CURSO DE PROGRAMACIÓN COMPETITIVA URJC - 2021

Sesión 6 (10ª Semana)

- David Morán (david.moran@urjc.es)
- Sergio Pérez (sergio.perez.pelo@urjc.es)
- Jesús Sánchez-Oro (jesus.sanchezoro@urjc.es)
- Isaac Lozano (isaac.lozano@urjc.es)
- Raúl Martín (raul.martin@urjc.es)
- Jakub Jan (jakubjanluczyn@gmail.com)
- Antonio Gonzalez (antonio.gpardo@urjc.es)
- Iván Martín (ivan.martin@urjc.es)
- Leonardo Antonio Santella (leocaracas2010@gmail.com)

Contenidos

- Programación Dinámica
  - Memoización
  - Estructura
  - Tipos de Programación Dinámica
  - Ejemplos
  - Retorno por elección (choice)

Programación Dinámica

Utilidad

- Reducir el tiempo de ejecución de un algoritmo conociendo subproblemas y sus respuestas

Principio

- Un problema se divide en 2 o más subproblemas

Programación Dinámica

- Cuando sabemos la solución del subproblema más pequeño, guardamos el resultado para su posterior reutilización (memoización)
- La solución al problema es la unión de la solución óptima de subproblemas

Programación Dinámica

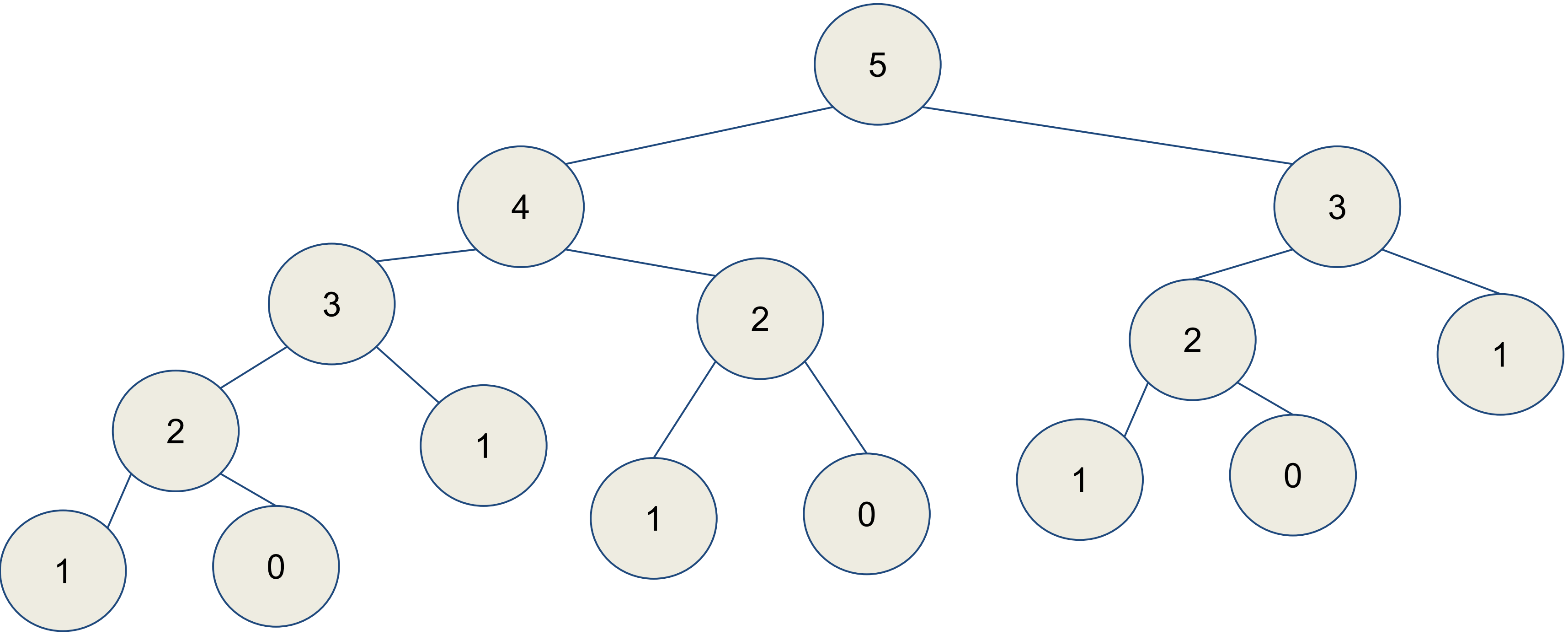
- Ejemplo: Secuencia Fibonacci
  - 1-1-2-3-5-8-13-21-34-55-89-...

```
function f(n):
  if n <= 1:
    return 1
  return f(n-1) + f(n-2)
```

