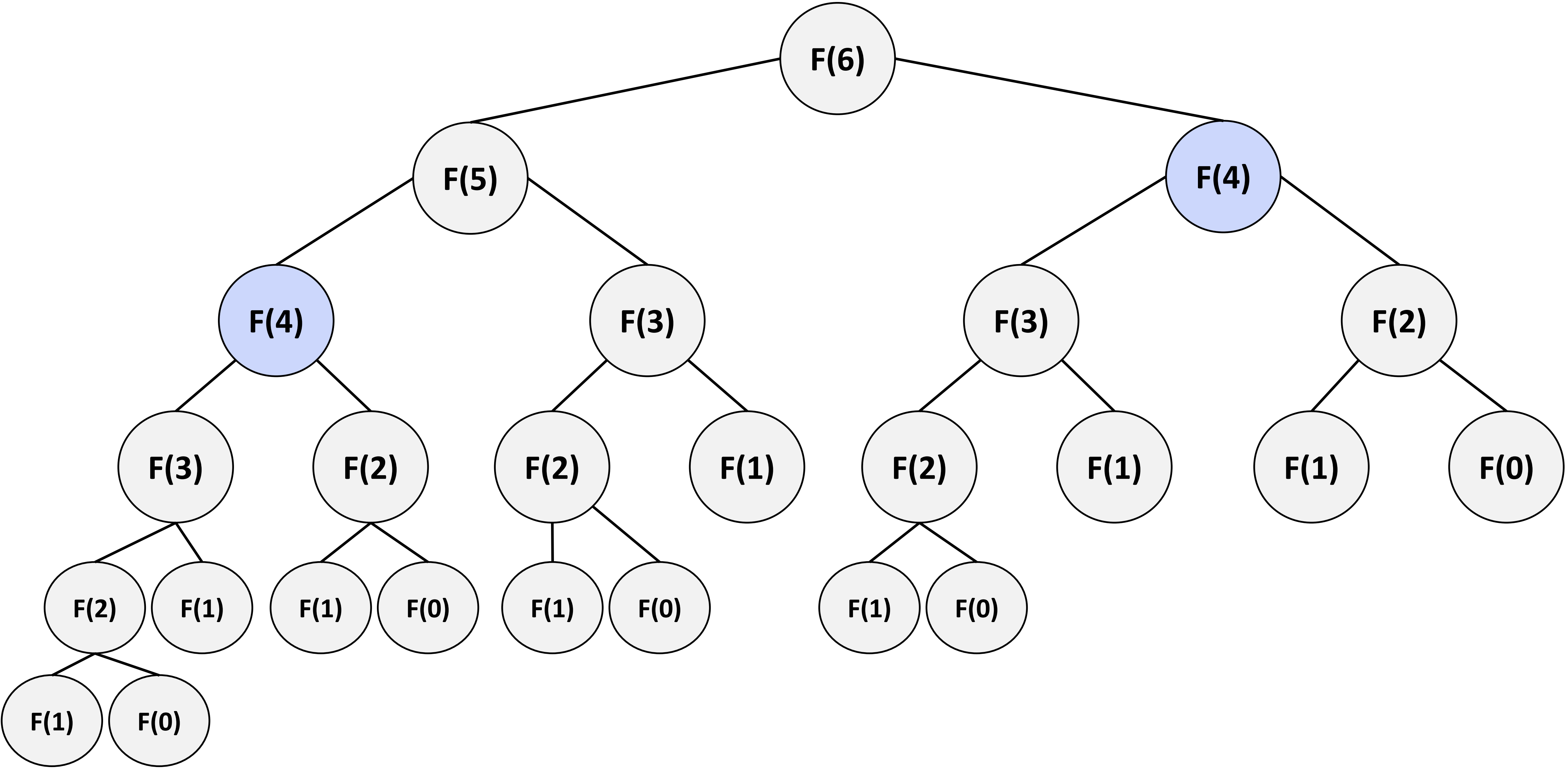
Recursive F_n computation



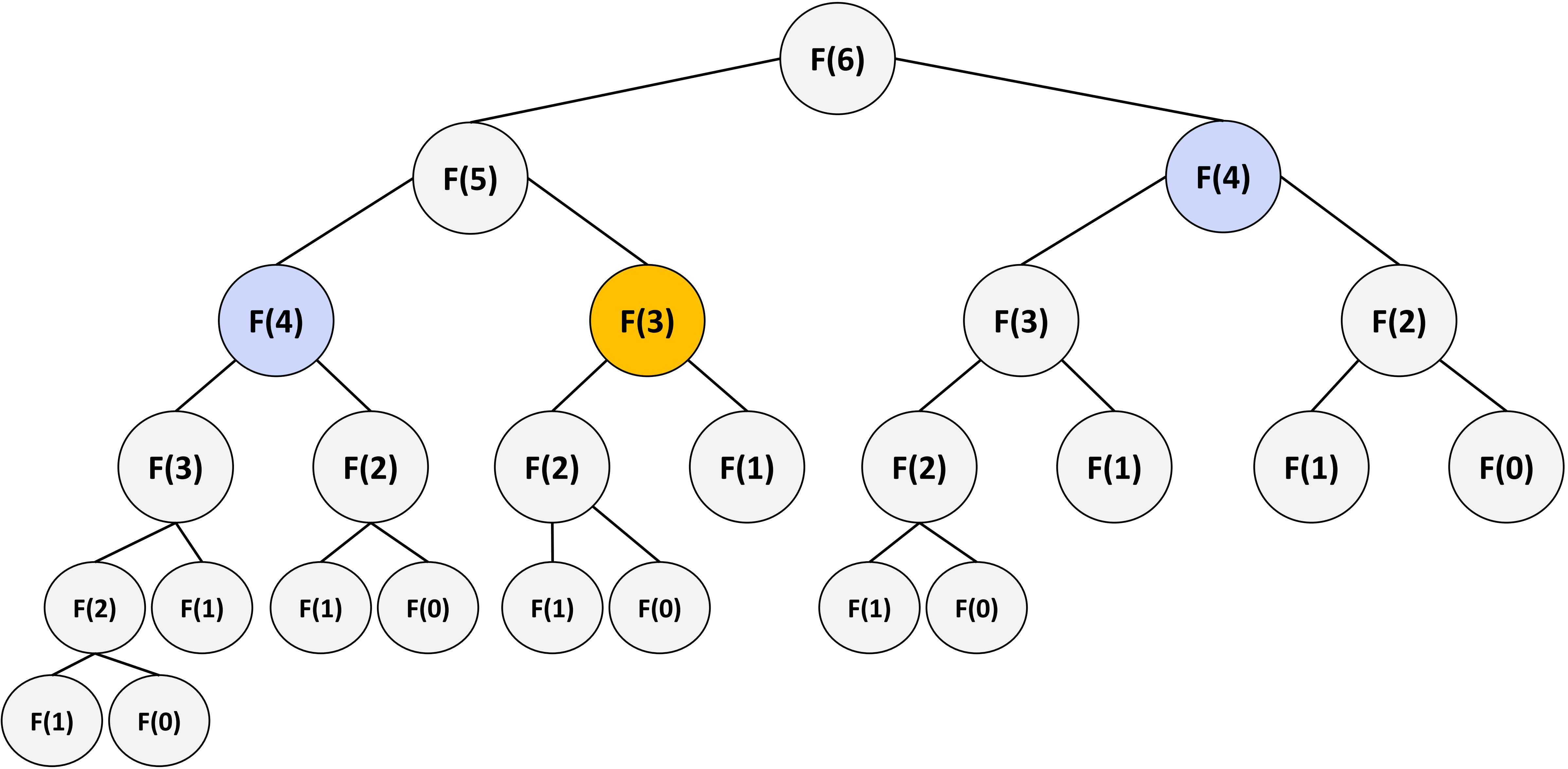
Recursive F_n computation



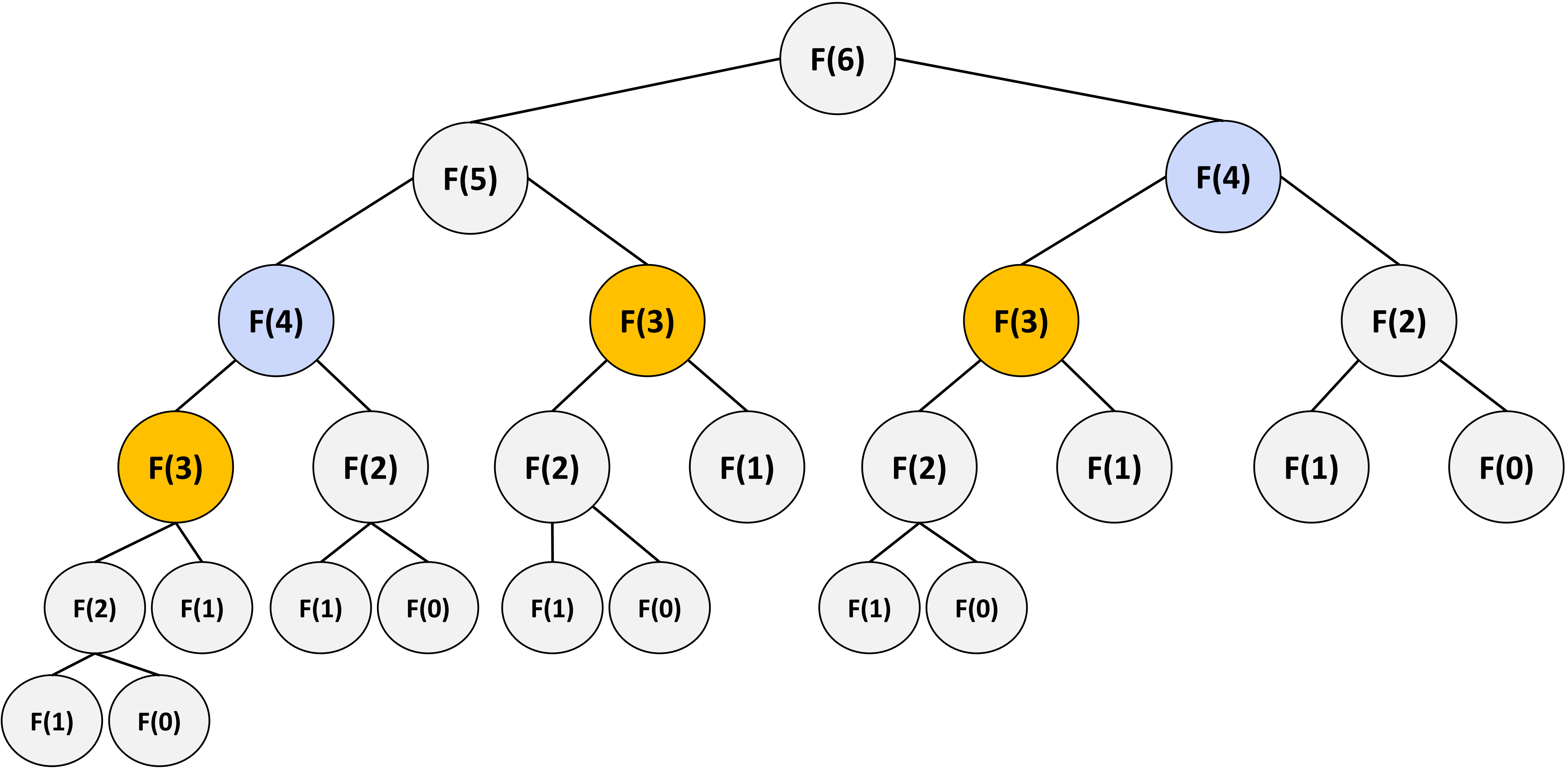
Recursive F_n computation



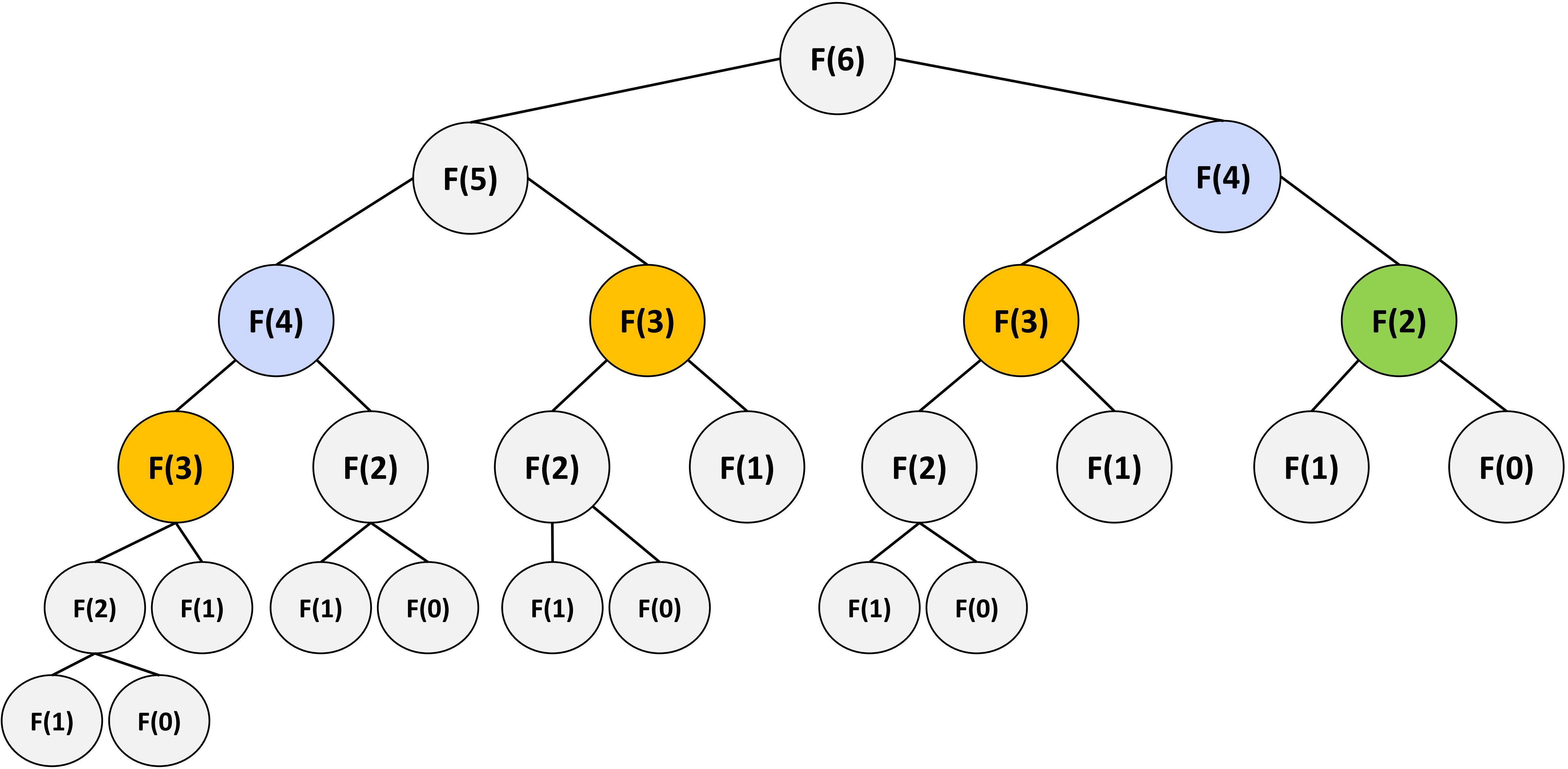
Recursive F_n computation



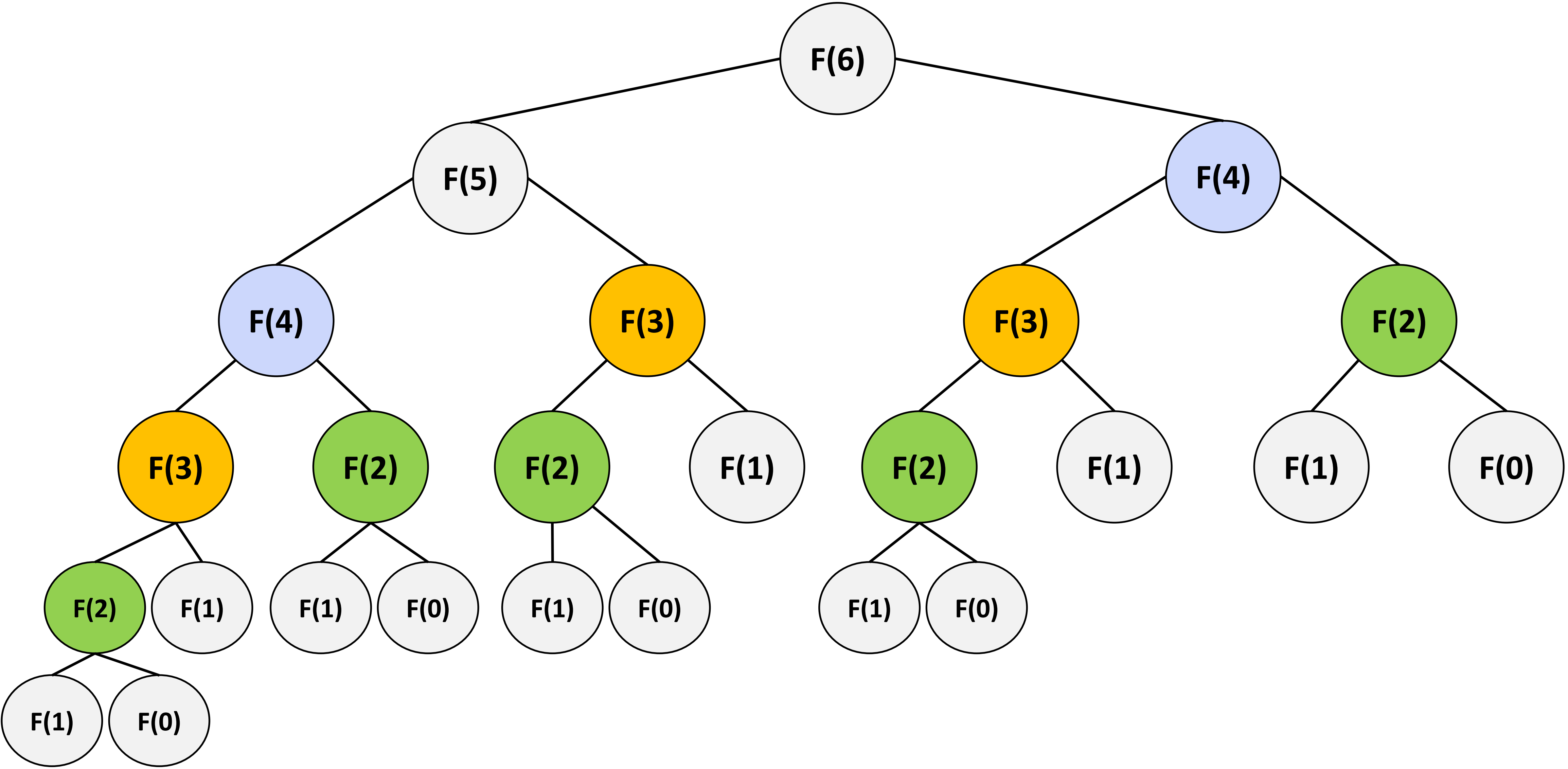
