Análisis y Diseño de Algoritmos II

Jesús Alexander Aranda Ph.D   Robinson Duque, Ph.D
Juan Francisco Díaz, Ph. D

Universidad del Valle

jesus.aranda@correounivalle.edu.co
robinson.duque@correounivalle.edu.co
juanfco.diaz@correounivalle.edu.co
Programa de Ingeniería de Sistemas
Escuela de Ingeniería de Sistemas y Computación

Programación dinámica

Introducción

Al igual que la técnica de *Dividir y conquistar*, la programación dinámica es una técnica para resolver problemas a partir de la solución de subproblemas y la combinación de esas soluciones.

La programación dinámica es útil cuando los subproblemas se repiten.

Un algoritmo que sigue esta técnica resuelve cada subproblema una sola vez y guarda dicha solución para poder ser reutilizada.

Programación dinámica

La programación dinámica es una técnica de diseño para desarrollar algoritmos eficientes partiendo de una solución algorítmica recursiva almacenando resultados parciales que se pueden utilizar varias veces.

Programación dinámica

La programación dinámica se suele aplicar para resolver problemas de optimización de forma eficiente y garantizando correctitud:

- Problemas en los que se pueden encontrar muchas soluciones
- Cada solución tiene un valor asociado
- Se busca una solución que tenga un valor asociado que sea óptimo (máximo o mínimo) entre las muchas soluciones que pueden existir

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

¿Cómo determinar el valor de f_i algoritmicamente?

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

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

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

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

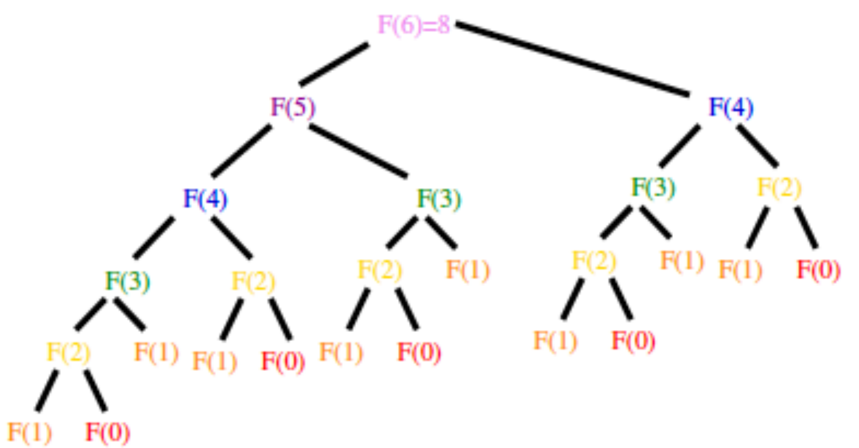
Si $f_n / f_{n-1} \approx 1.61$

¿Cúal sería el orden de complejidad de Fibonacci?

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

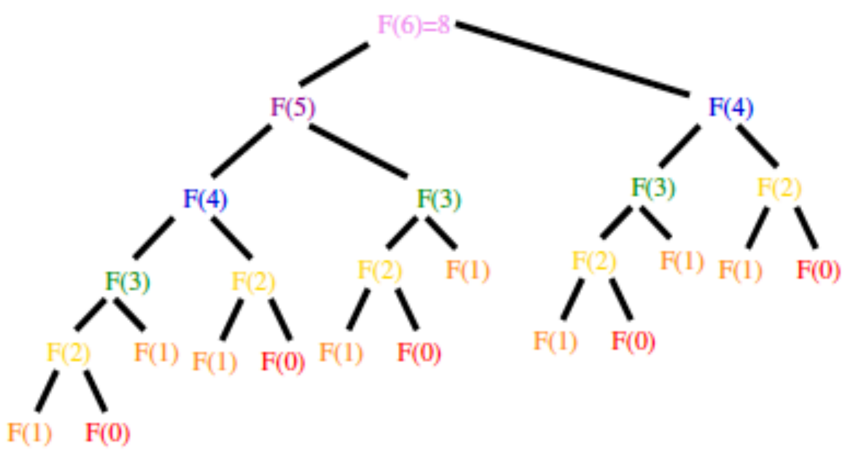


Observación clave:
$f_n / f_{n-1} \approx 1.61$

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$



Observación clave:
$\boldsymbol{f_n / f_{n-1} \approx 1.61}$

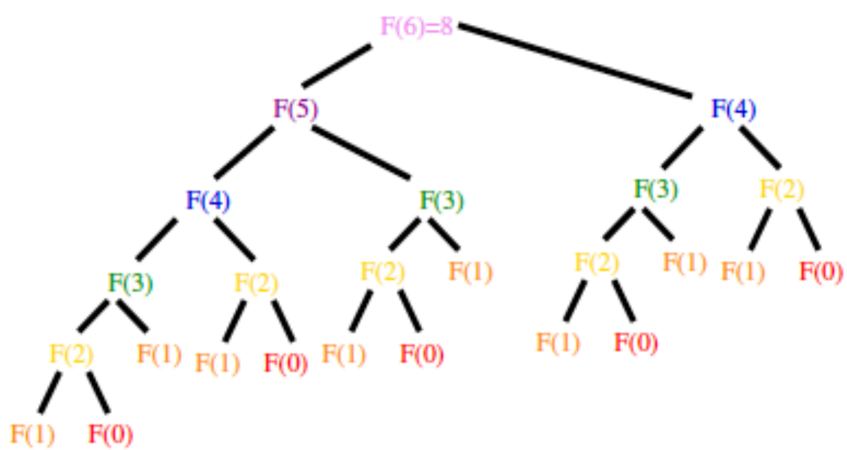
$\boldsymbol{f_n > 1.6^n}$

¡Orden Exponencial!
¿Por qué?

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

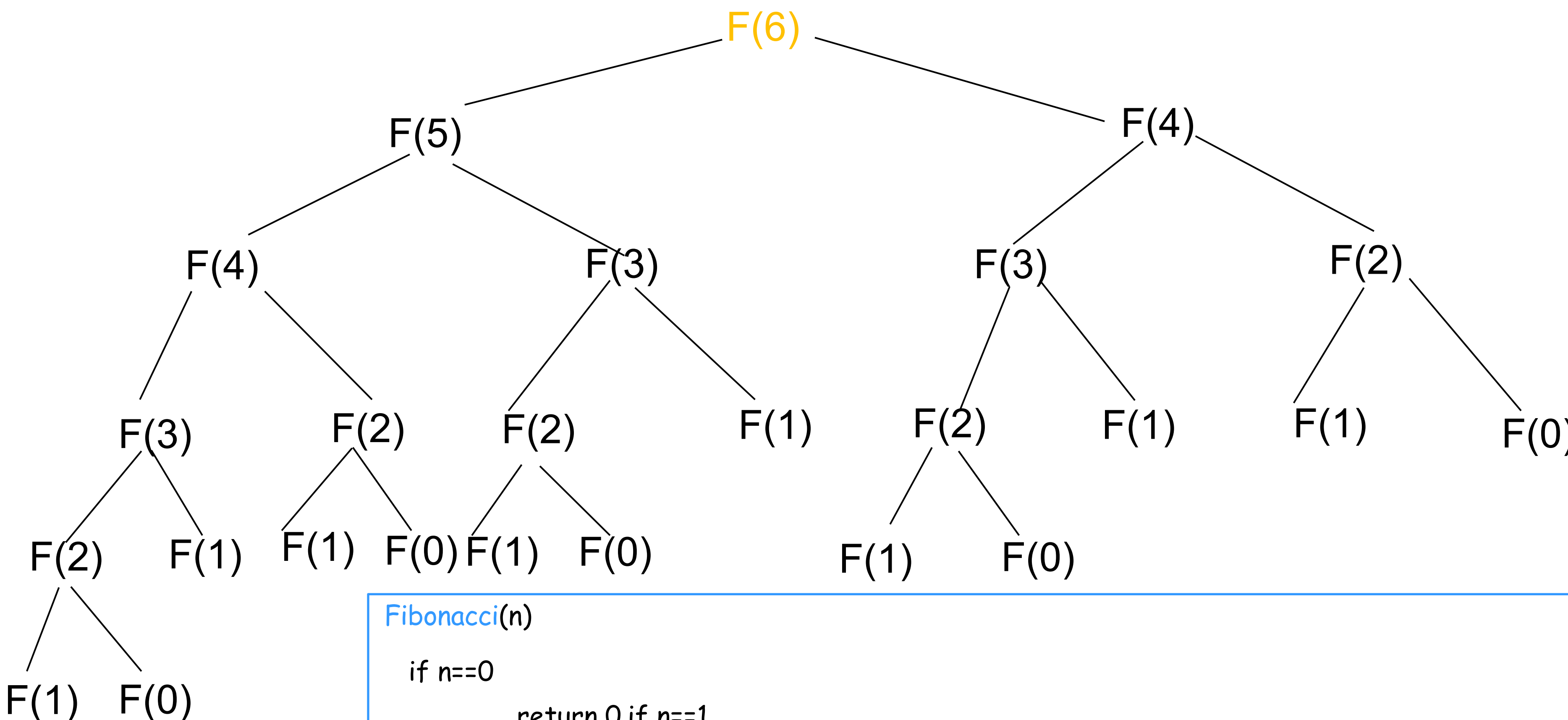
```
Fibonacci(n)
 if n==0
        return 0
 if n==1
        return 1
 else
        return Fibonacci(n-1) + Fibonacci(n-2)
```



Se necesita calcular muchas veces la solución a un mismo subproblema para encontrar la solución.