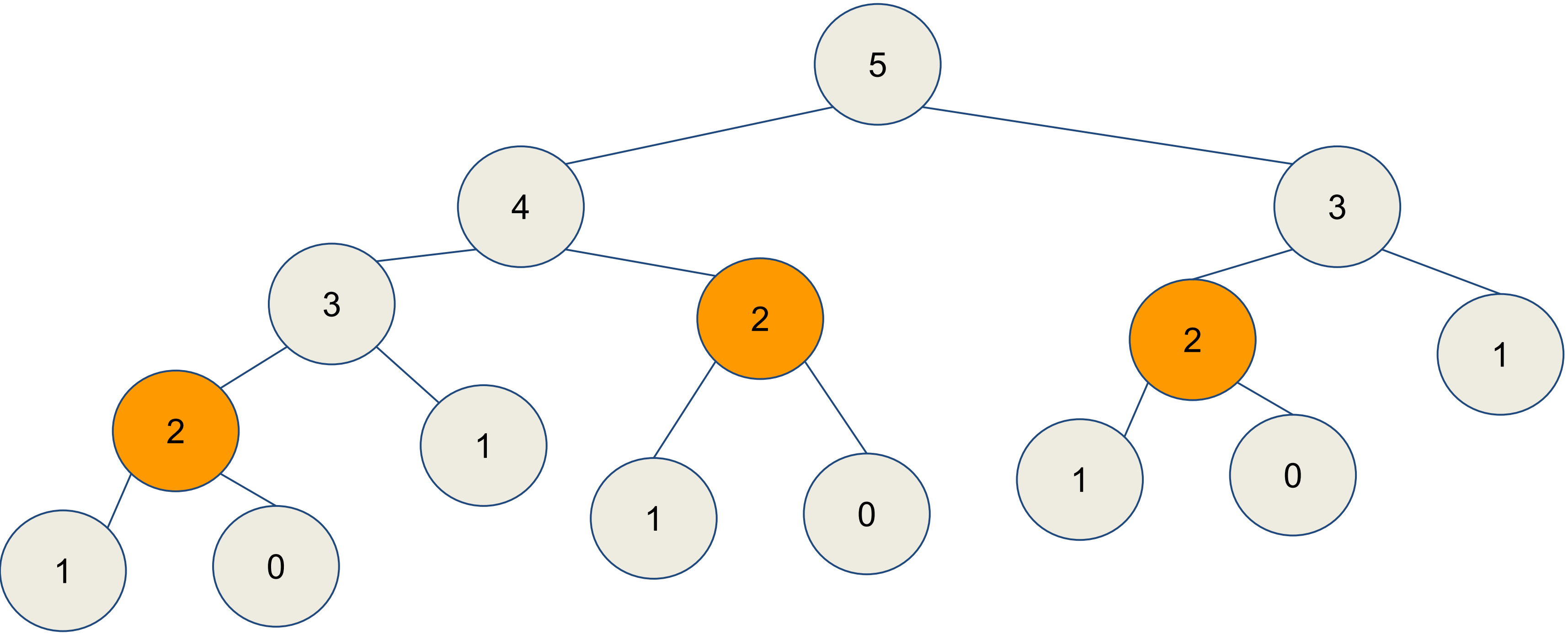
Programación Dinámica

- Por cada llamada a la función, recursivamente, la llamamos 2 veces para n-1 y n-2, esto ocurre hasta que N = 1 o menor.
- Independientemente, cada función recorre el número desde N hasta 0-1
- Al tener dos posibles llamadas, decimos que este algoritmo tiene complejidad O(2^N)