Recursive F_n computation



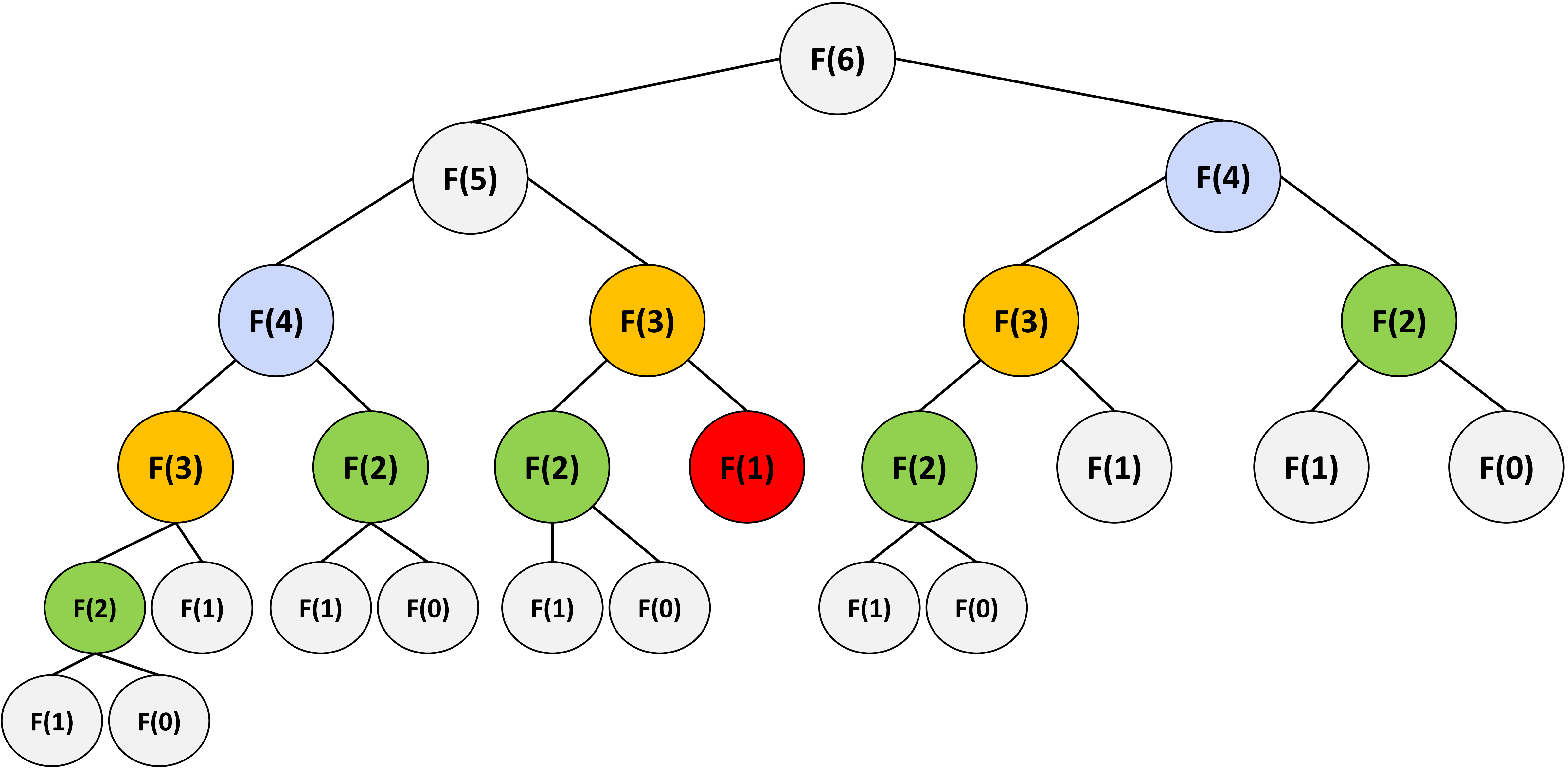
Recursive F_n computation



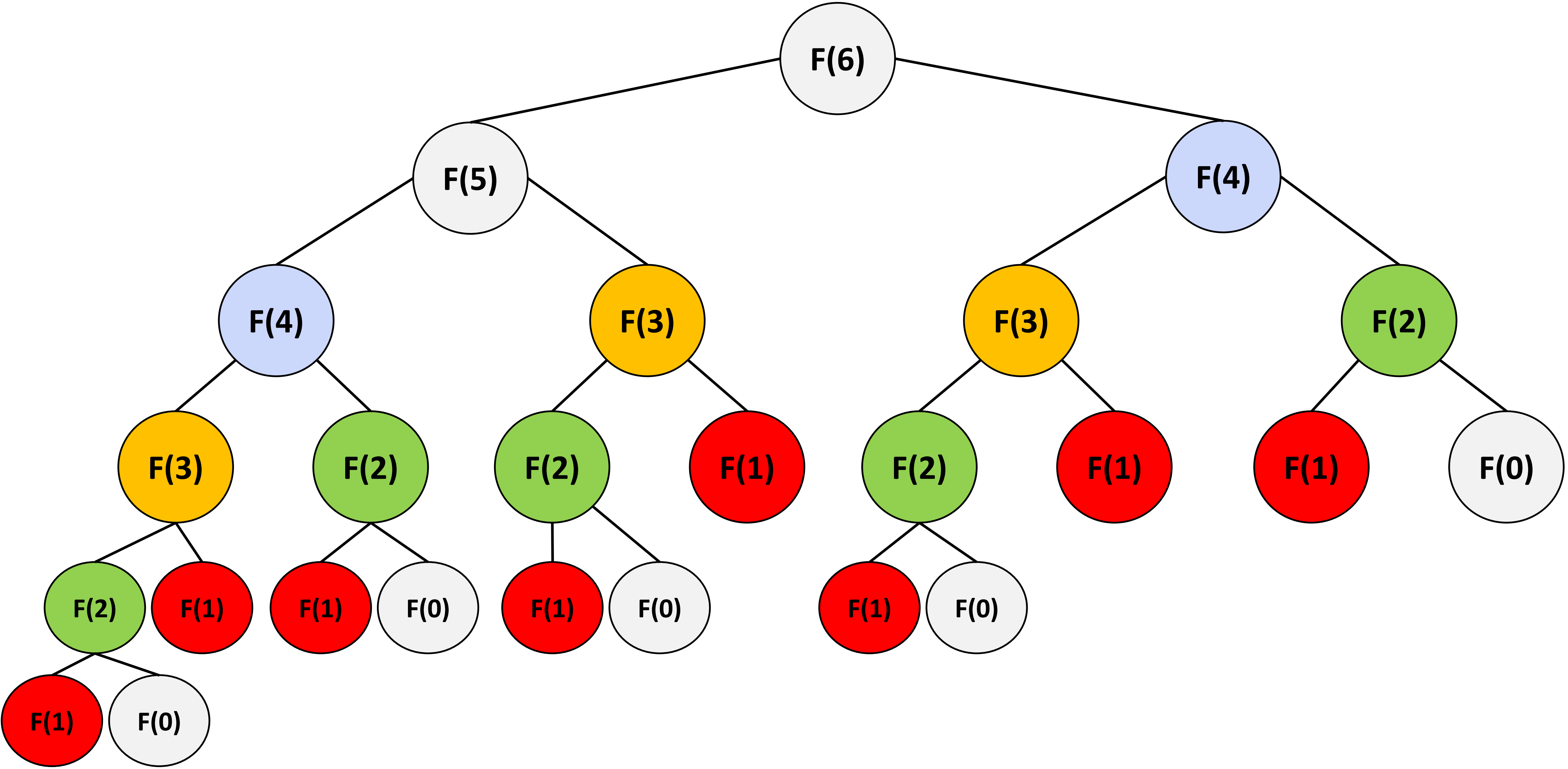
Recursive F_n computation



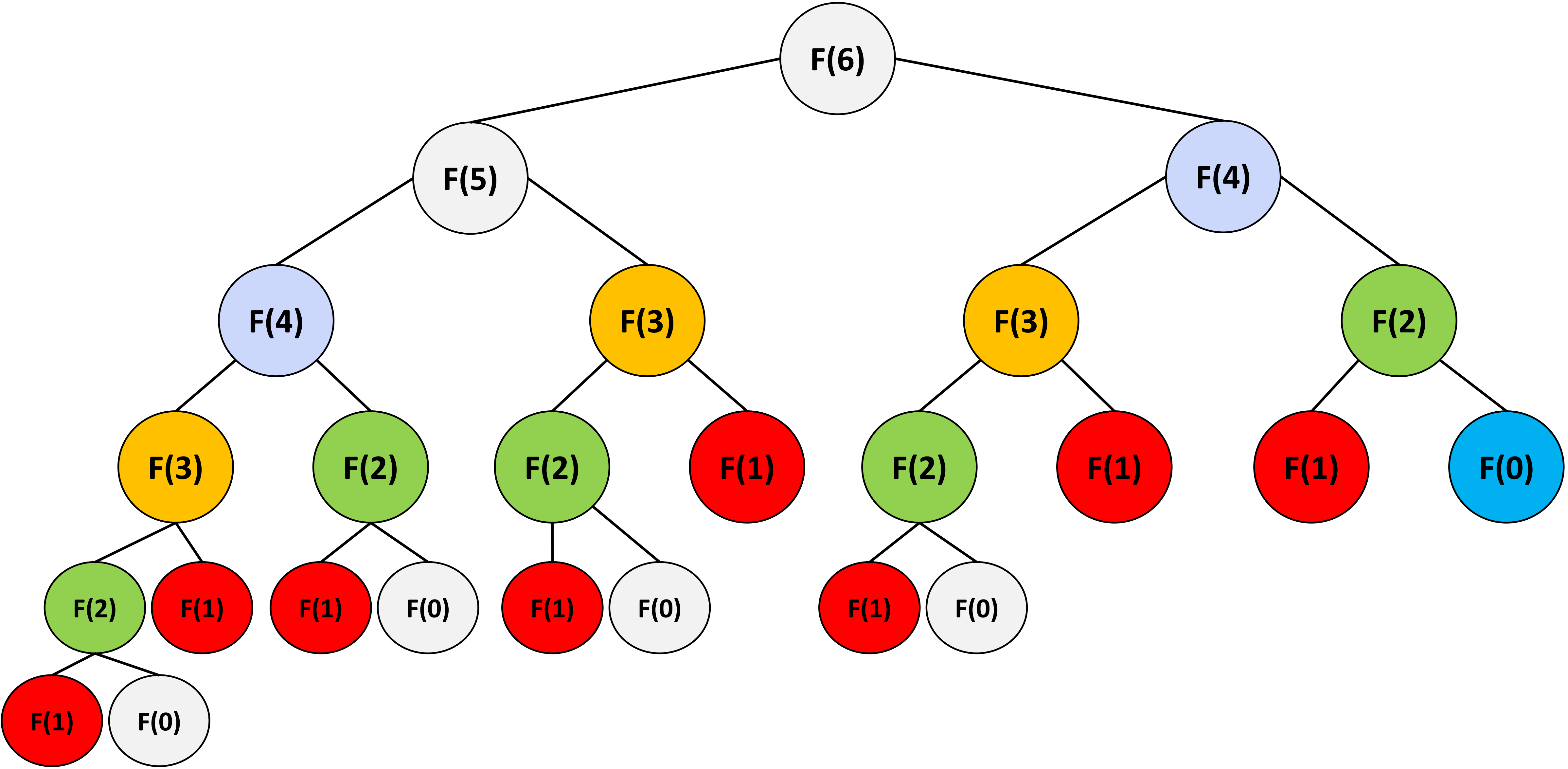
Recursive F_n computation



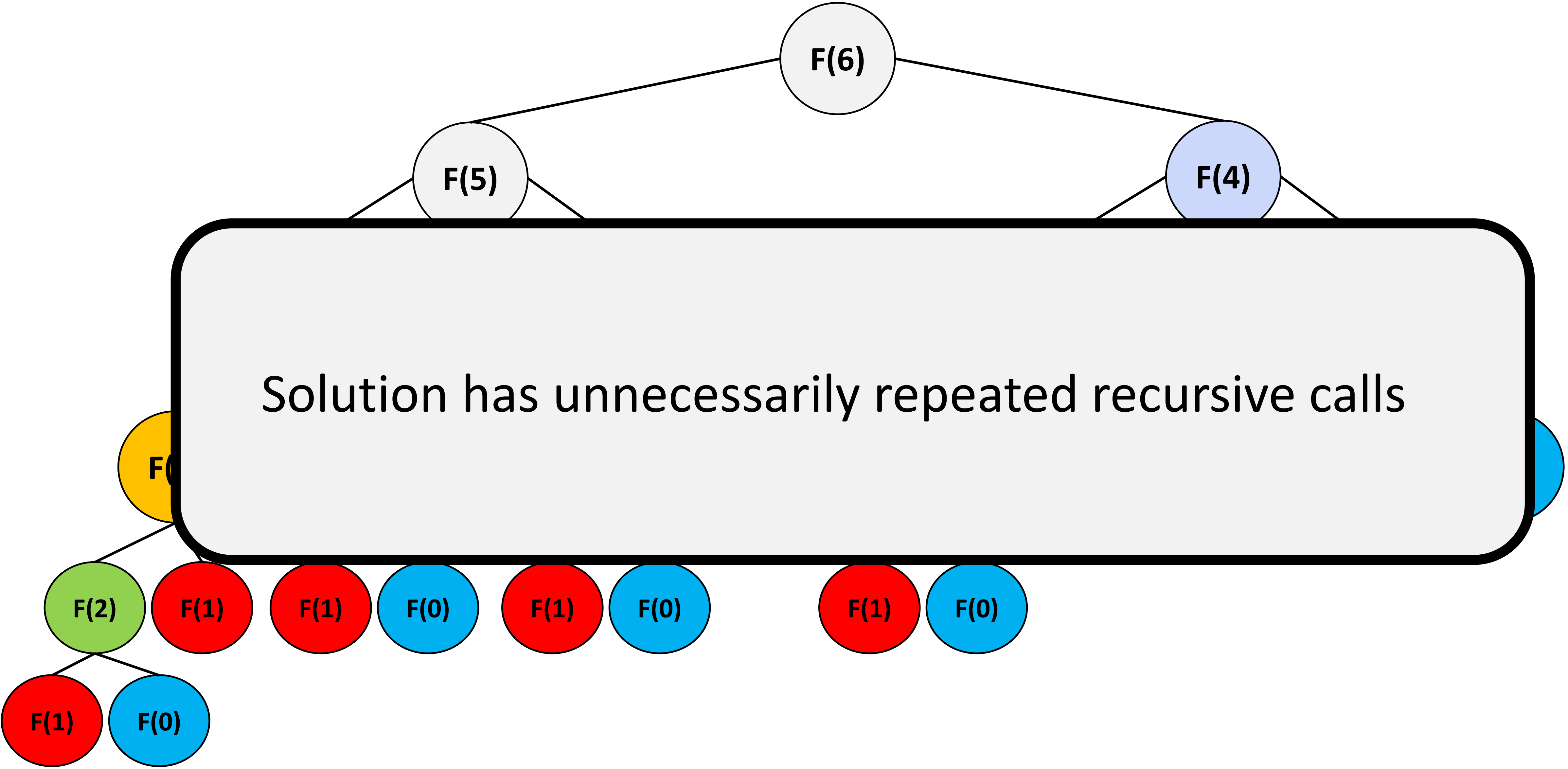
Recursive F_n computation