Programación dinámica

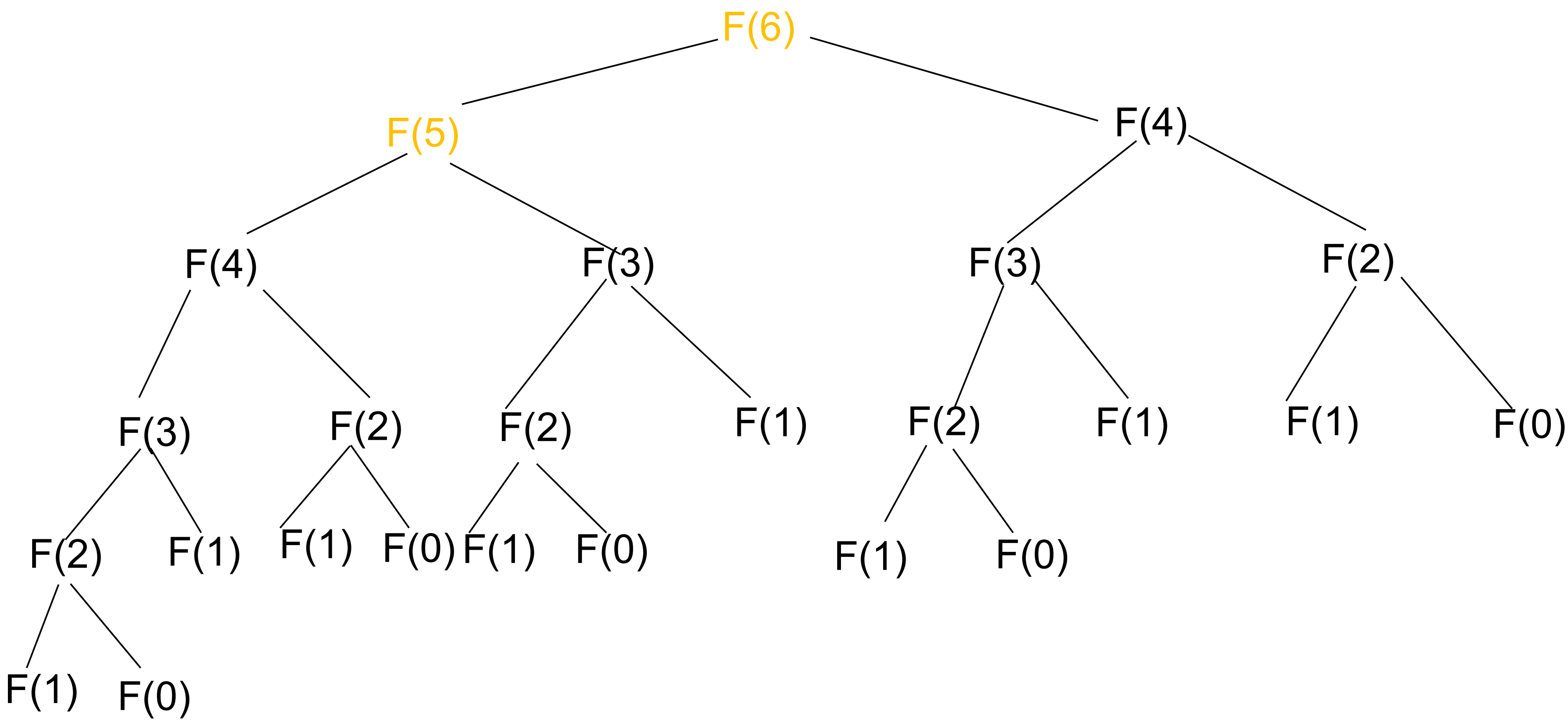
Caso Estudio: Sucesión de Fibonacci




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

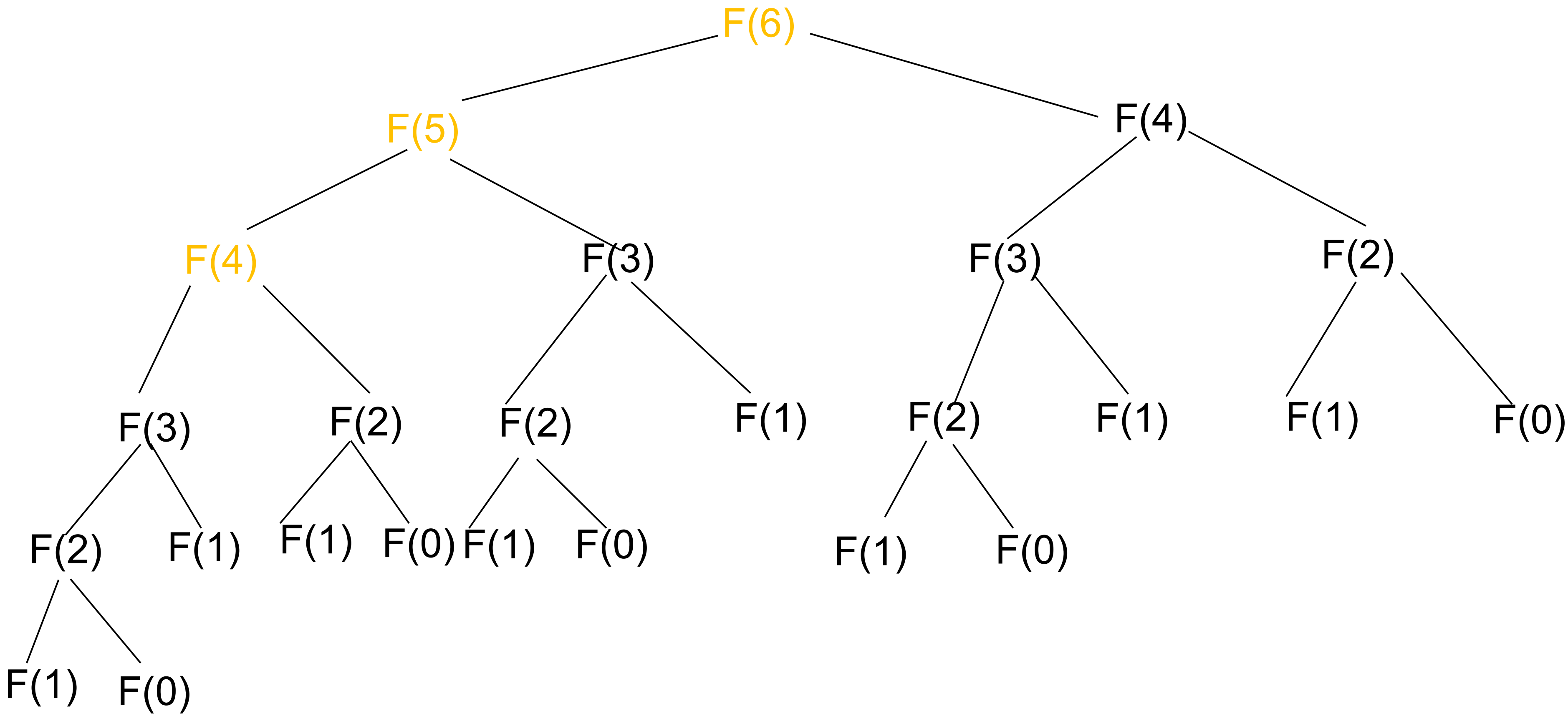
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



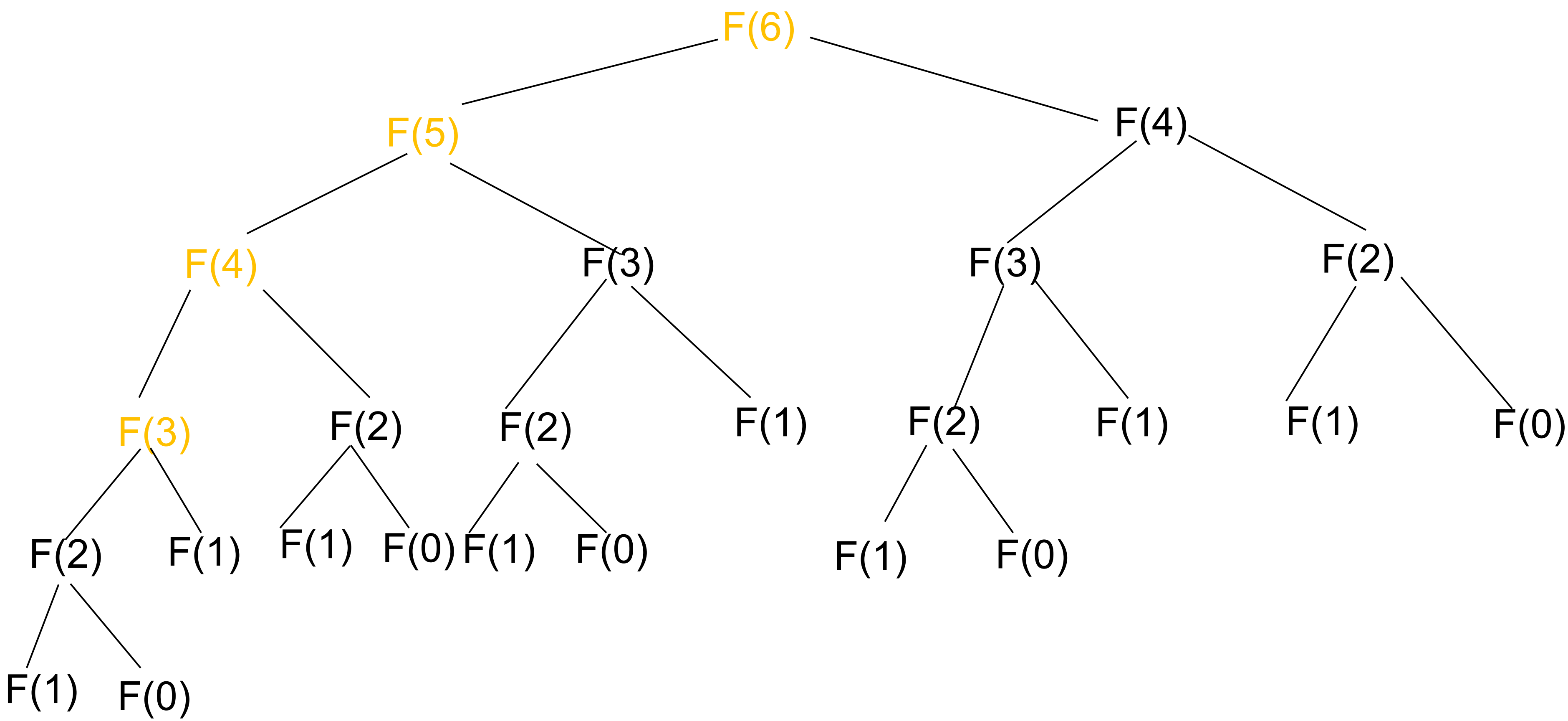
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



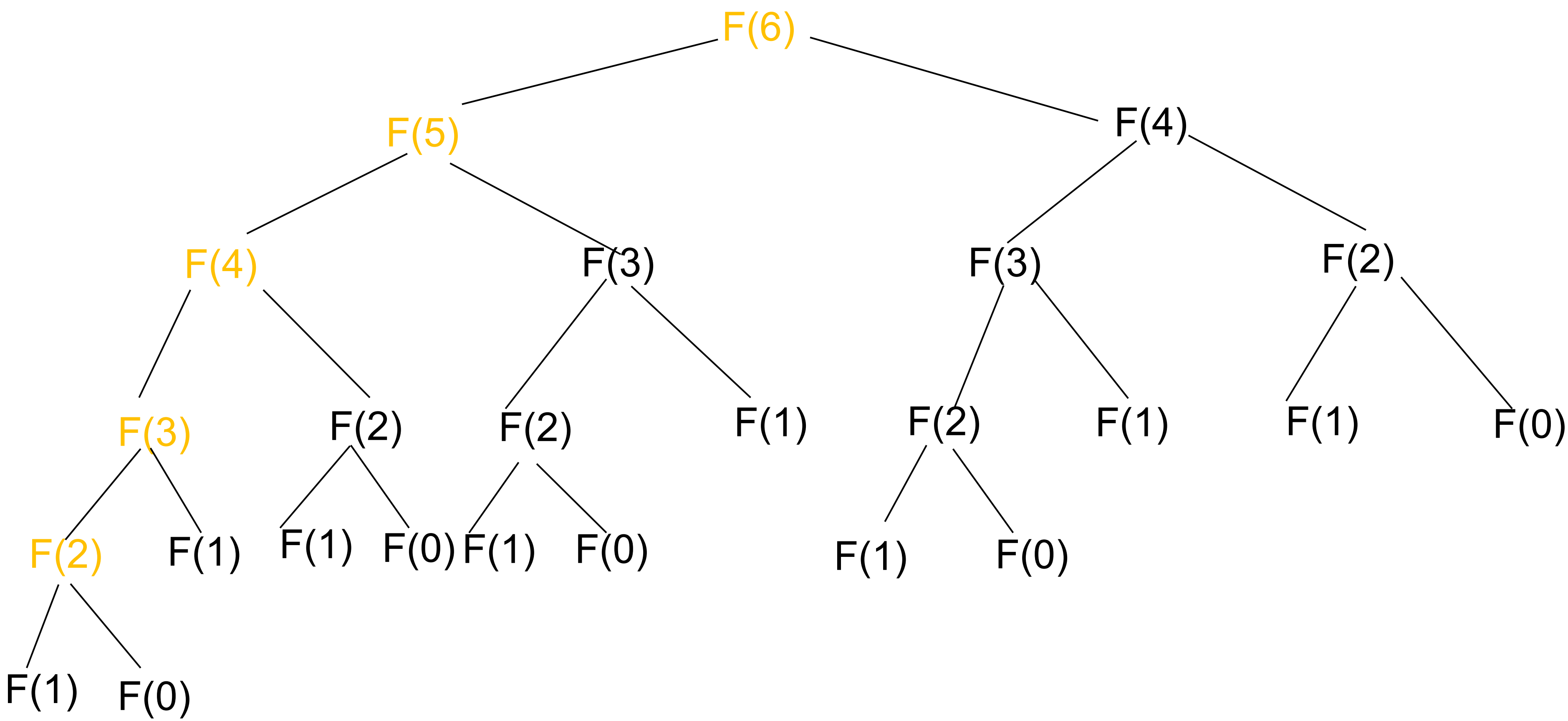
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



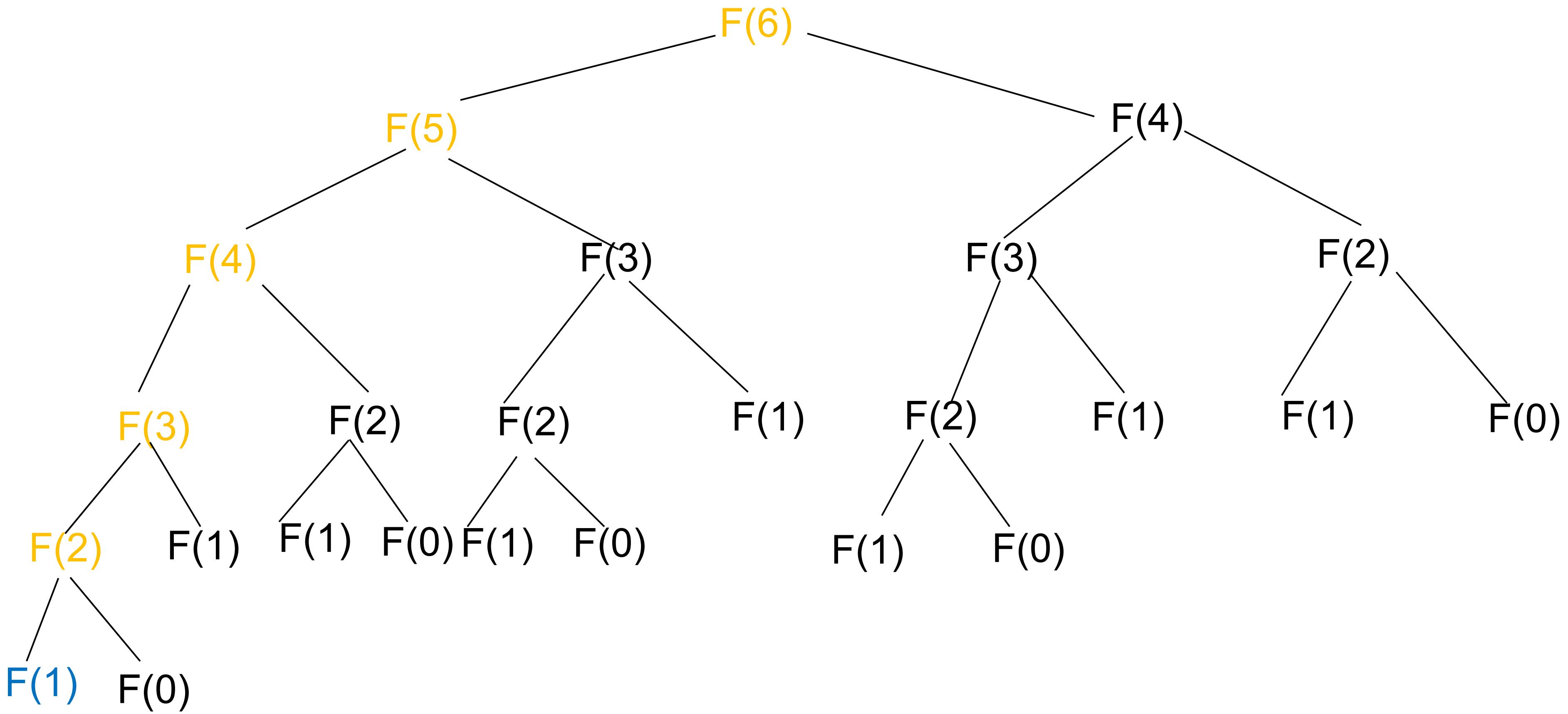
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