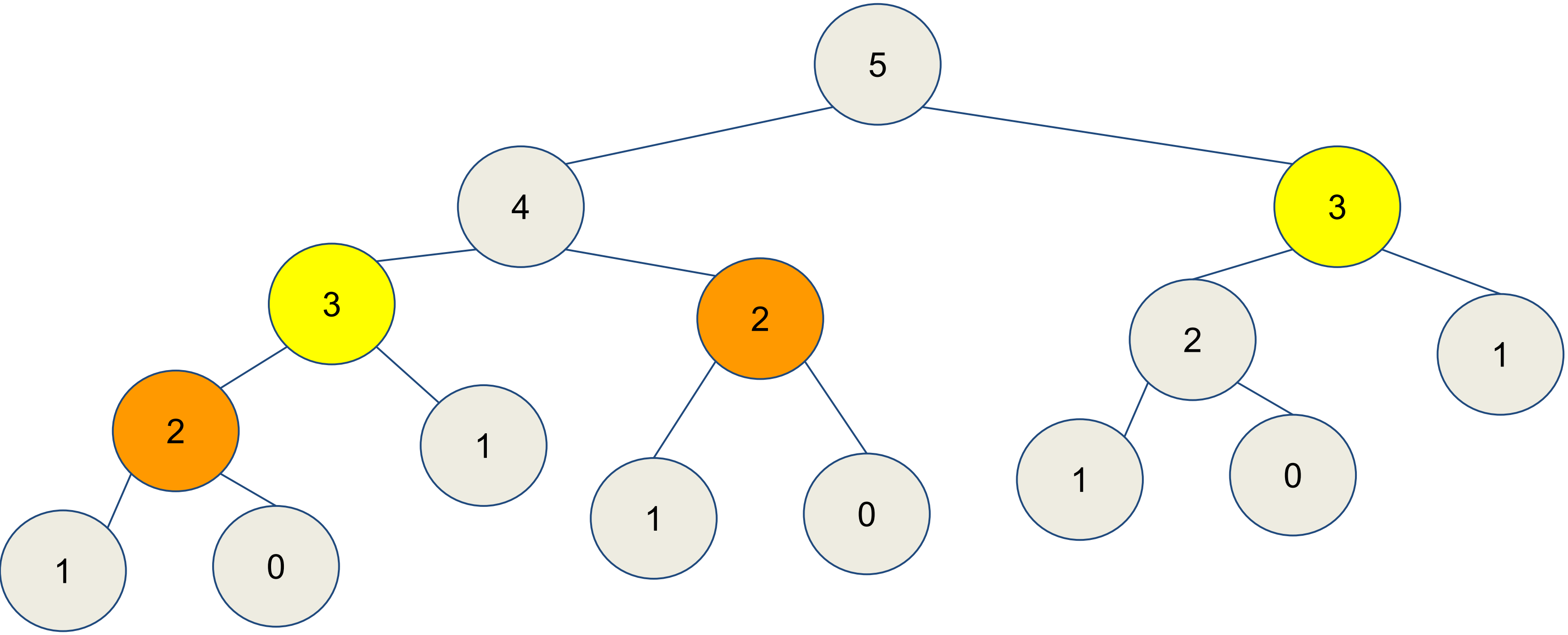
Programación Dinámica



Programación Dinámica



Programación Dinámica

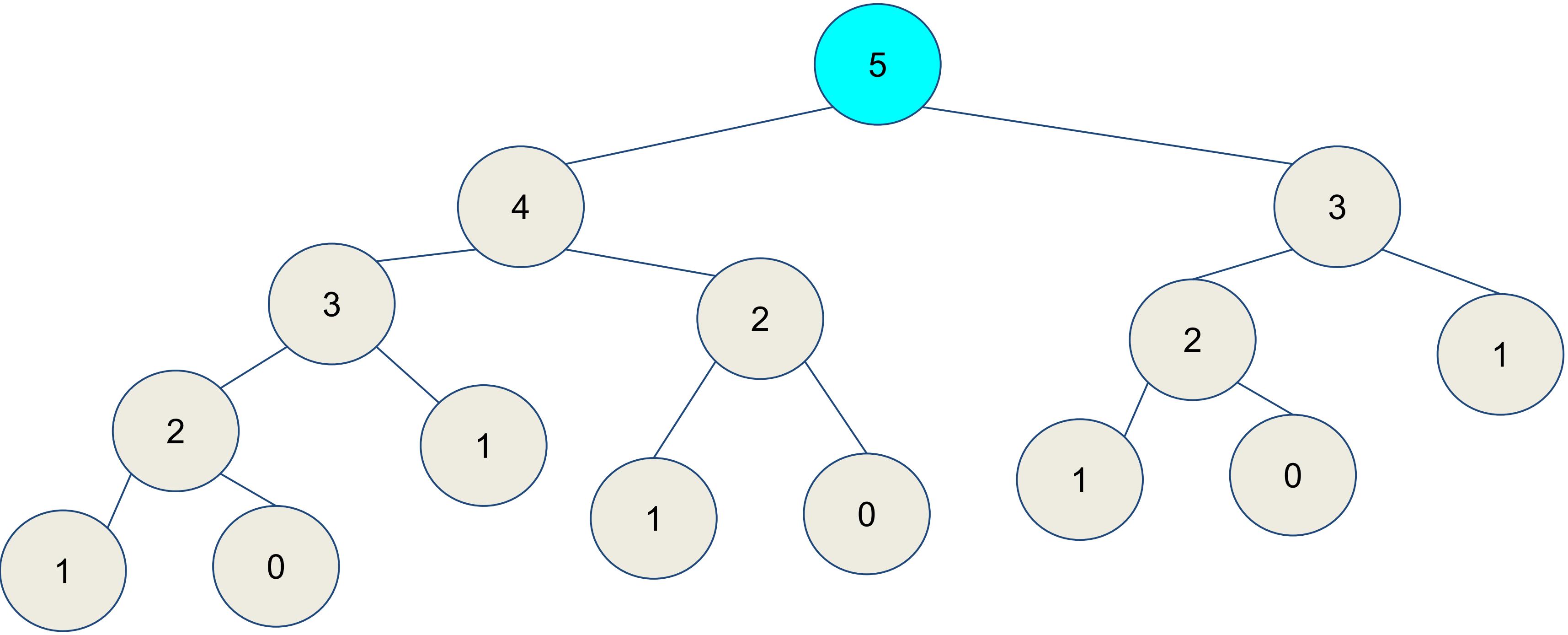


Programación Dinámica

Memoización

- Guardamos en una estructura de datos el resultado a nuestro subproblema
- La estructura **suele** ser un array N-dimensional
- Discretizar el problema (e.g. strings)