Recursive F_n computation

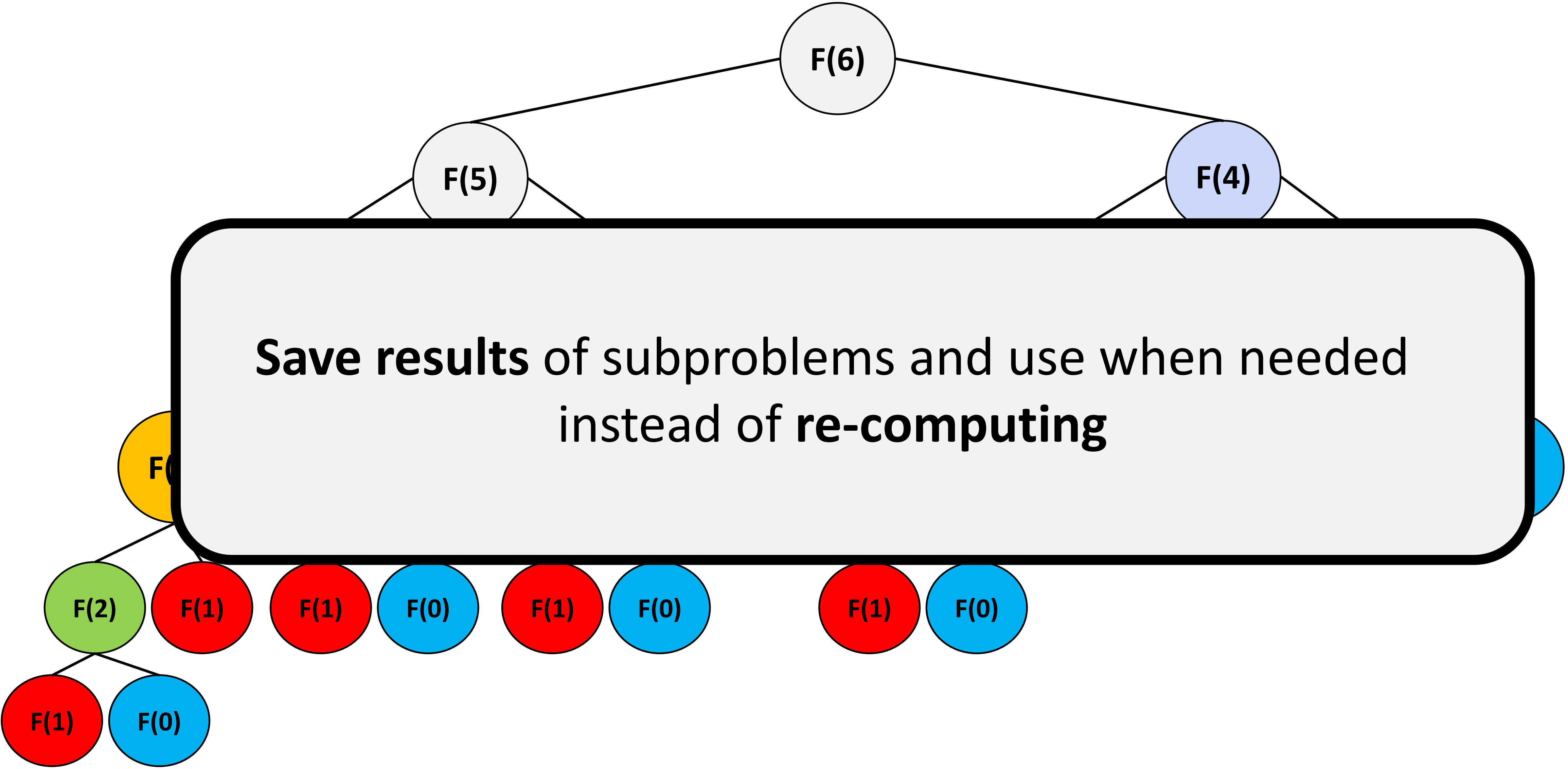


Recursive F_n computation



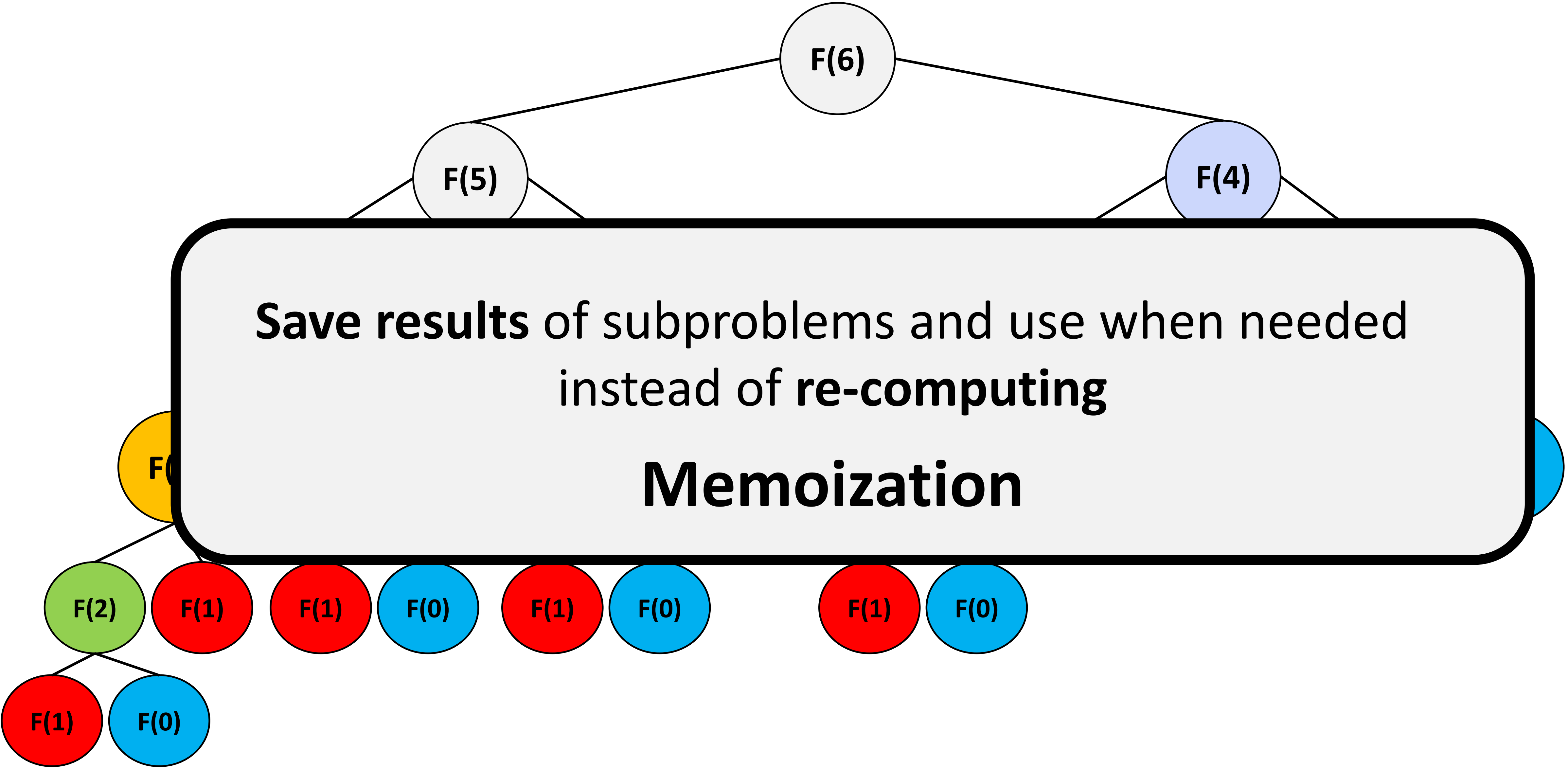
Solution has unnecessarily repeated recursive calls

Recursive F_n computation



Save results of subproblems and use when needed instead of **re-computing**

Recursive F_n computation



Save results of subproblems and use when needed instead of **re-computing**

Memoization

F_n computation with Memoization

Algorithm Recursive F_n computation

function $\textsc{Fib1}(n)$
  if $n = 0$ **then**
    return 0
  else if $n = 1$ **then**
    return 1
  else
    return $\textsc{Fib1}(n-1) + \textsc{Fib1}(n-2)$