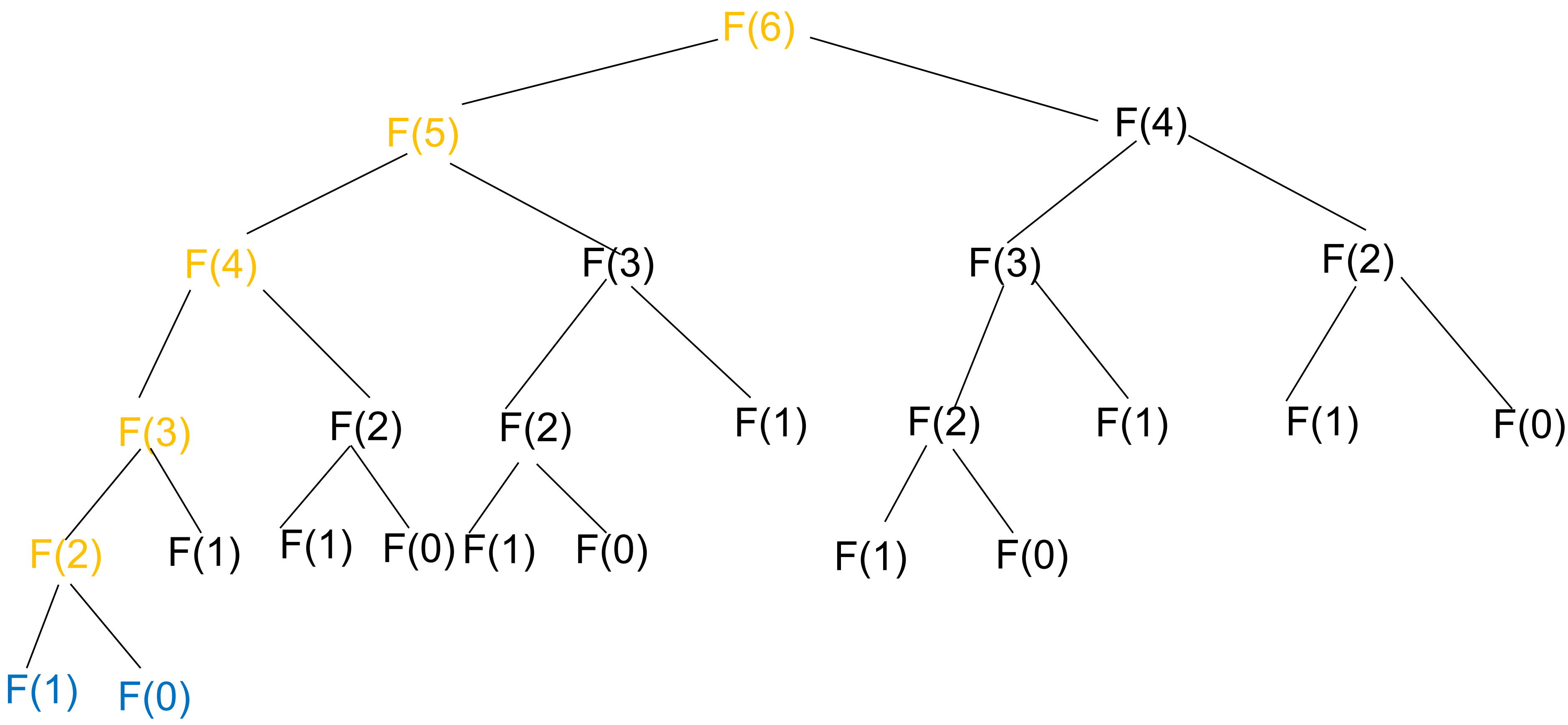
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



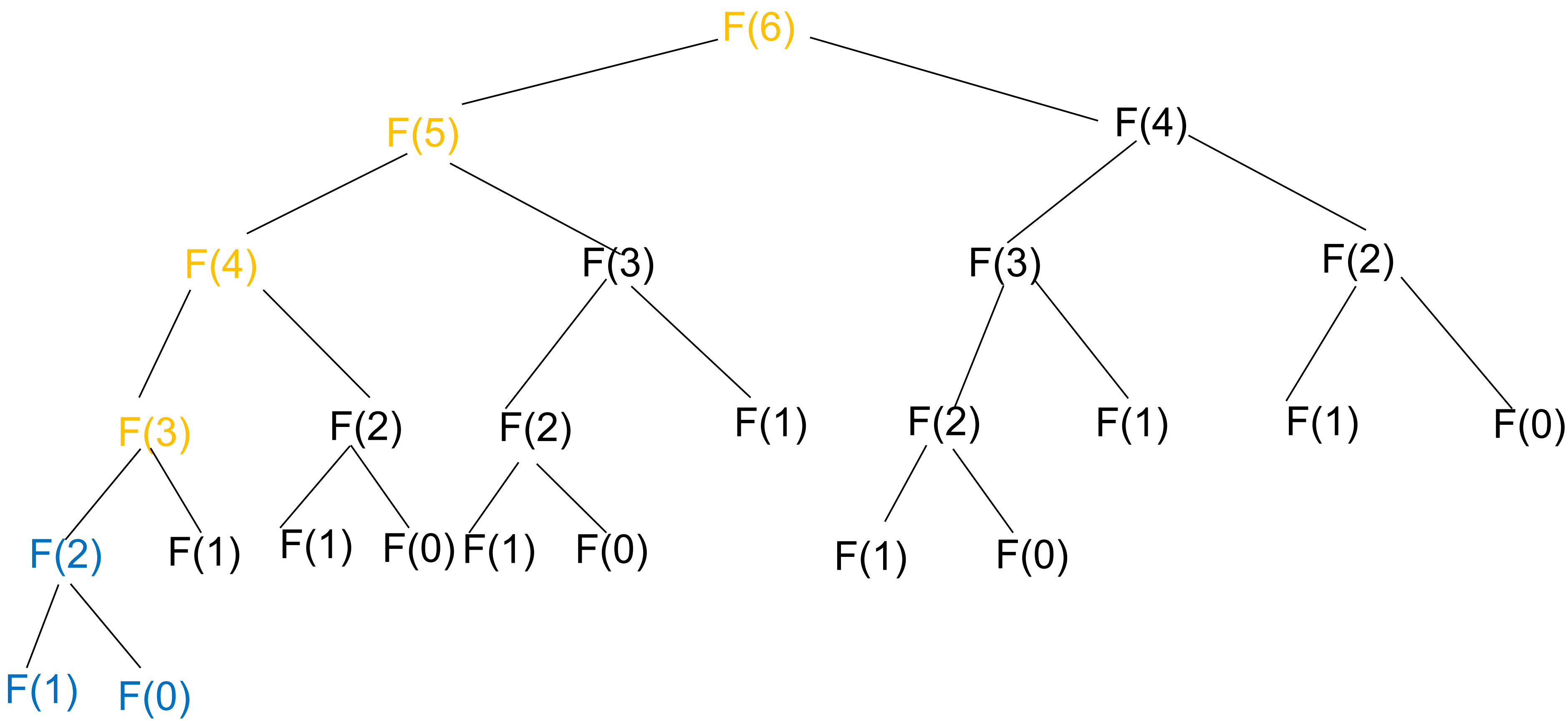
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



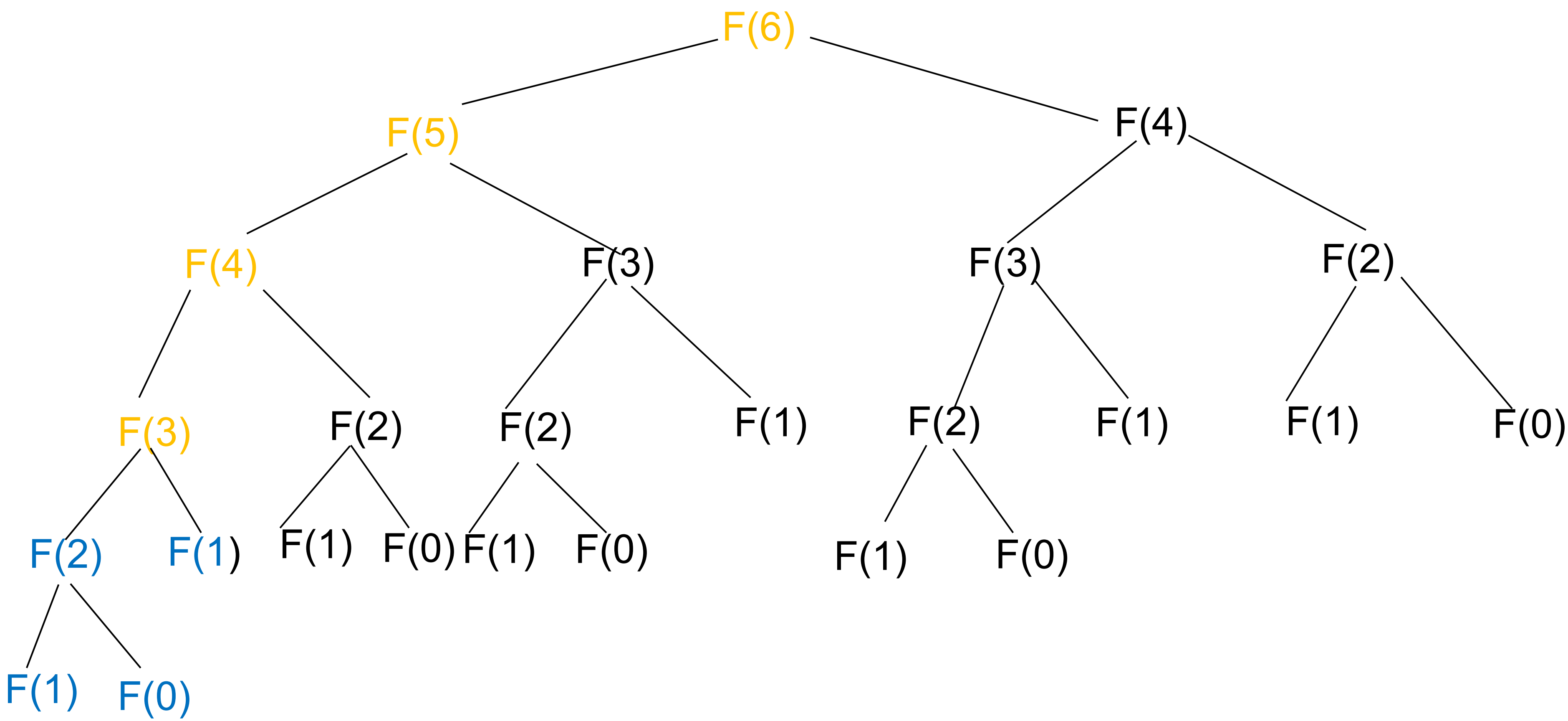
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