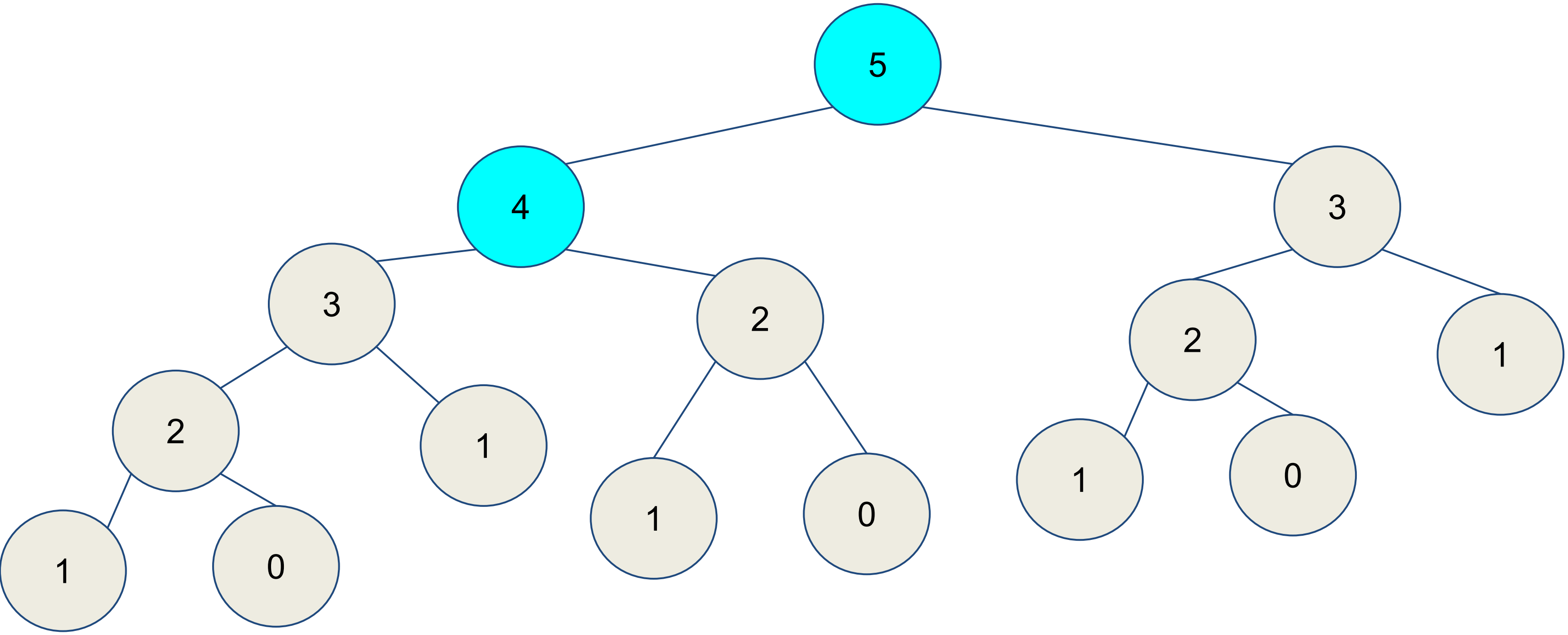
Programación Dinámica



Memo = { 1, 1, -1, -1, -1, -1 }

F(5) = F(4) + F(3)

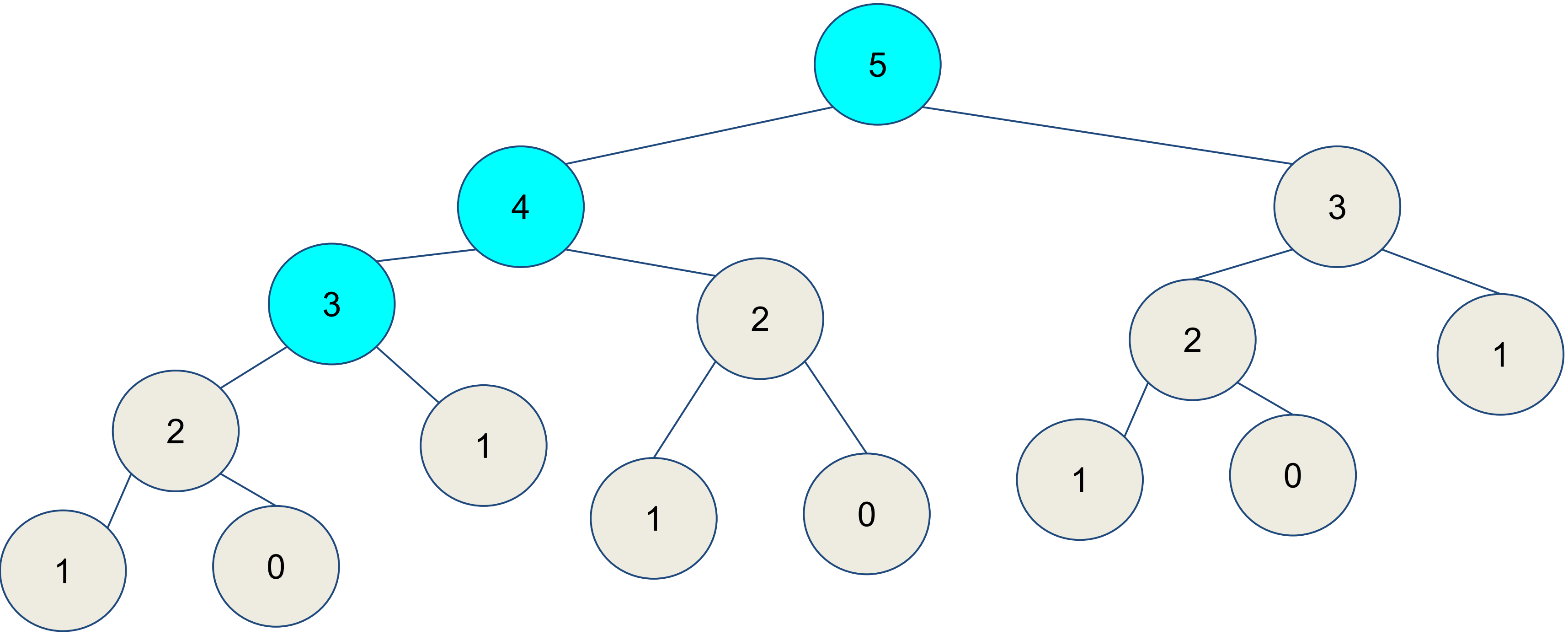
Programación Dinámica



Memo = { 1, 1, -1, -1, -1, -1 }

F(4) = F(3) + F(2)