F[0]	0
F[1]	1
F[2]	-1
F[3]	-1
F[4]	-1
⋮	⋮
F[n-2]	-1
F[n-1]	-1

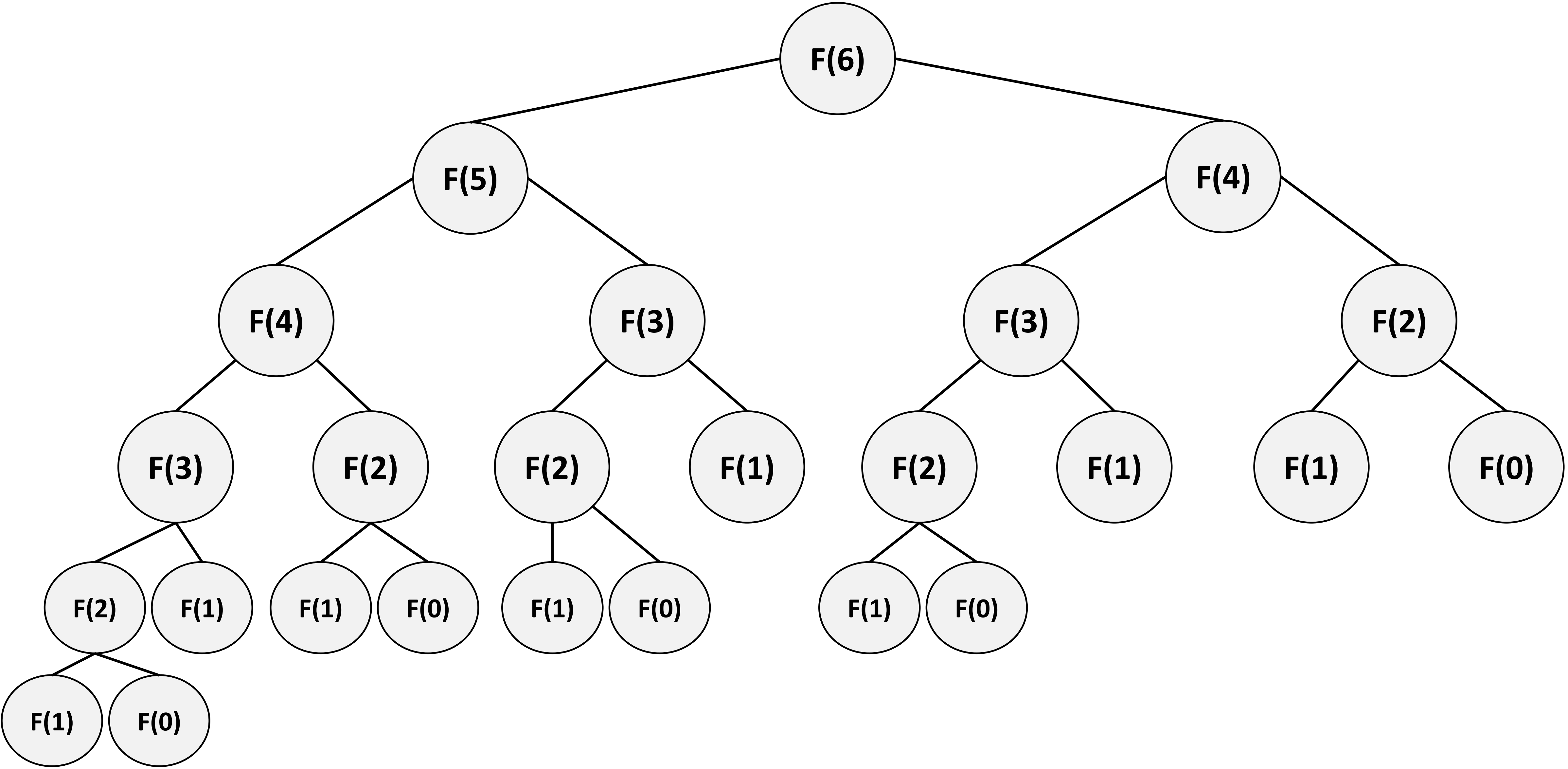
F_n computation with Memoization

Algorithm F_n computation with memoization

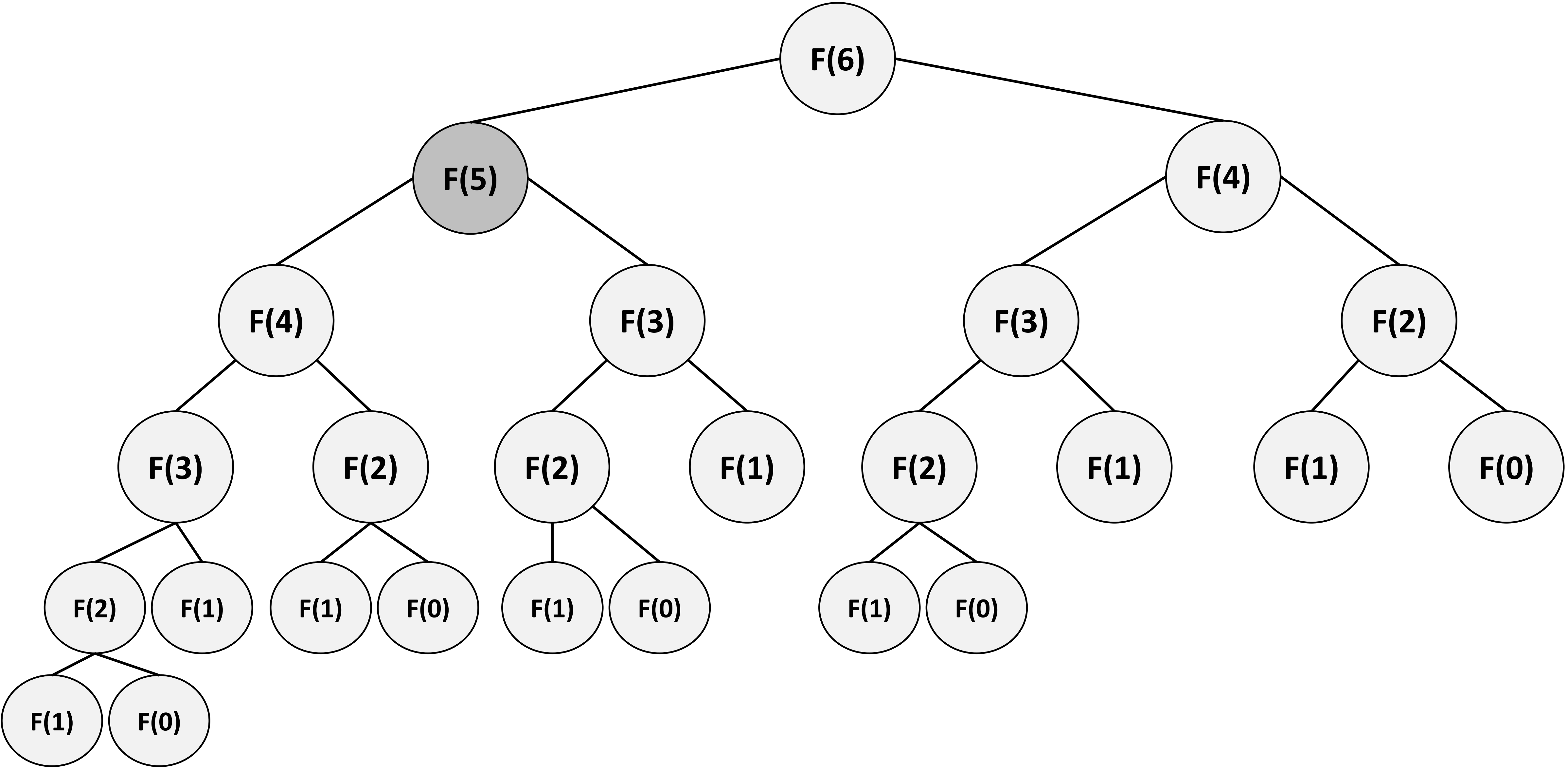
function $\text{FIB2}(n)$
    if $F[n-1] = -1$ **then**
        $F[n-1] \leftarrow \text{FIB2}(n-1)$ $\triangleright$ Call FIB2 function only if $F[n-1] = -1$
    if $F[n-2] = -1$ **then**
        $F[n-2] \leftarrow \text{FIB2}(n-2)$
    return $F[n-1] + F[n-2]$