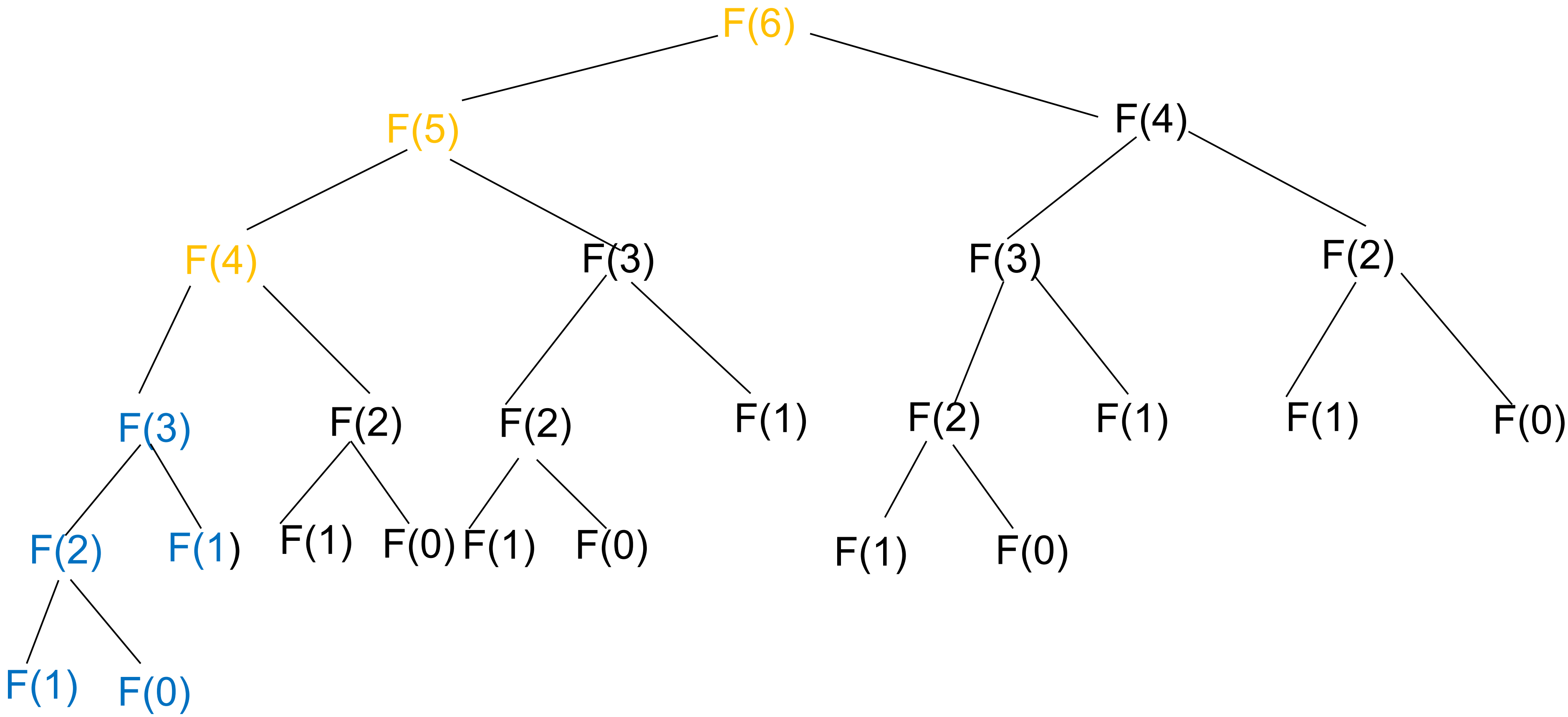
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



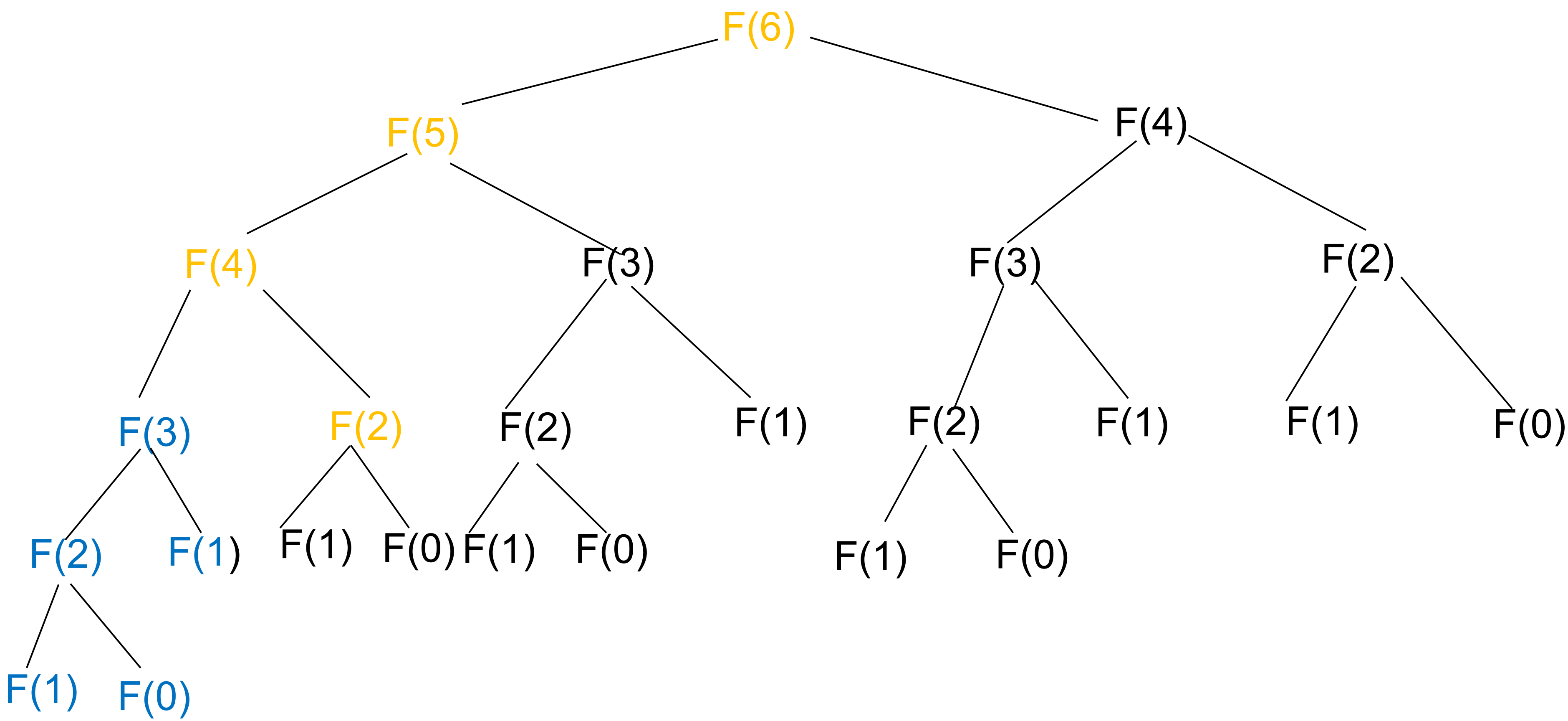
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



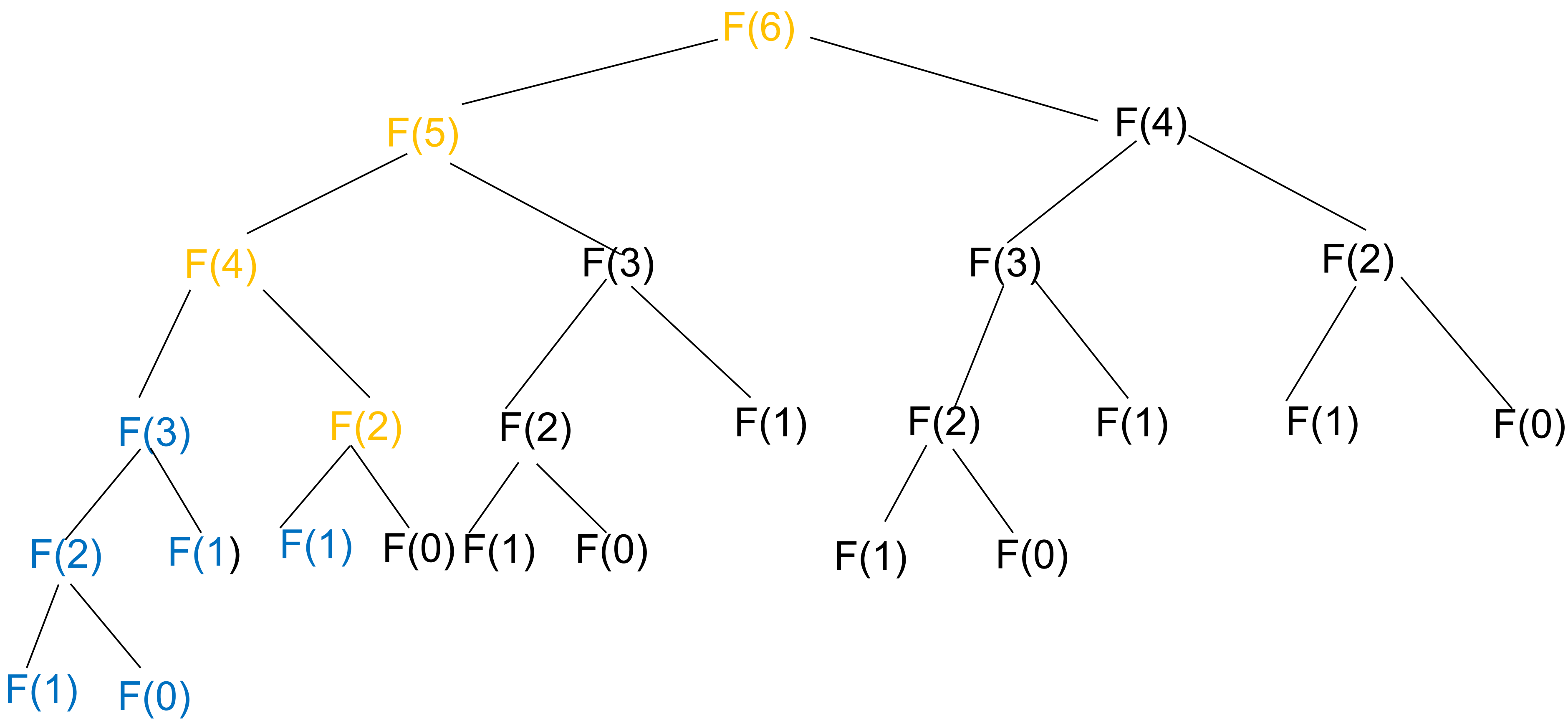
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



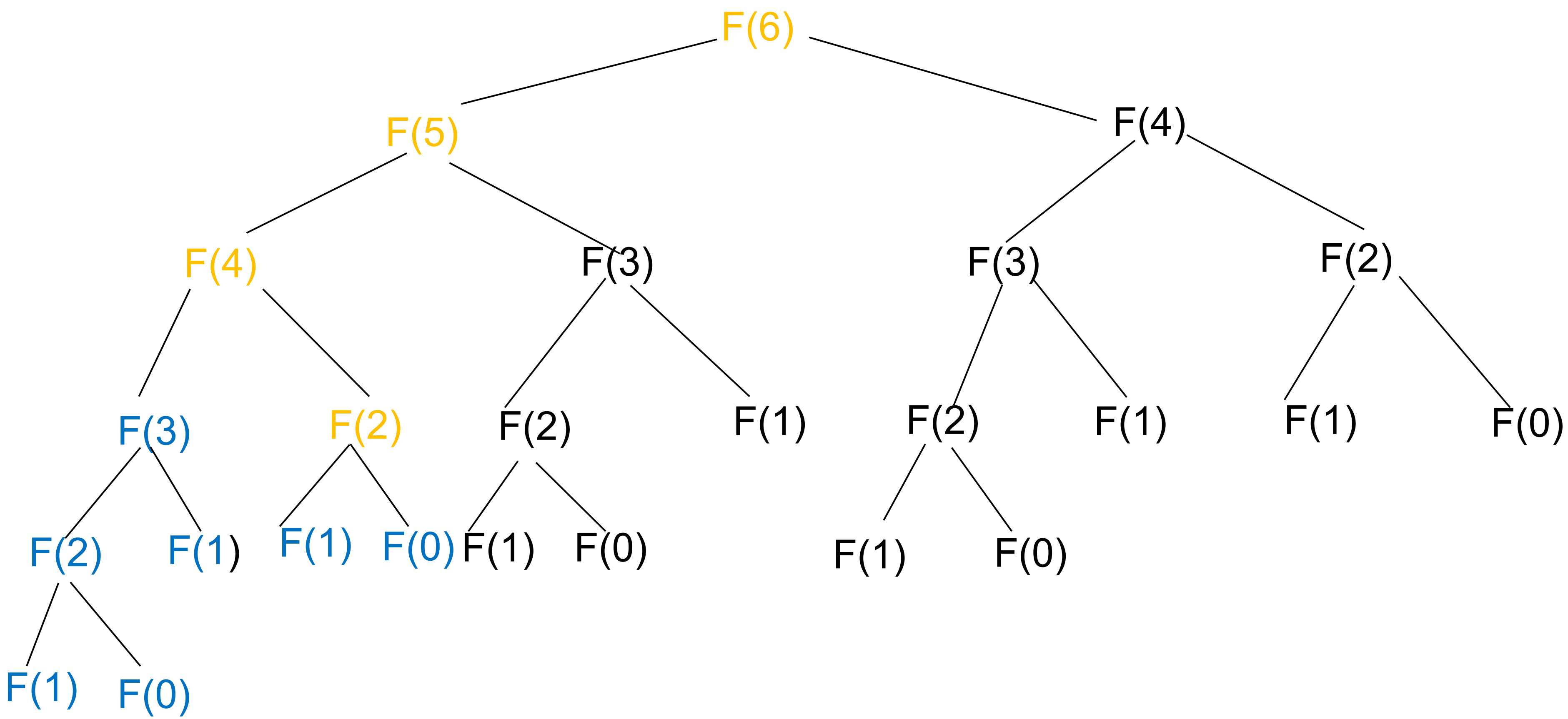
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