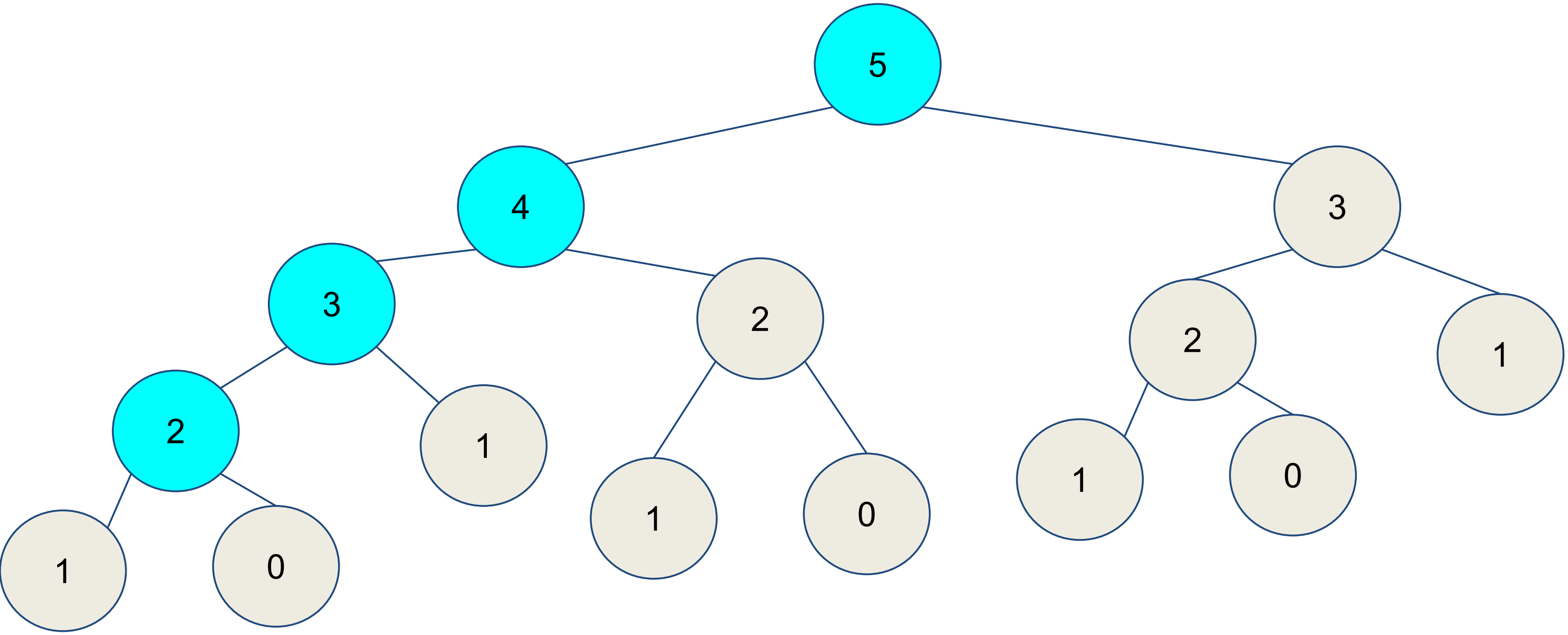
Programación Dinámica



Memo = { 1, 1, -1, -1, -1, -1 }

F(3) = F(2) + F(1)