F[0]	0
F[1]	1
F[2]	-1
F[3]	-1
F[4]	-1
⋮	⋮
F[n-2]	-1
F[n-1]	-1

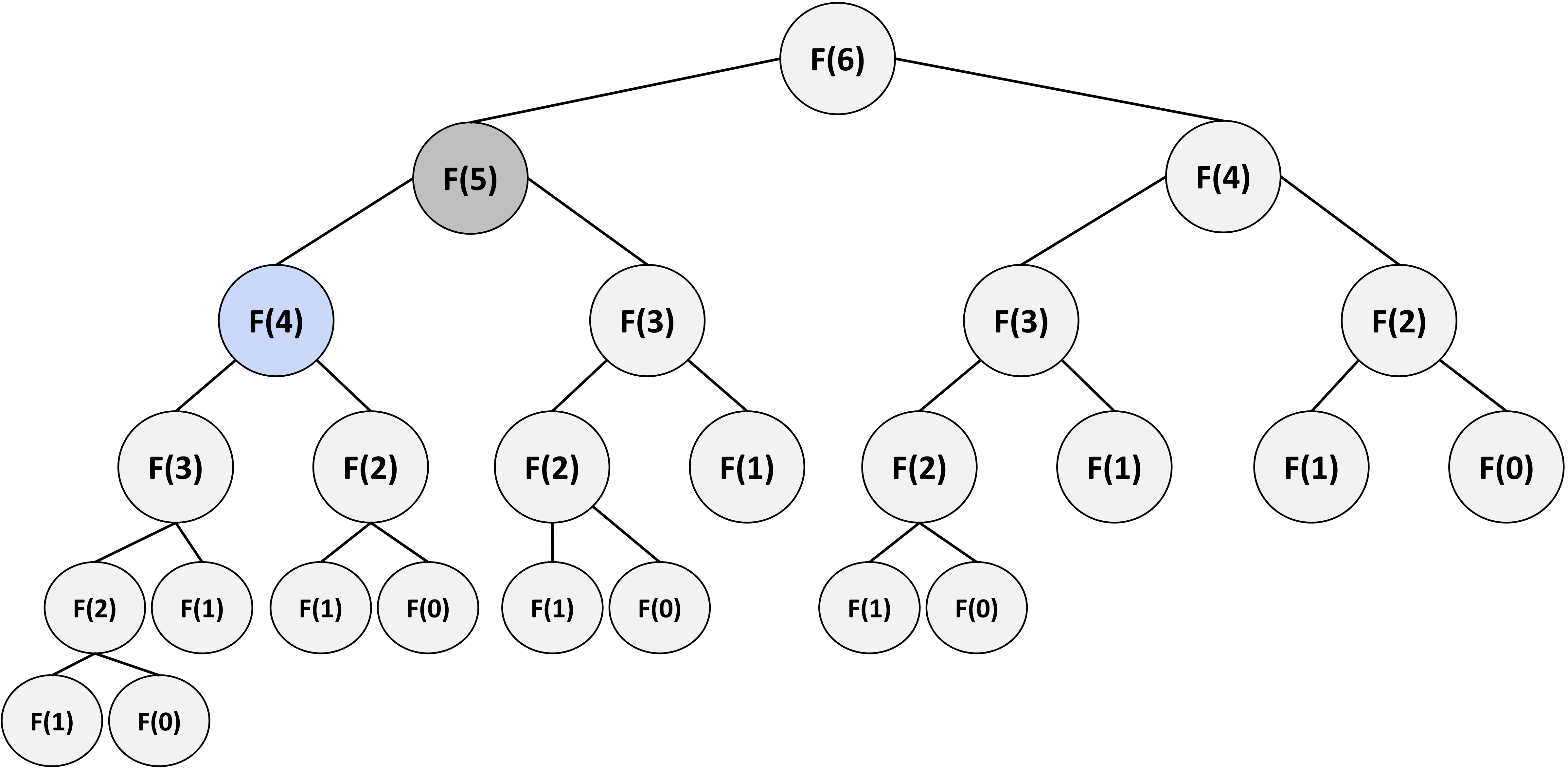
Recursive F_n computation



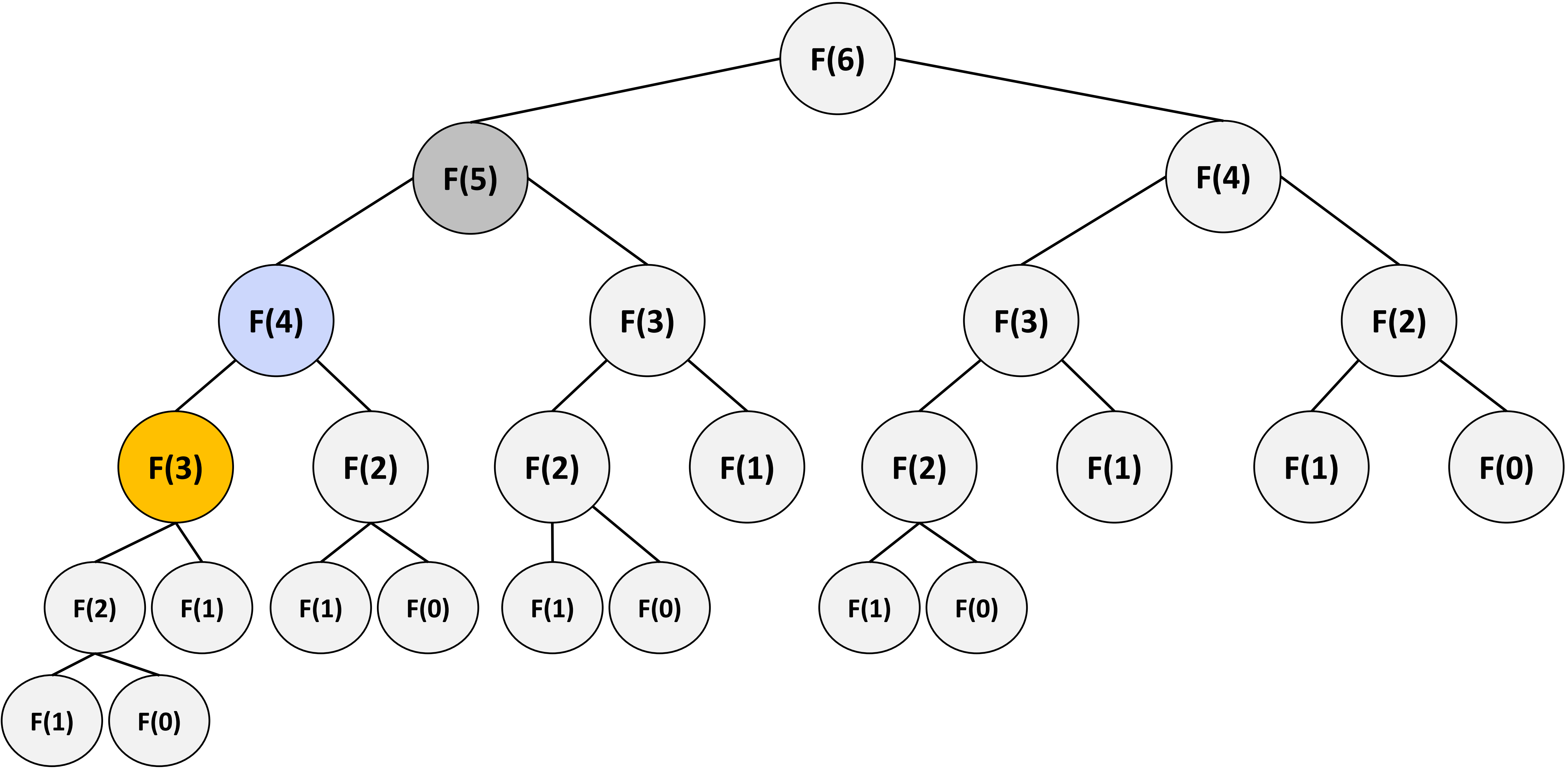
Recursive F_n computation



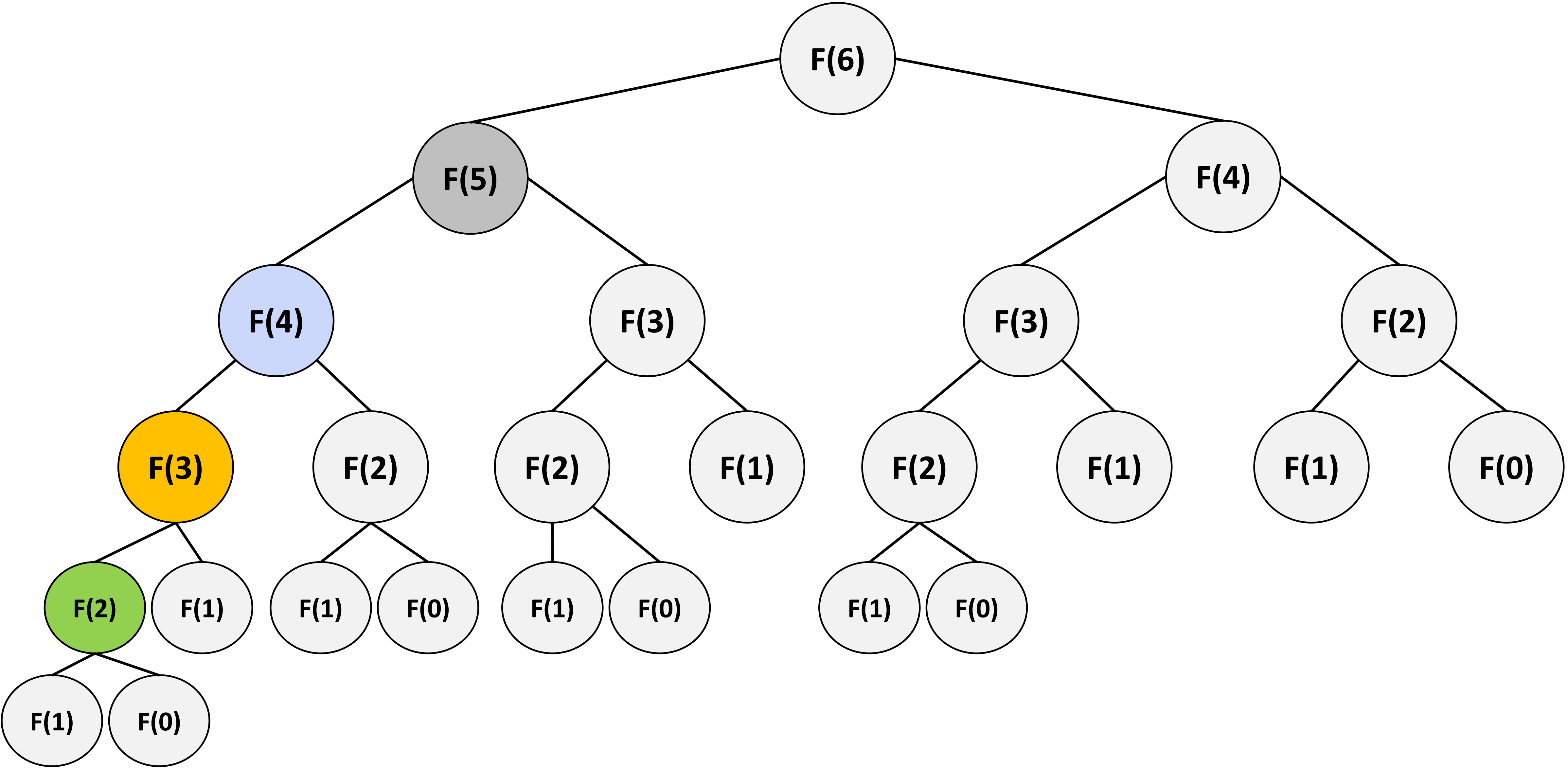
Recursive F_n computation



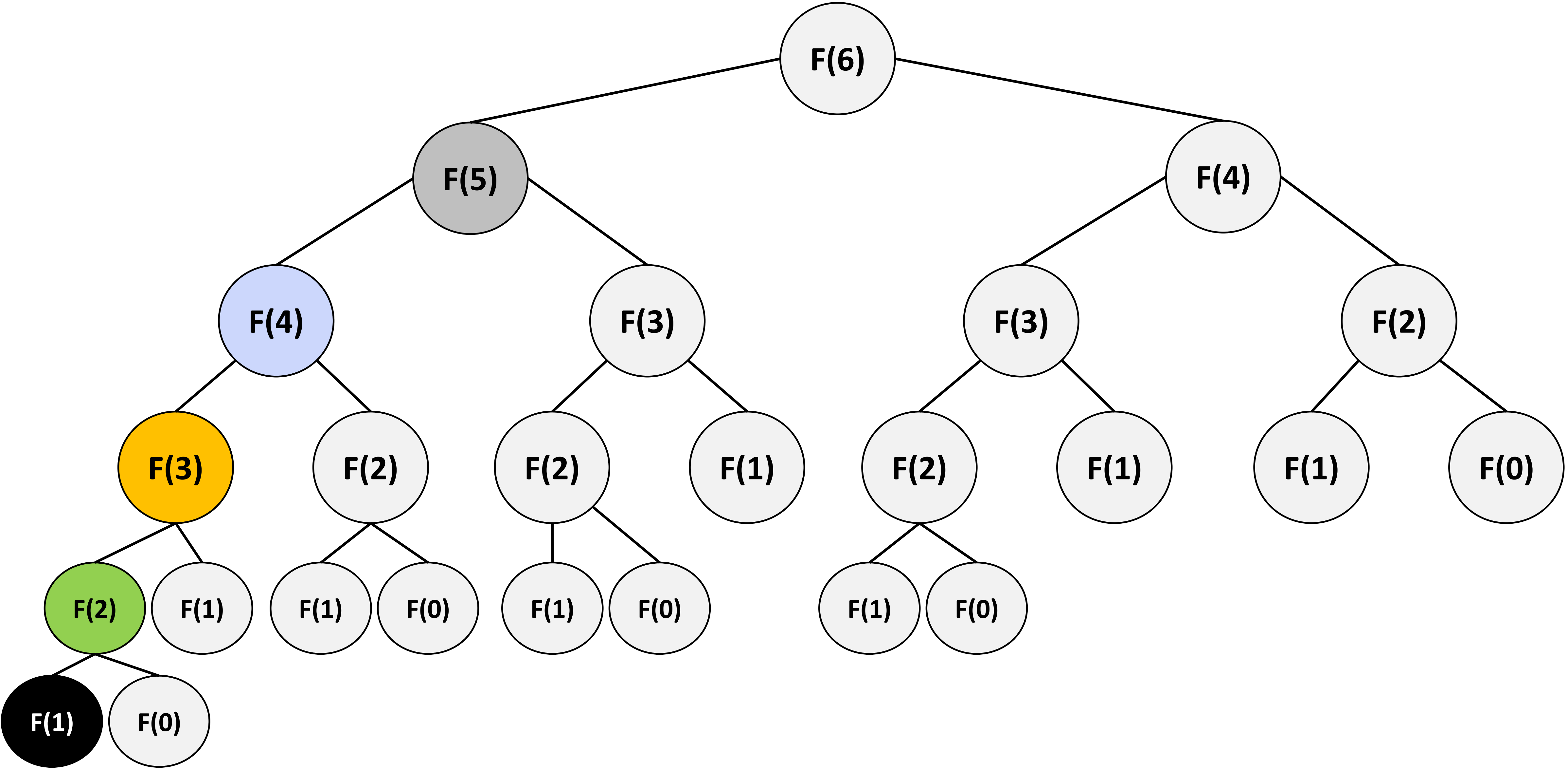
Recursive F_n computation