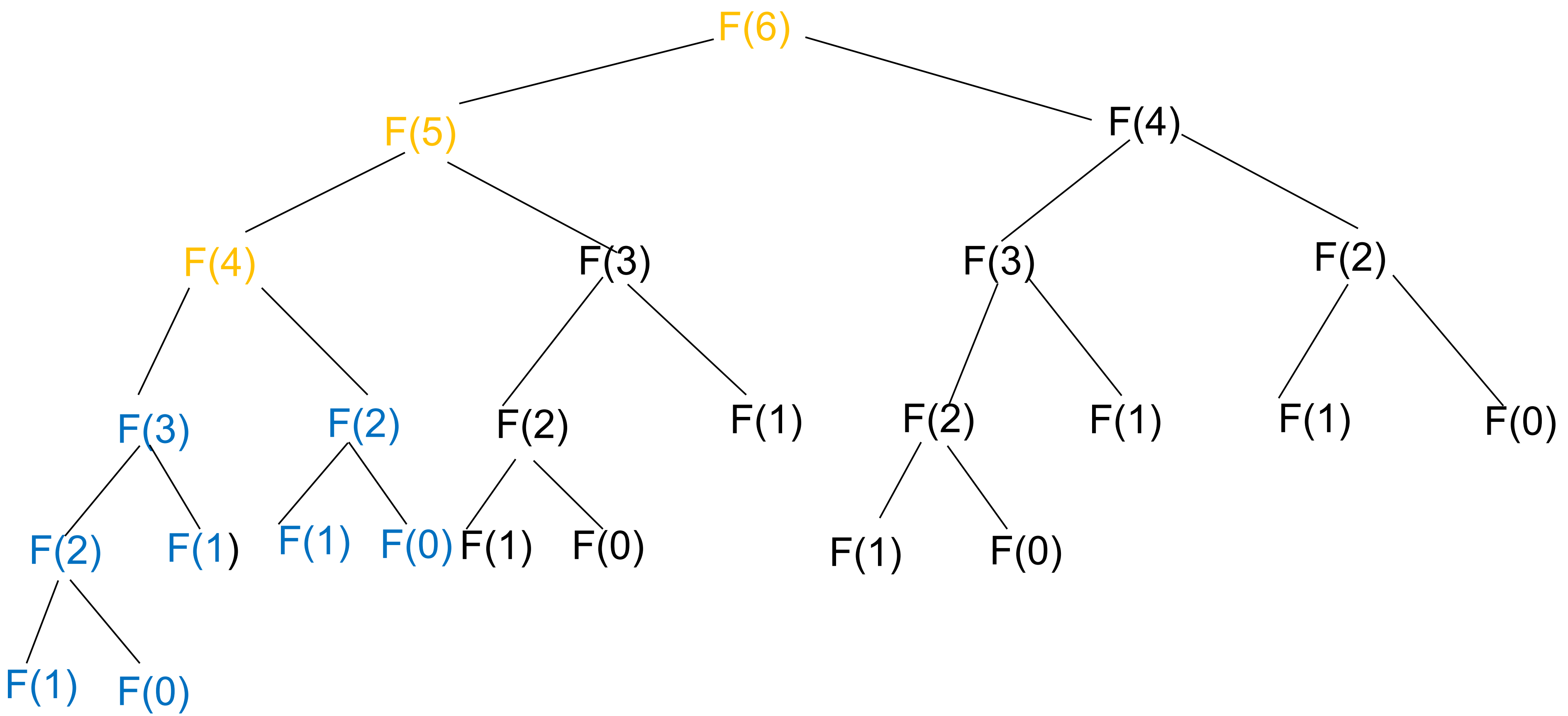
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



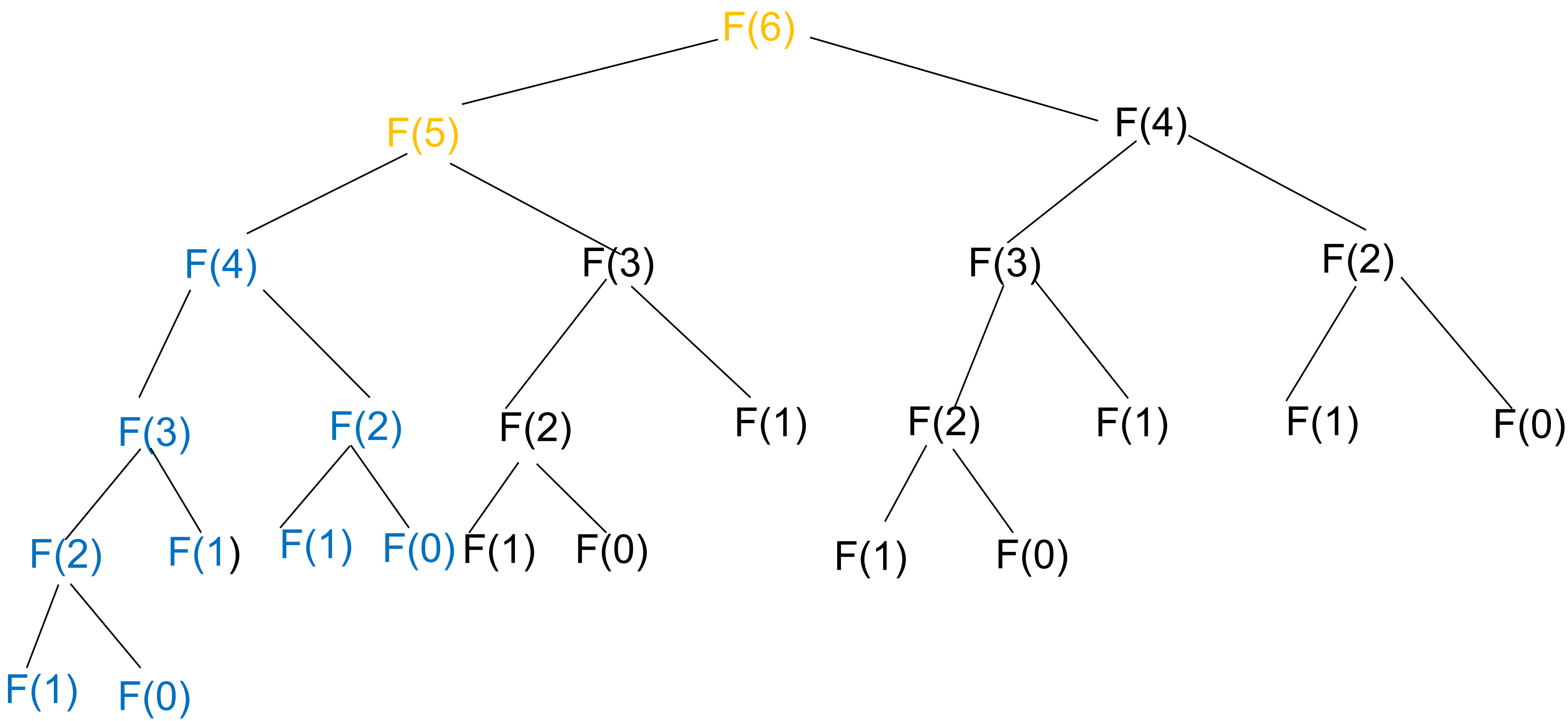
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



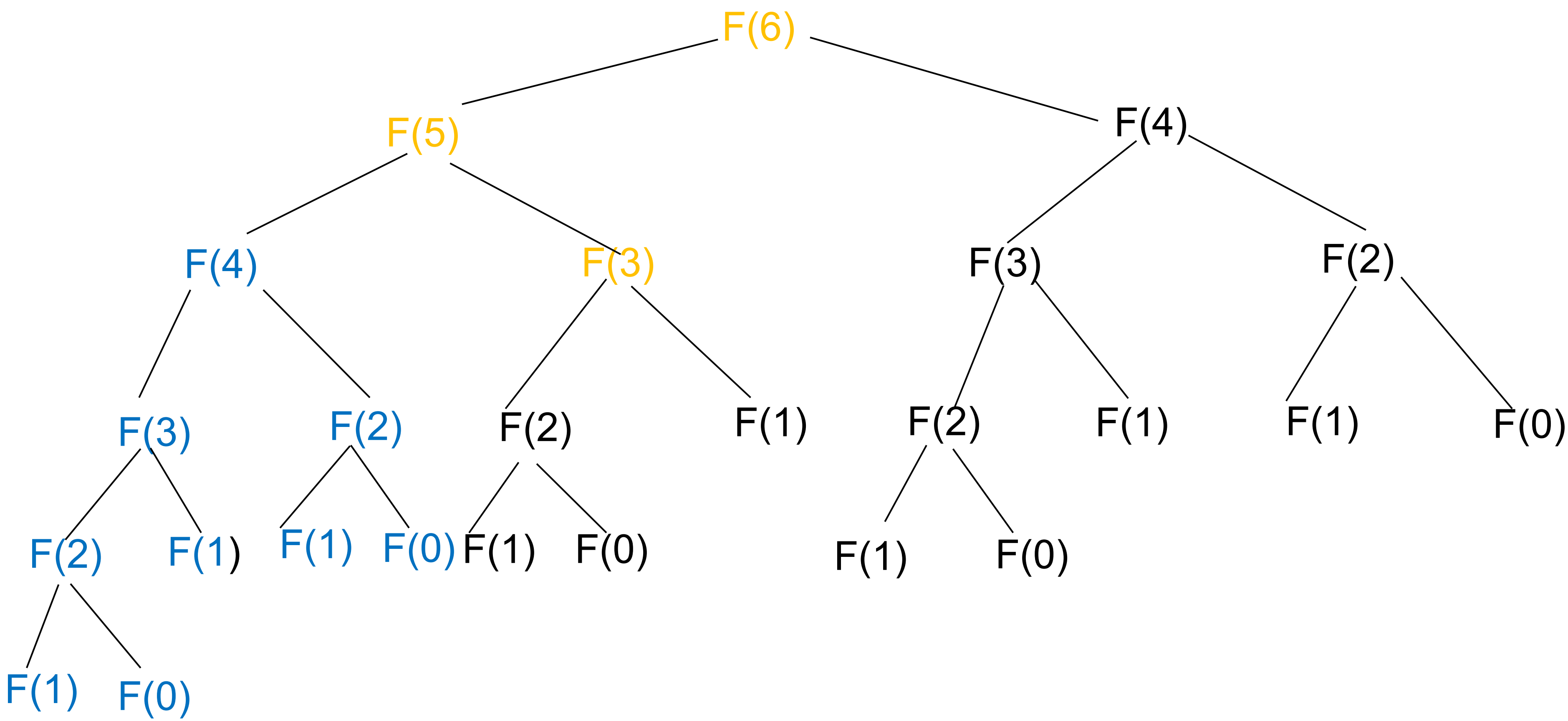
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



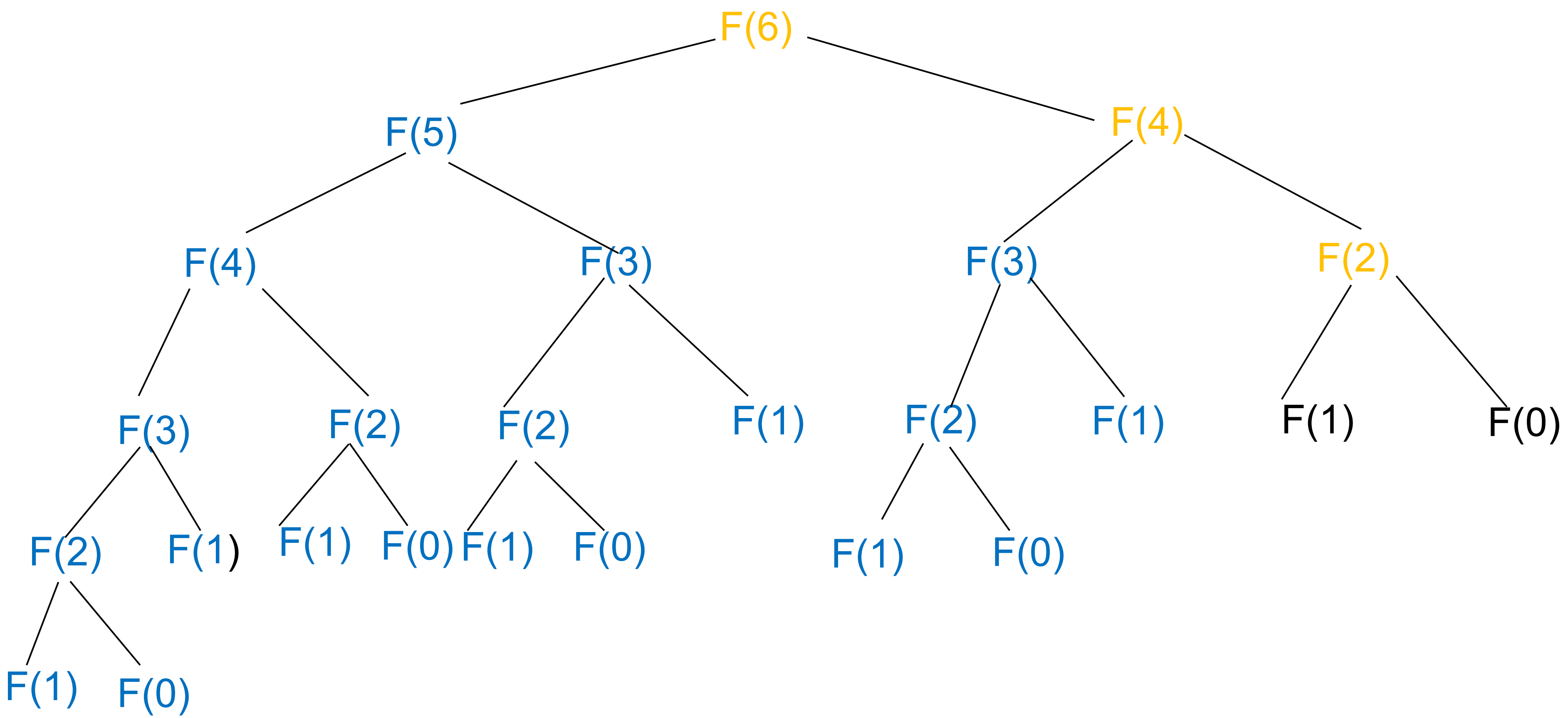
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