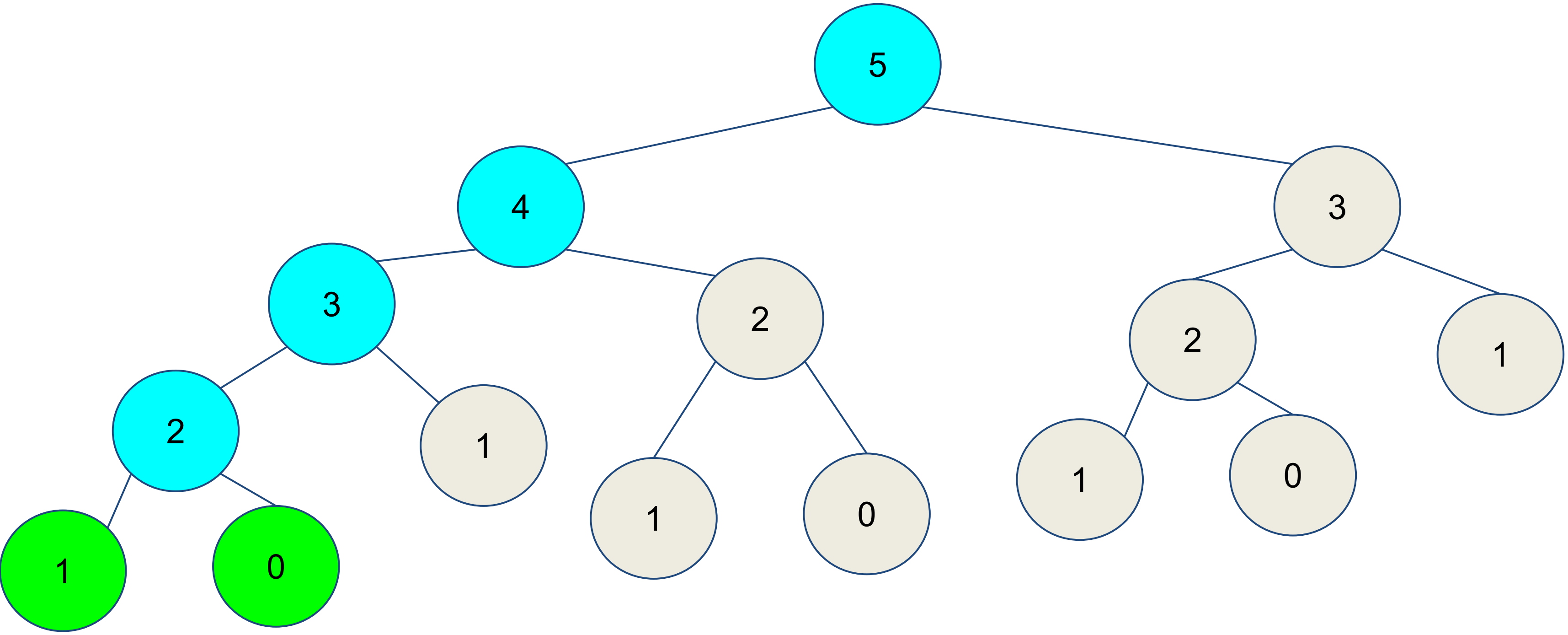
Programación Dinámica



Memo = { 1, 1, -1, -1, -1, -1 }

F(2) = F(1) + F(0)