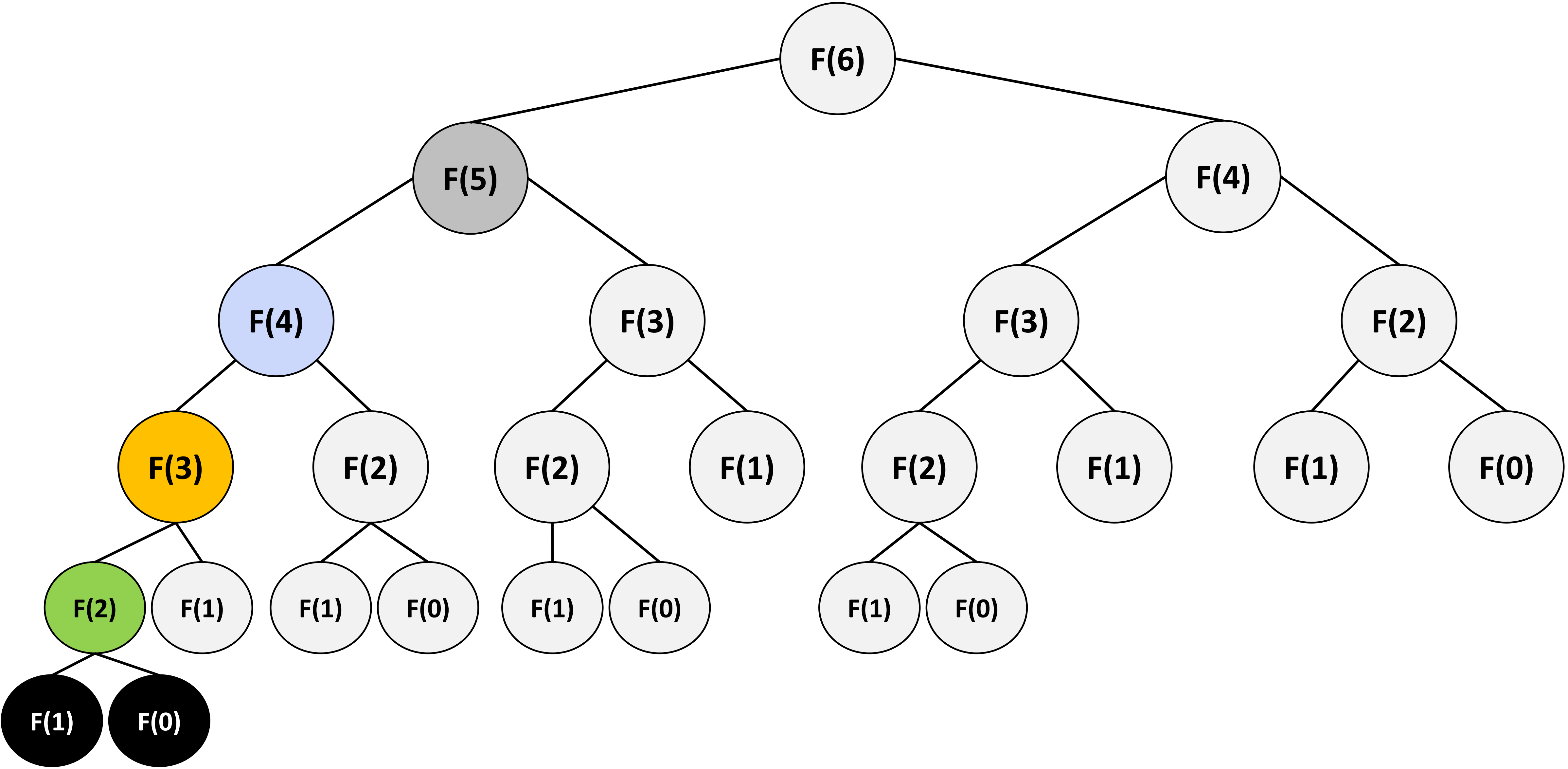
Recursive F_n computation



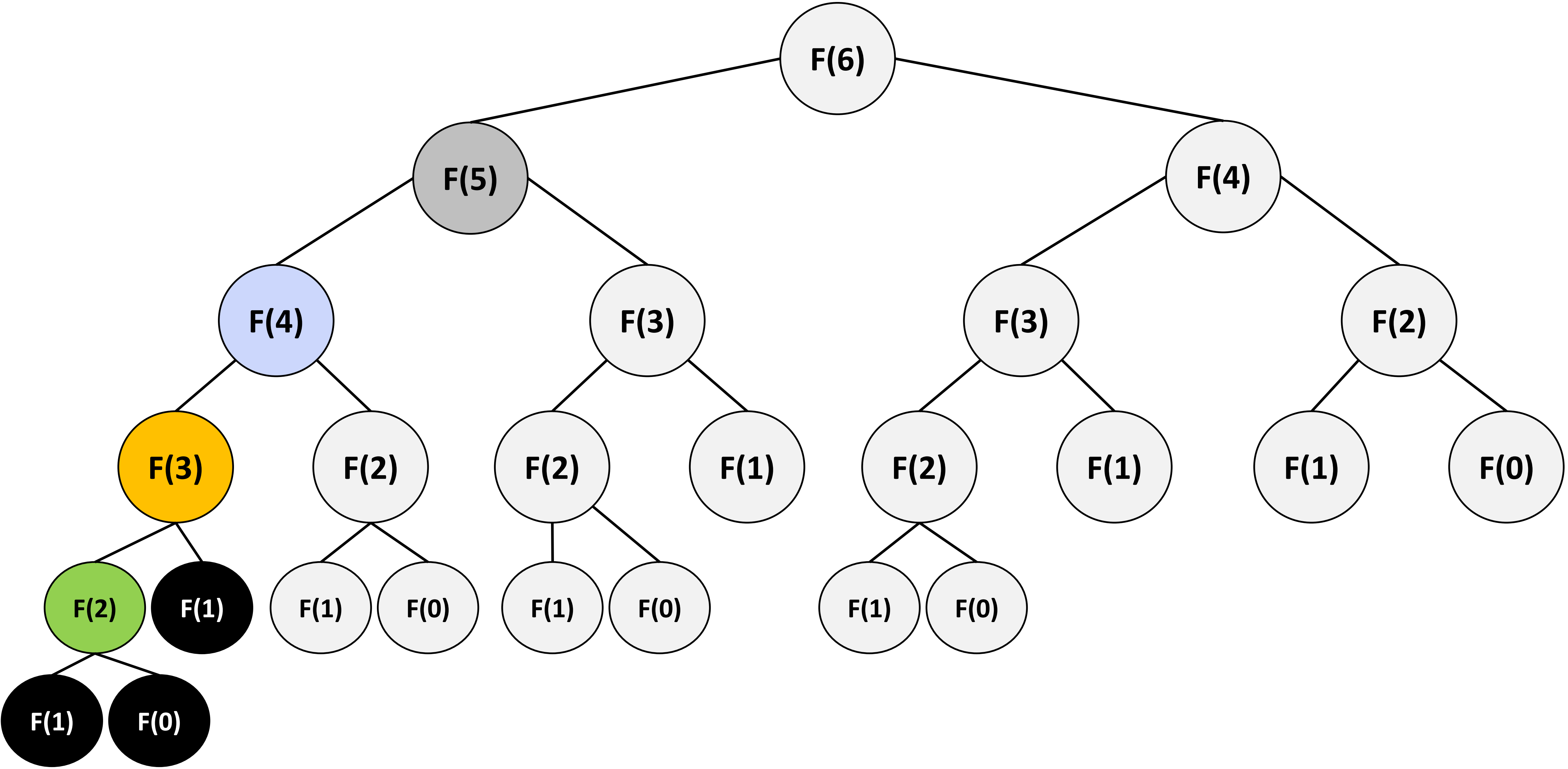
Recursive F_n computation



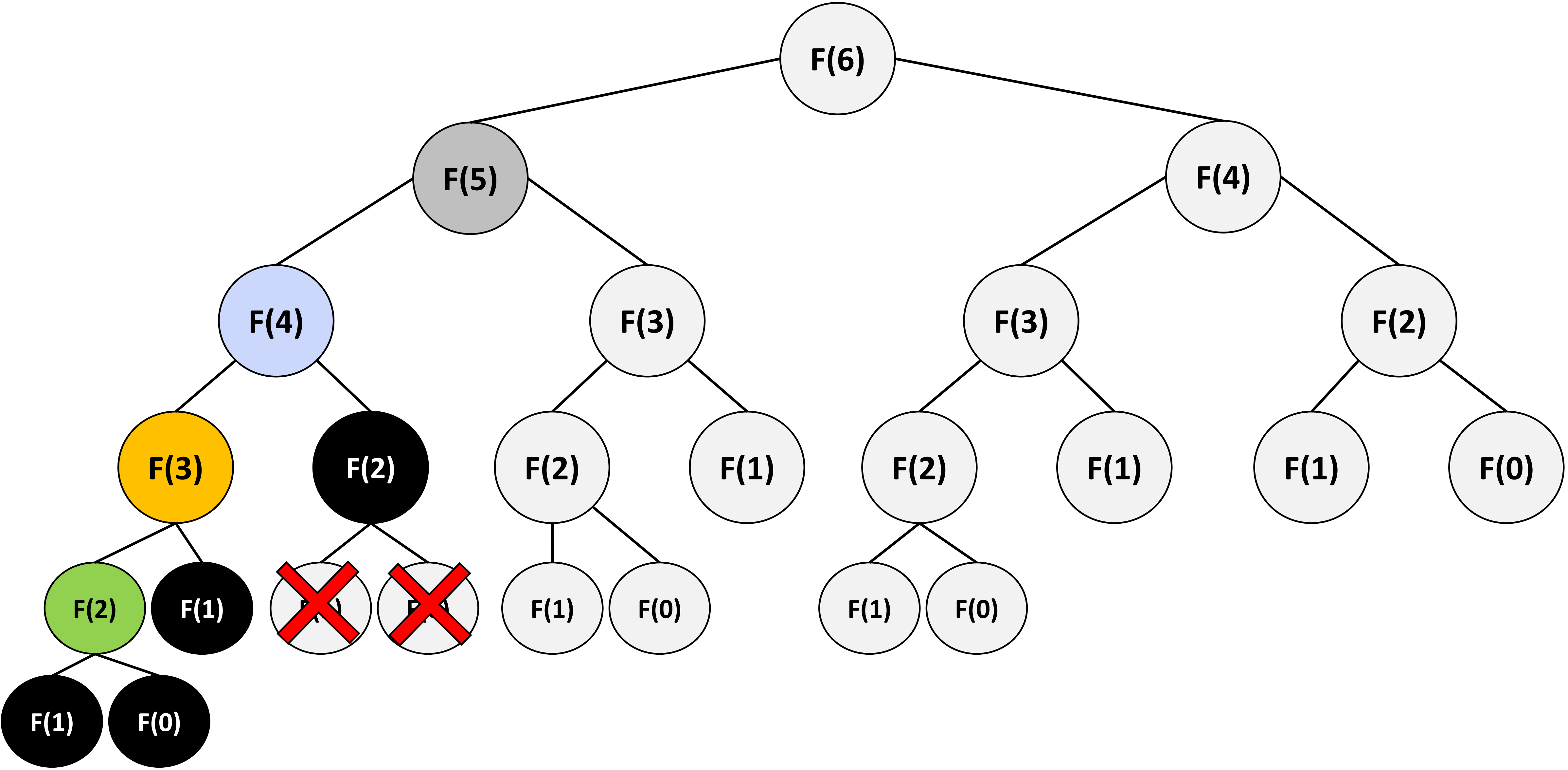
Recursive F_n computation



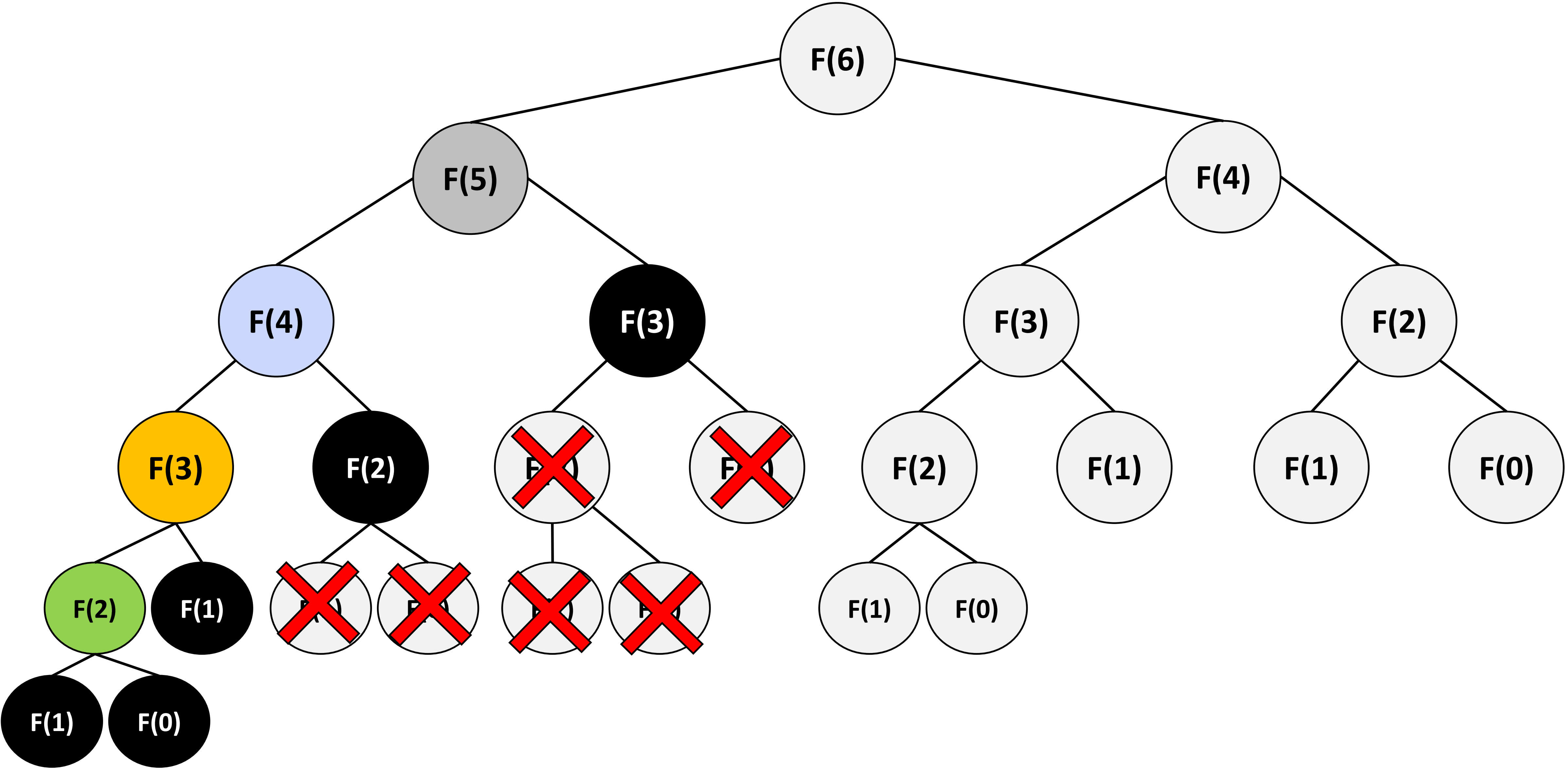
Recursive F_n computation



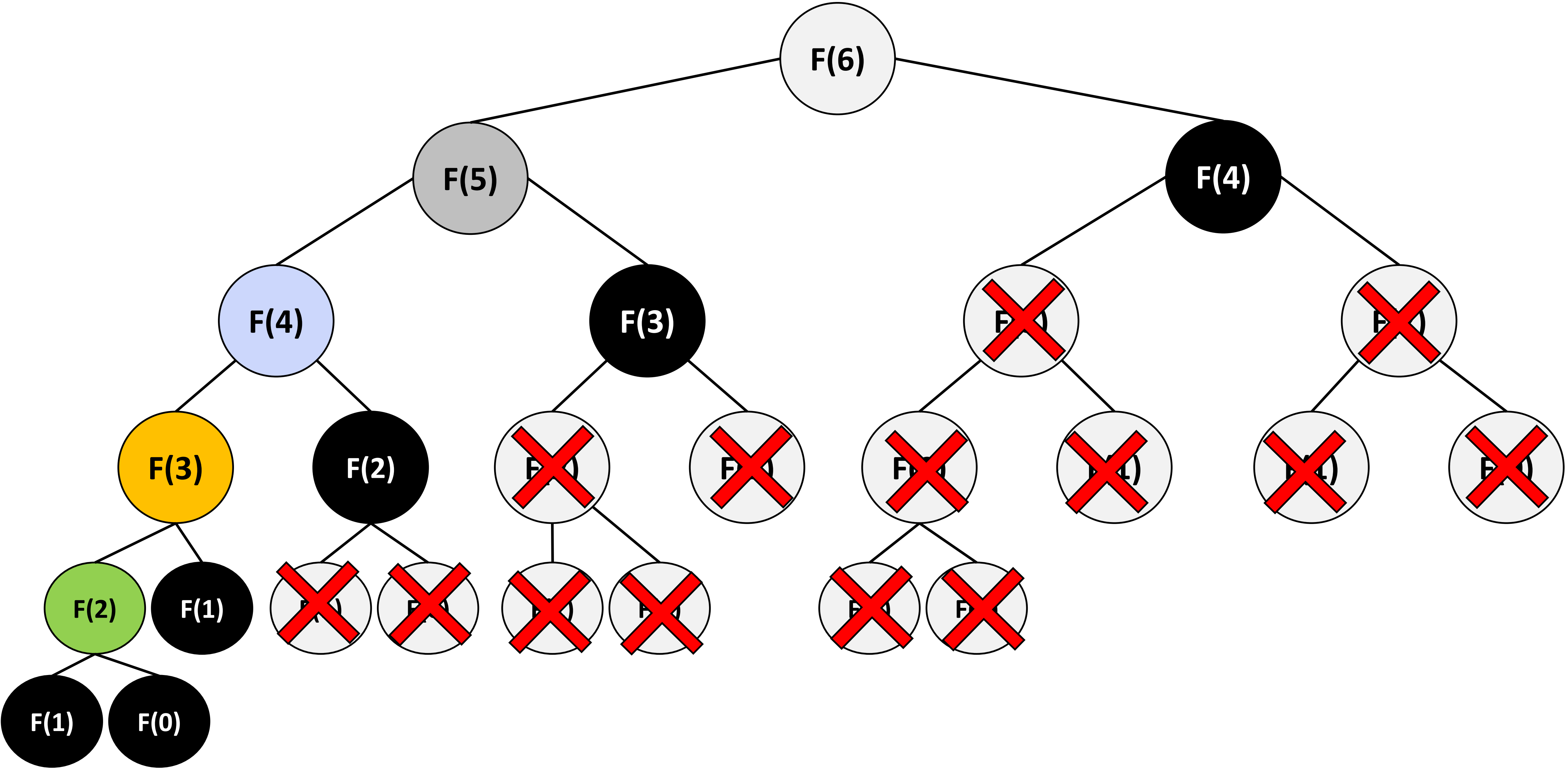
Recursive F_n computation



Recursive F_n computation



Recursive F_n computation



F_n computation with Memoization (How many Calls Now)

FROM

Time Complexity =O(2^n)

To

Time Complexity =O(n)

Weighted Interval Scheduling Problem

Next Class

39

Thanks a lot

If you are taking a Nap, **wake up**........Lecture Over