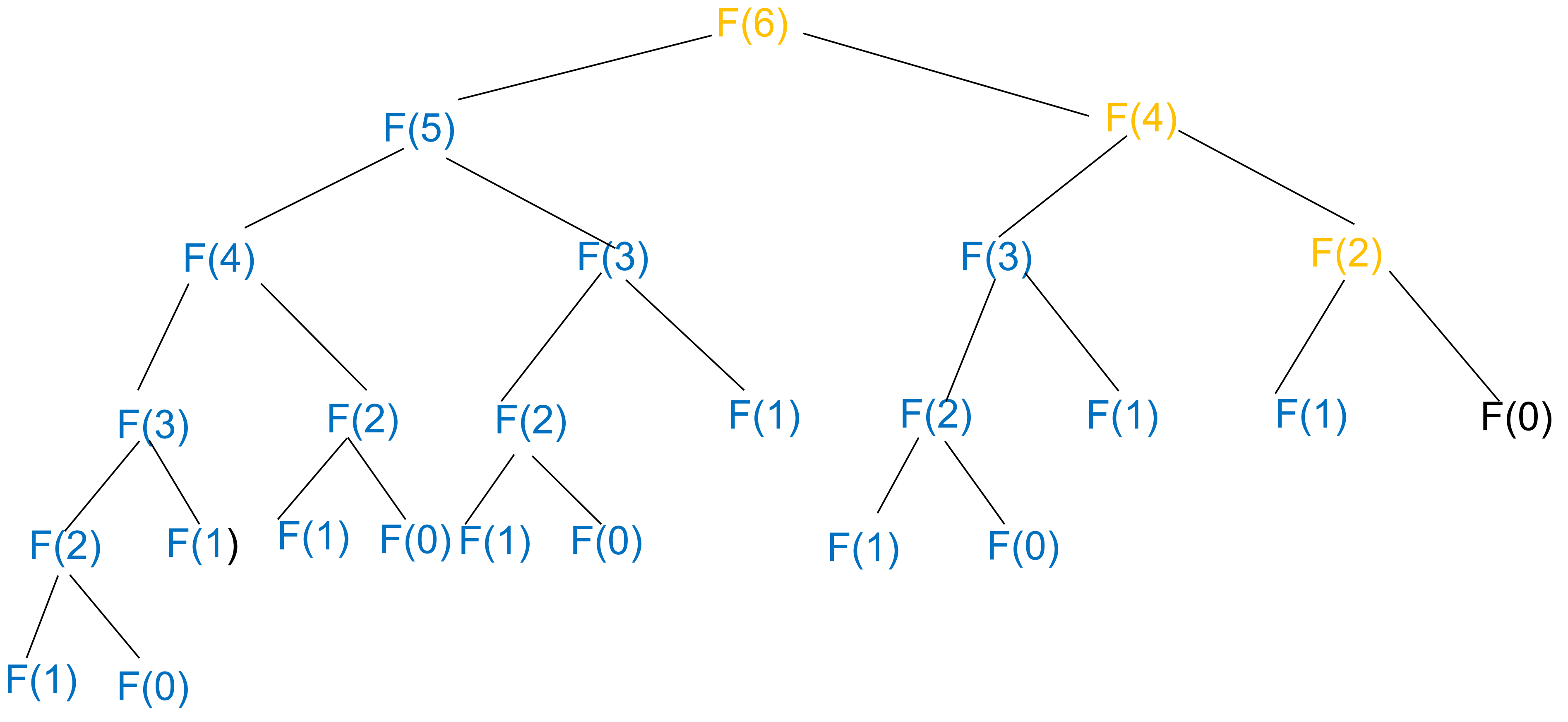
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



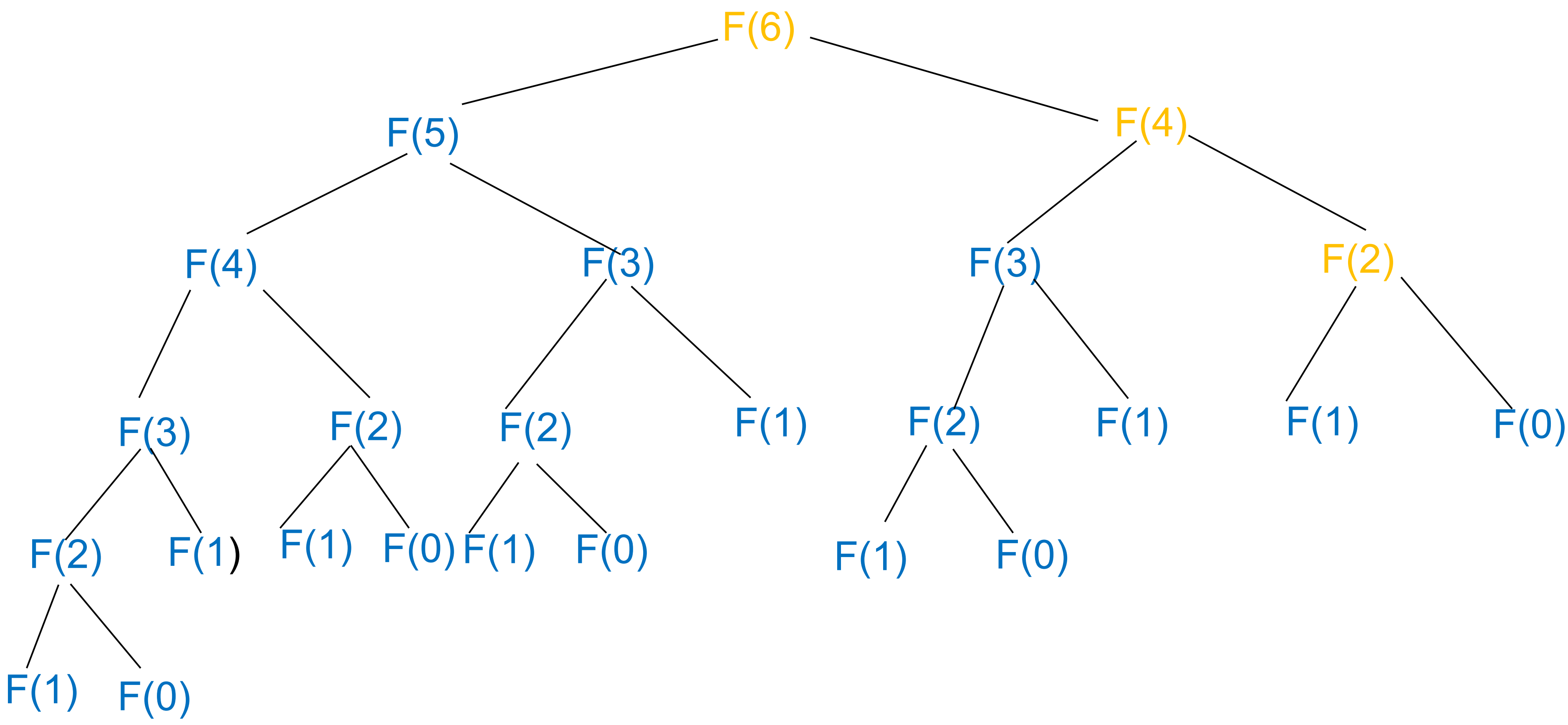
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



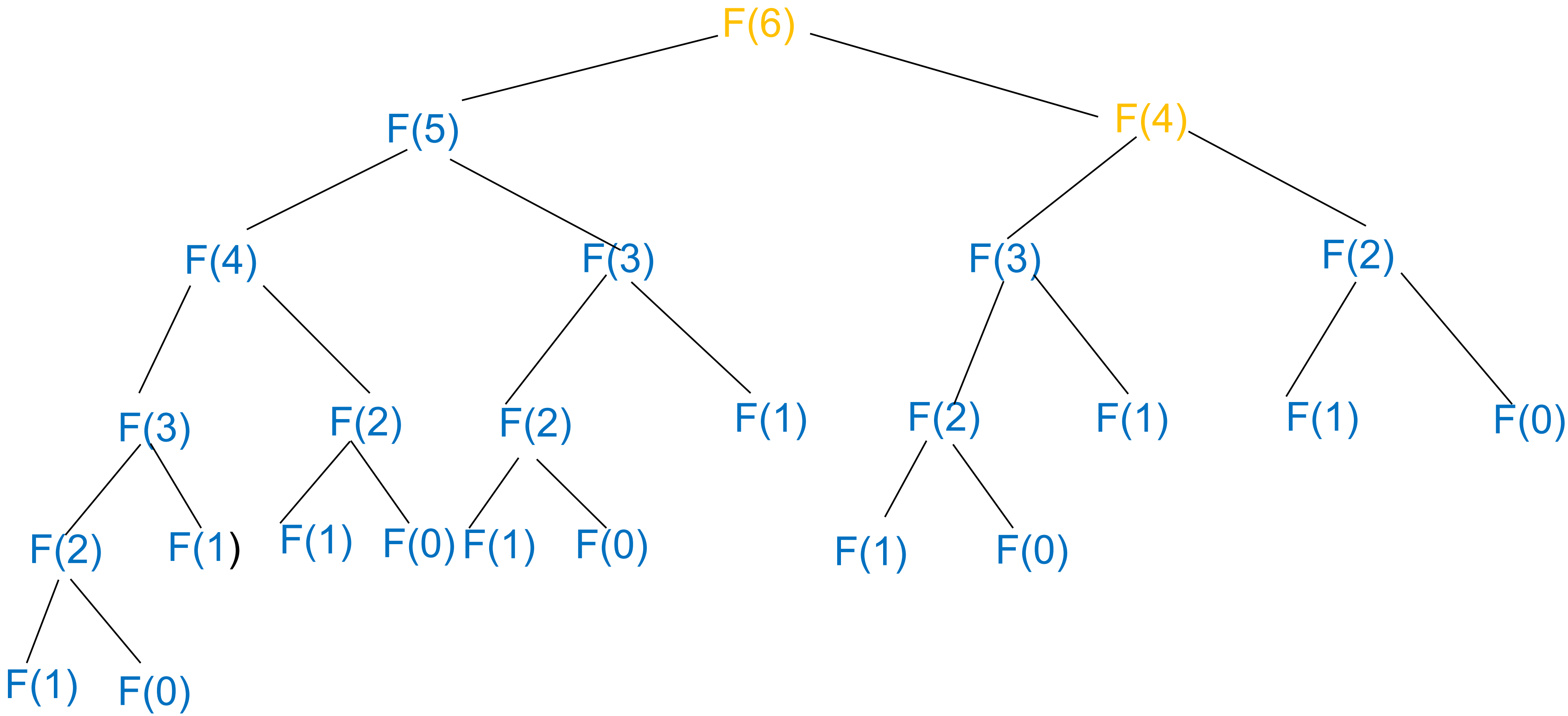
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



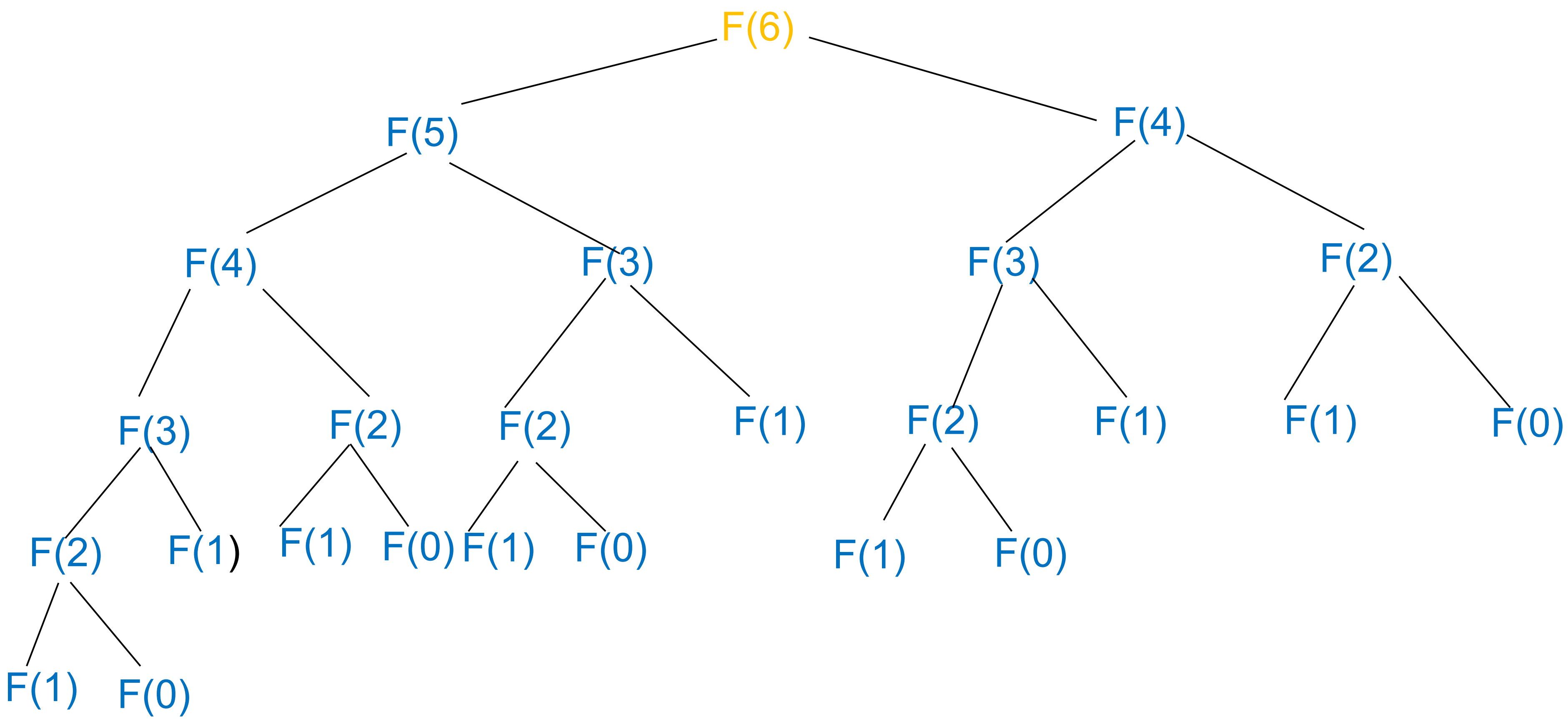
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



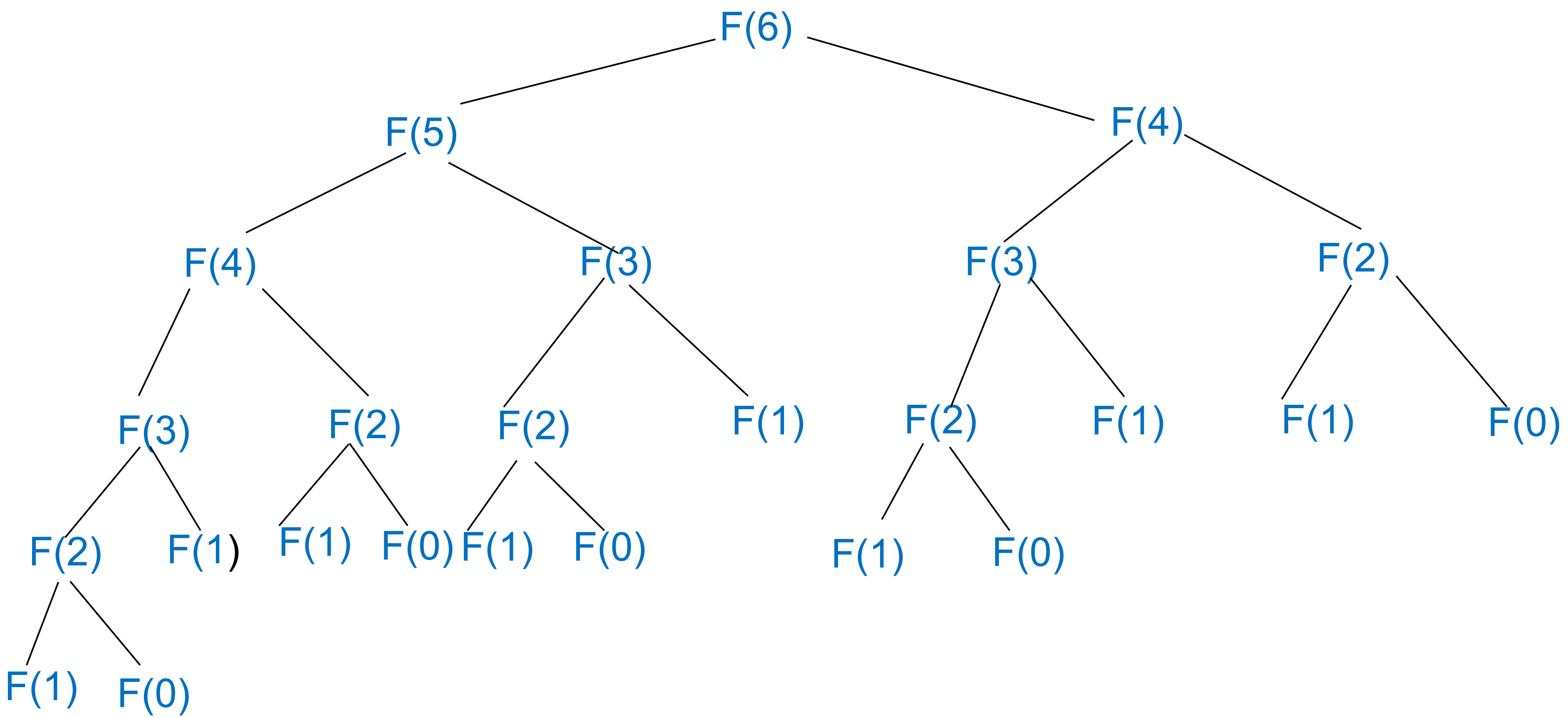
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



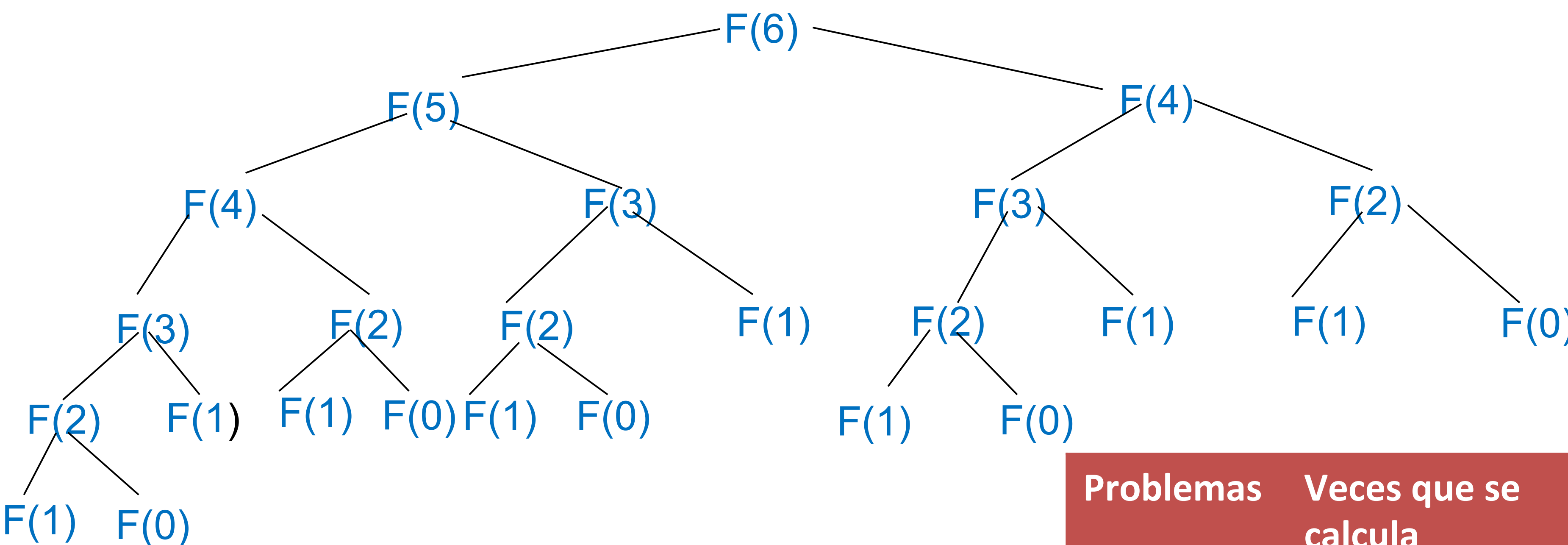
Programación dinámica

Caso Estudio: Sucesión de Fibonacci



Programación dinámica

Caso Estudio: Sucesión de Fibonacci




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

Problemas	Veces que se calcula
F(0)	5
F(1)	8
F(2)	5
F(3)	3

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

Alternativas para no calcular muchas veces la solución a un mismo subproblema:

- Estrategia top-down (Memoization): Cuando la solución del problema requiera la de un subproblema, se calcula y se almacena; así la siguiente vez que se requiera dicha solución se recupera. (Algorítmicamente se sigue partiendo del problema más grande a los más pequeños).

- Estrategia bottom-up: Se parte de la solución de los subproblemas más pequeños, las cuales van siendo almacenadas, de tal forma que un subproblema más grande las pueda usar. Finalmente se llega a la solución del problema original.

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

Algoritmo usando Memoization

```
Memoized_Fibonacci(n)
 let f[0:n] be a new array
 f[0]  = 0
 f[1]  = 1
 for i = 2  to n
      f[i] = unknown
 return  Fibo_aux(n,f)
```

```
Fibo_aux(n,f)
if f[n] == unknown
      f[n] = Fibo_aux(n-1,f) + Fibo_aux(n-2,f)

return f[n]
```

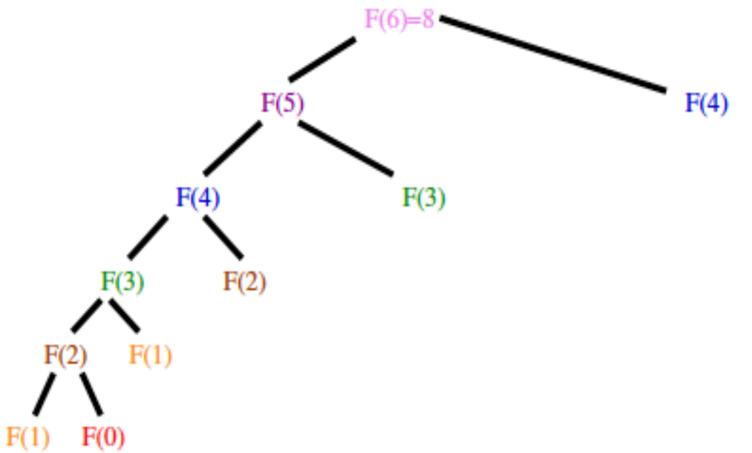
Programación dinámica

Caso Estudio: Sucesión de Fibonacci

Algoritmo usando Memoization

```
Memoized_Fibonacci(n)
 let f[0:n] be a new array
 f[0]  = 0
 f[1]   = 1
 for i = 2  to n
      f[i] = unknown
 return  Fibo_aux(n,f)
```

```
Fibo_aux(n,f)
if f[n] == unknown
      f[n] = Fibo_aux(n-1,f) + Fibo_aux(n-2,f)
return f[n]
```



Programación dinámica

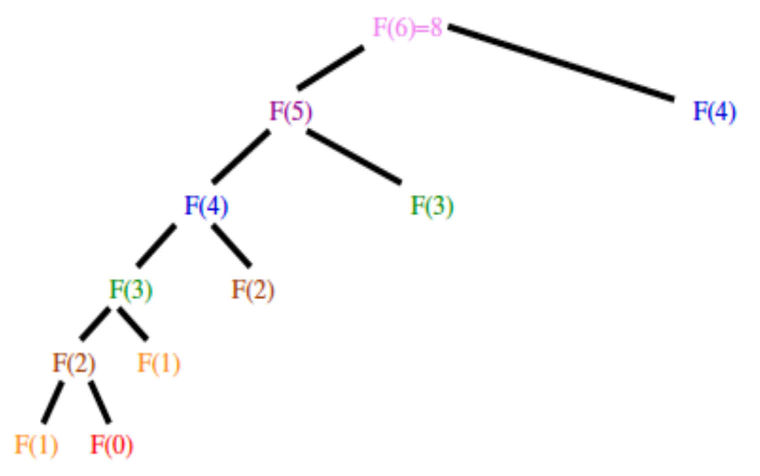
Caso Estudio: Sucesión de Fibonacci

Algoritmo usando Memoization

```
Memoized_Fibonacci(n)
 let f[0:n] be a new array
 f[0]  = 0
 f[1]   = 1
 for i = 2  to n
      f[i] = unknown
 return  Fibo_aux(n,f)
```

```
Fibo_aux(n,f)
if f[n] == unknown
      f[n] = Fibo_aux(n-1,f) + Fibo_aux(n-2,f)
return f[n]
```

¿Cuál sería su costo computacional?



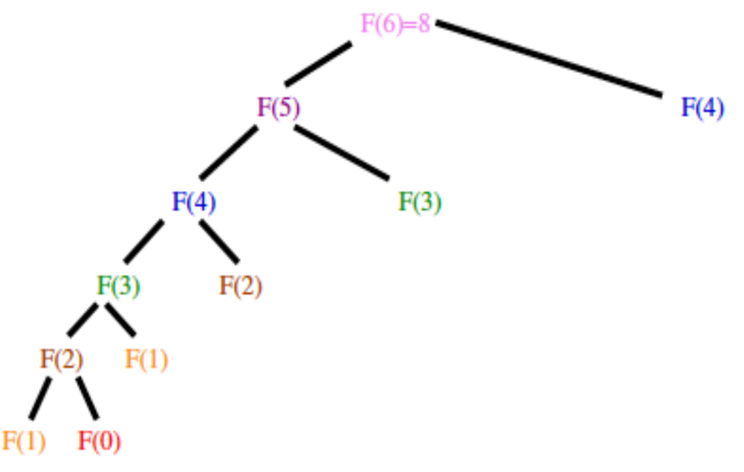
Programación dinámica

Caso Estudio: Sucesión de Fibonacci

Algoritmo usando Memoization

```
Memoized_Fibonacci(n)
 let f[0:n] be a new array
 f[0]  = 0
 f[1]   = 1
 for i = 2  to n
      f[i] = unknown
 return  Fibo_aux(n,f)
```

```
Fibo_aux(n,f)
if f[n] == unknown
       f[n] = Fibo_aux(n-1,f) + Fibo_aux(n-2,f)
return f[n]
```



¿Cuál sería su costo computacional?

[0, 1, u, u, u, u, u] -> [0, 1, 1, u, u, u, u] ->

[0, 1, 1, 2, u, u, u] -> [0, 1, 1, 2, 3, u, u] ->

[0, 1, 1, 2, 3, 5, u] -> [0, 1, 1, 2, 3, 5, 8]

Programación dinámica

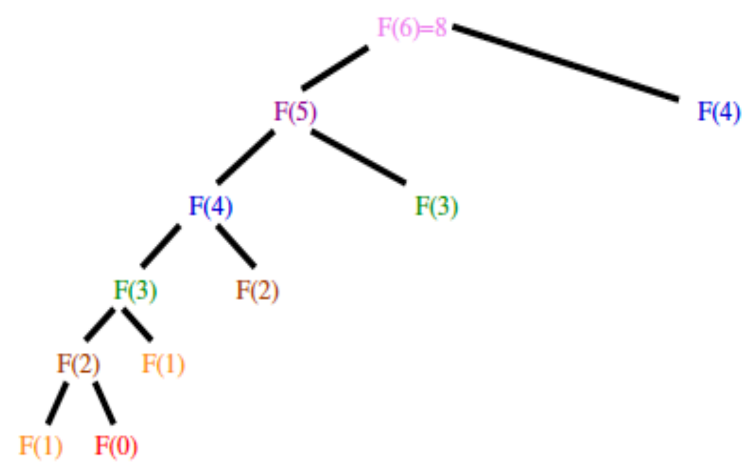
Caso Estudio: Sucesión de Fibonacci

Algoritmo usando Memoization

```
Memoized_Fibonacci(n)
 let f[0:n] be a new array
 f[0]  = 0
 f[1]  = 1
 for i = 2  to n
      f[i] = unknown
 return  Fibo_aux(n,f)
```

```
Fibo_aux(n,f)
if f[n] == unknown
      f[n] = Fibo_aux(n-1,f) + Fibo_aux(n-2,f)
return f[n]
```

Costo Computacional O(n)



Programación dinámica

Caso Estudio: Sucesión de Fibonacci

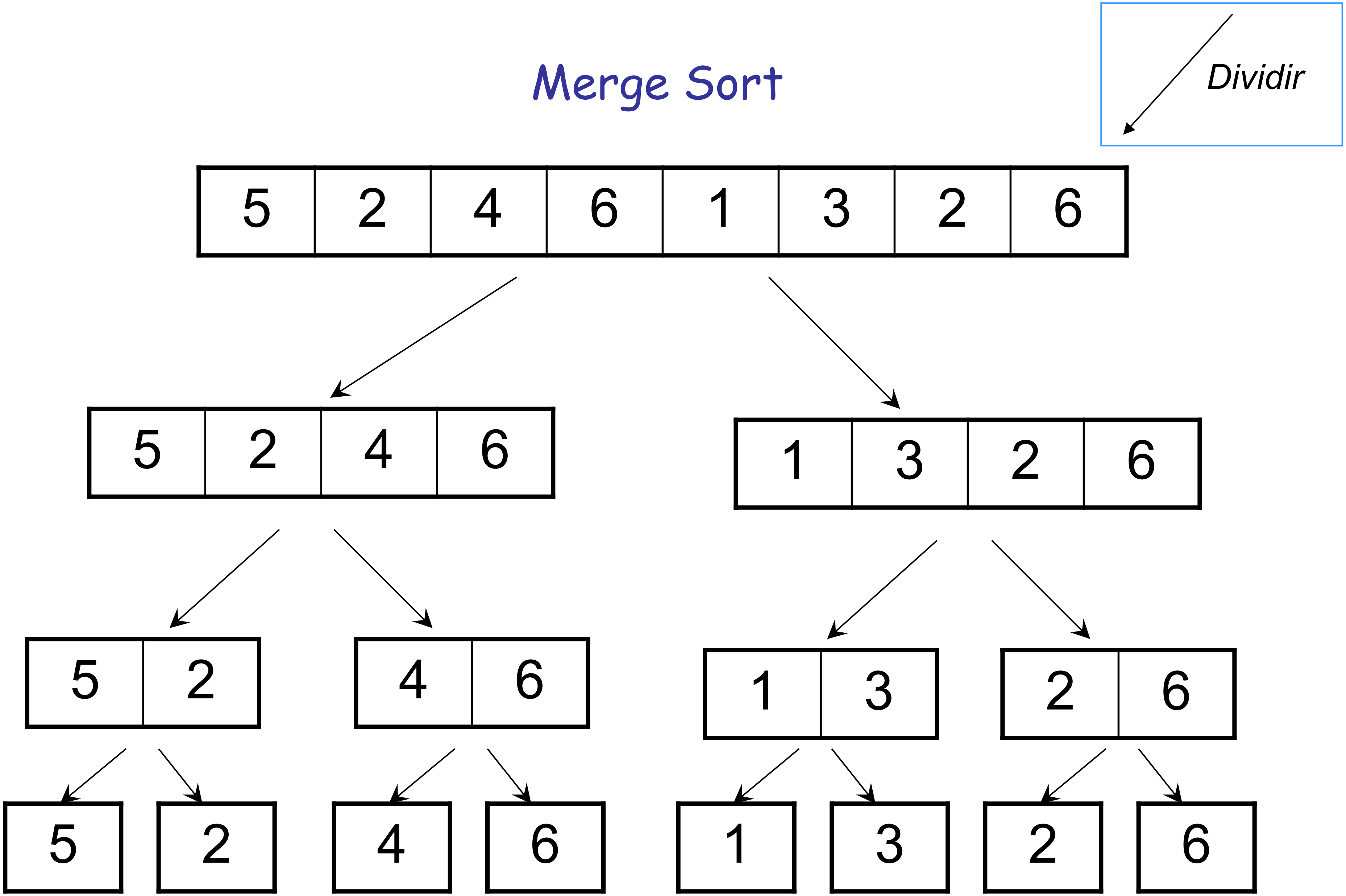
Estrategia usando Memoization

¿Podríamos aplicar esta estrategia que usó para Fibonacci en otros problemas que se pueden resolver recursivamente?

¿Se podría aplicar para el caso del Merge-sort o Quick-sort?

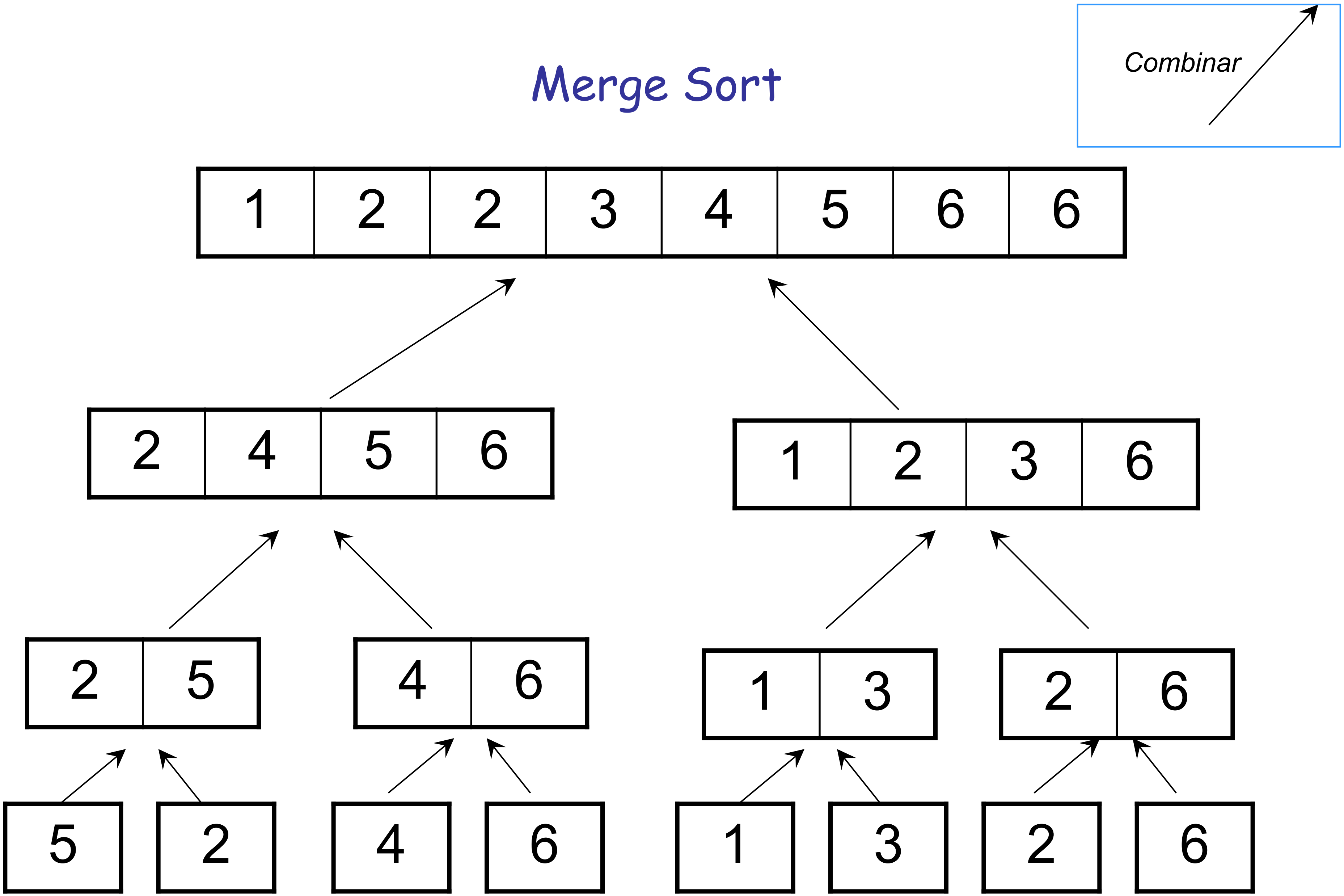
Programación Dinámica

Merge Sort



Programación dinámica

Merge Sort



Programación dinámica

Caso Estudio: Sucesión de Fibonacci

Estrategia usando Memoization

¿Podríamos aplicar esta estrategia que usó para Fibonacci en otros problemas que se pueden resolver recursivamente?

¿Se podría aplicar para el caso del Merge-sort o Quick-sort?

Para usar, con sentido, memoization es necesario que haya problemas que se solapen (overlapping problems).

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

Algoritmo usando Enfoque Bottom-up

```
DP_Fibonacci(n)
 let f[0:n] be a new array
 f[0]  = 0
 f[1]   = 1
 for i = 2  to n
      f[i] = f[i-1] + f[i-2]
 return  f[n]
```

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

Algoritmo usando Enfoque Bottom-up

```
DP_Fibonacci(n)
 let f[0:n] be a new array
 f[0]  = 0
 f[1]   = 1
 for i = 2  to n
       f[i] = f[i-1] + f[i-2]
 return  f[n]
```

¿Cuál sería su costo computacional?

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

Algoritmo usando Enfoque Bottom-up

```
DP_Fibonacci(n)
 let f[0:n] be a new array
 f[0]  = 0
 f[1]   = 1
 for i = 2  to n
      f[i] = f[i-1] + f[i-2]
 return  f[n]
```

Costo Temporal O(n)
Costo Espacial O(n)

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

Algoritmo usando Enfoque Bottom-up (reduciendo espacio)

```
DP_Fibonacci(n)
 back2 = 0
 back1 = 1
 if n == 0
        return 0
for i = 2  to n-1
        next = back1 + back2
        back2 = back1
        back1 = next

return (back1 + back2)
```

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

Algoritmo usando Enfoque Bottom-up (reduciendo espacio)

```
DP_Fibonacci(n)
 back2 = 0
 back1 = 1
 if n == 0
        return 0
for i = 2  to n-1
        next = back1 + back2
        back2 = back1
        back1 = next

return (back1 + back2)
```

¿Costo espacial?

Programación dinámica

Caso Estudio: Sucesión de Fibonacci

Algoritmo usando Enfoque Bottom-up (reduciendo espacio)

```
DP_Fibonacci(n)
 back2 = 0
 back1 = 1
 if n == 0
       return 0
for i = 2  to n-1
       next = back1 + back2
       back2 = back1
       back1 = next

return (back1 + back2)
```

¿Costo espacial?

O(1)

Programación dinámica

En las próxima sesiones veremos cómo se aplican estas estrategias para resolver eficientemente varios problemas de optimización.