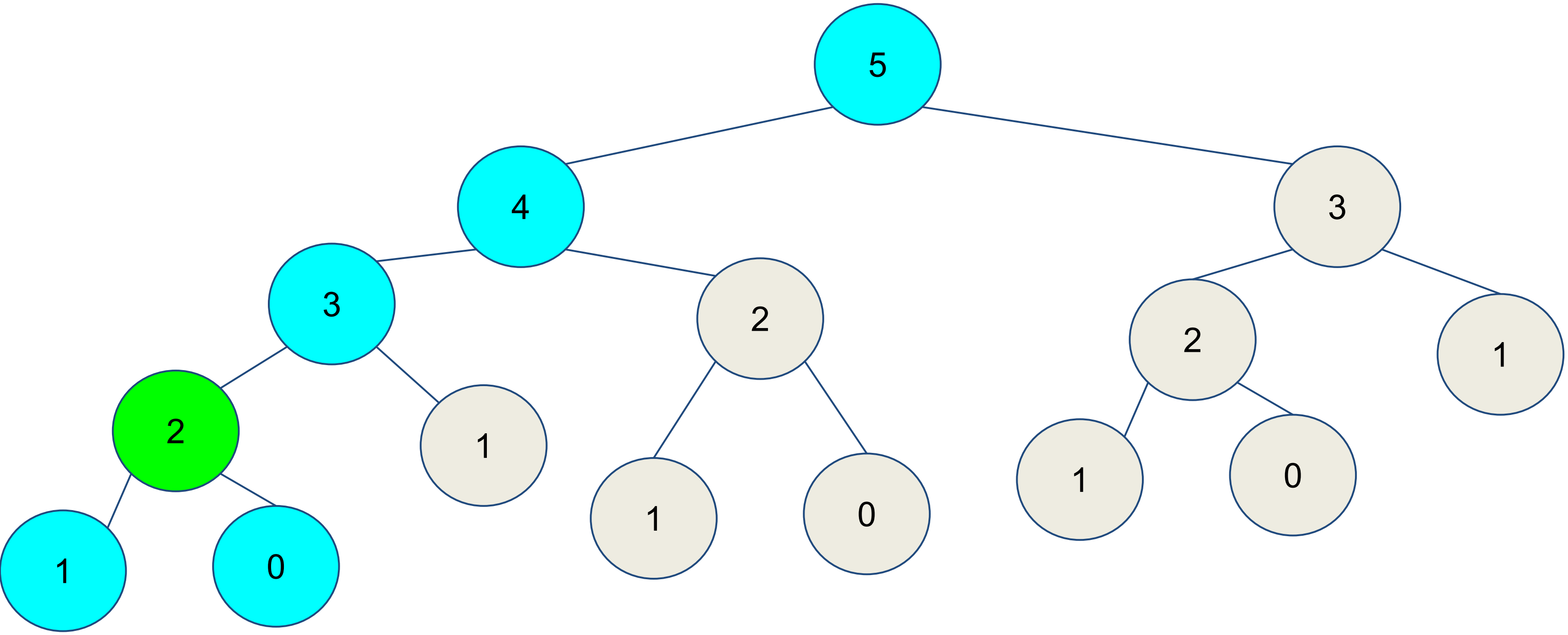
Programación Dinámica



Memo = { 1, 1, -1, -1, -1, -1 }

F(2) = 1 + 1 $\rightarrow$ 2

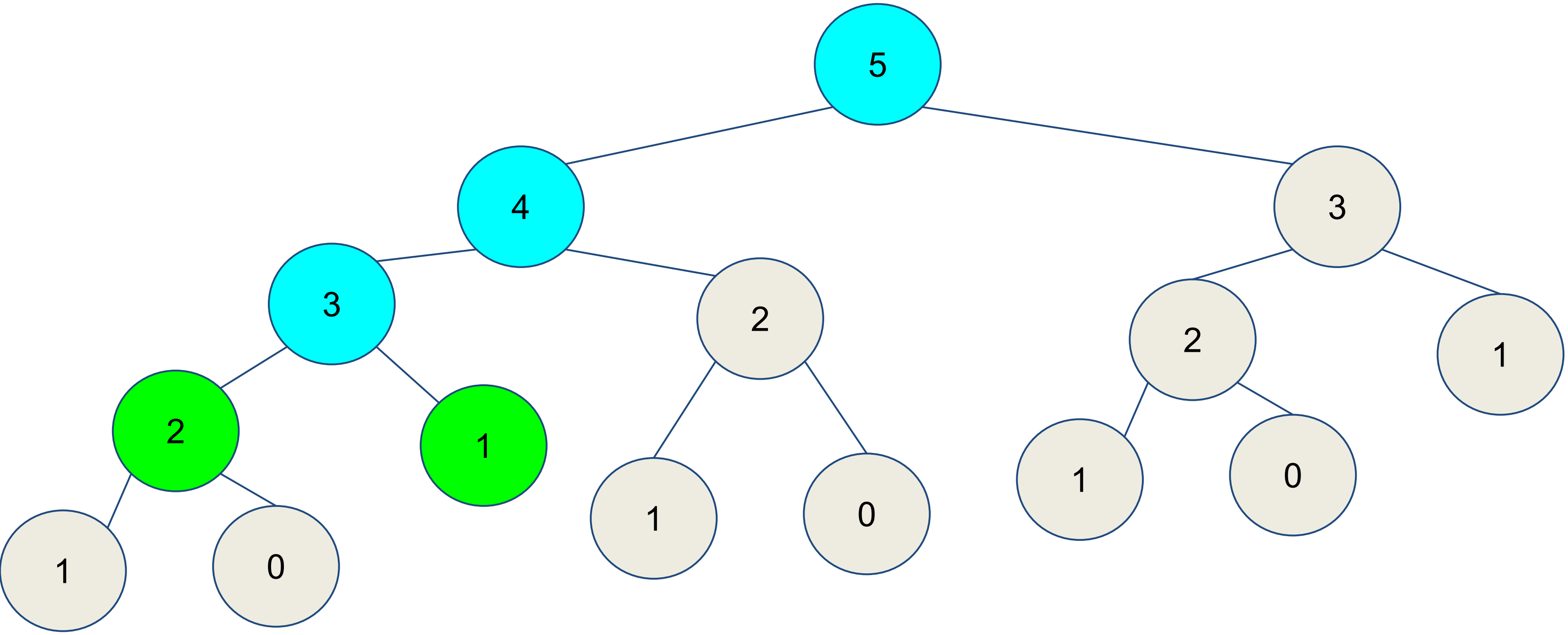
Programación Dinámica



Memo = { 1, 1, 2, -1, -1, -1 }

F(2) = 1 + 1 $\rightarrow$ 2

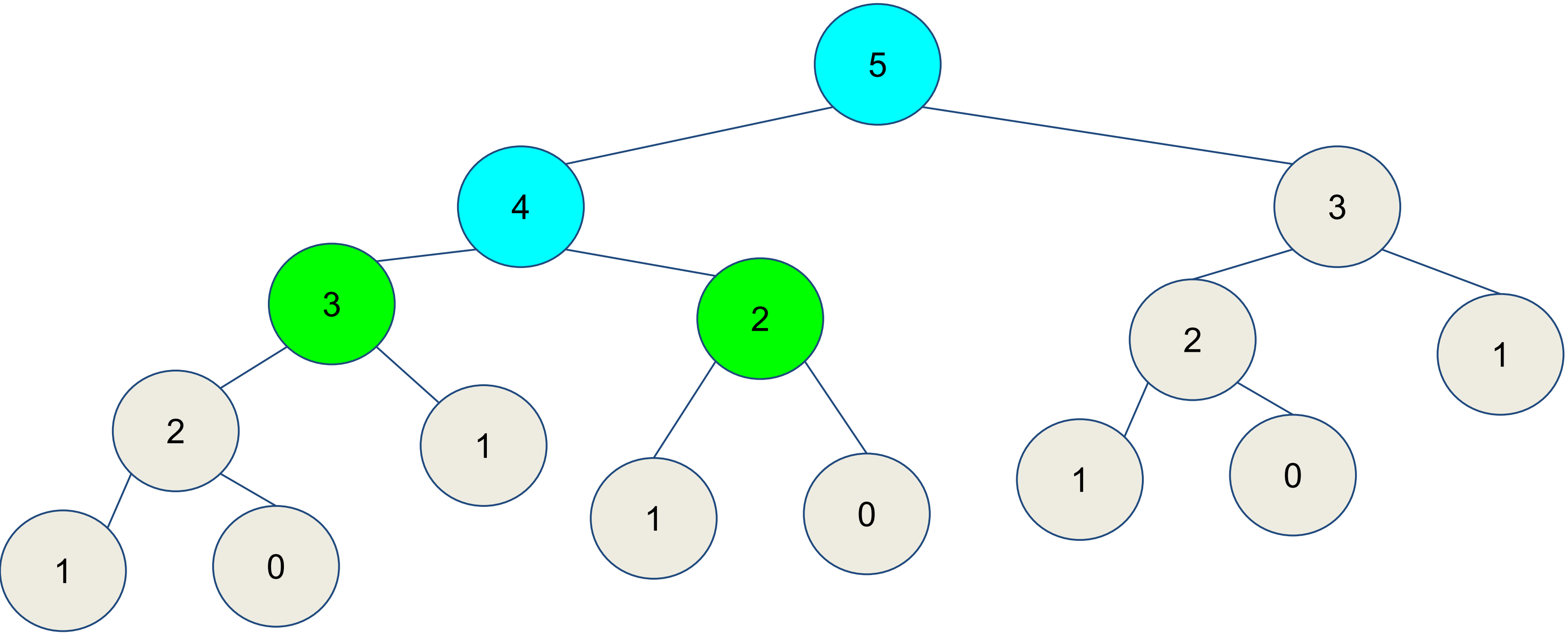
Programación Dinámica



Memo = { 1, 1, 2, 3, -1, -1 }

F(3) = 2 + 1 → 3

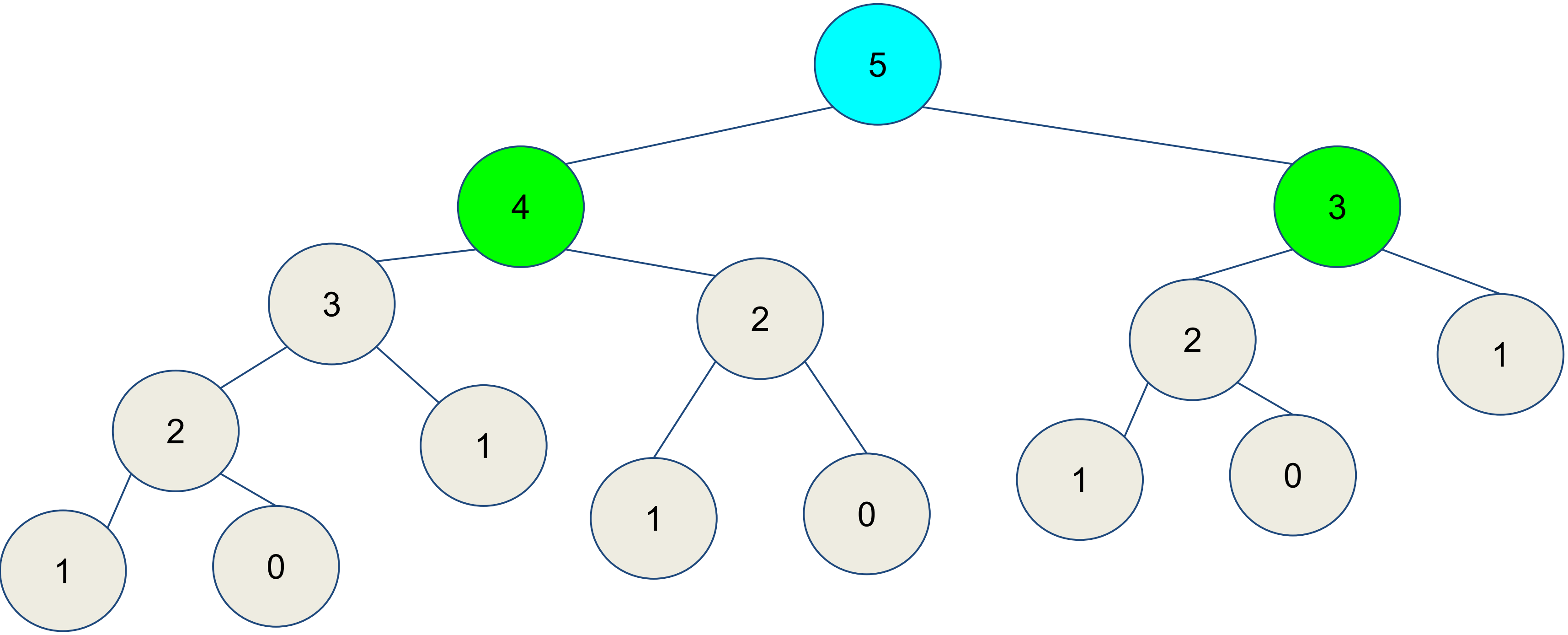
Programación Dinámica



Memo = { 1, 1, 2, 3, 5, -1 }

F(4) = 3 + 2 $\rightarrow$ 5

Programación Dinámica



Memo = { 1, 1, 2, 3, 5, 8 }

F(5) = 5 + 3 → 8

Programación Dinámica

- Por cada llamada a la función, revisamos si tenemos la respuesta al problema revisando si memo[i] != -1
- Si no es -1, tenemos solución conocida para el subproblema
- Resolvemos el subproblema y guardamos la información en memo[i] para futuros usos
- La complejidad queda reducida a O(N)

Programación Dinámica

- Algoritmo de Fibonacci + Memoización

```
function f(n):
  if n <= 1:
    return 1
  if memo[n] != -1:
    return memo[n]
  memo[n] = f(n-1) + f(n-2)
  return memo[n]
```

Programación Dinámica

Definición de términos

- **Estado**: Estado actual en el que se encuentra el DP antes de tomar cualquier decisión
- **Transición**: Cambios que deben hacerse para ir a otro subproblema
- **Memo**: Estructura donde se guarda la respuesta de los subproblemas
- **Casos base**: Subproblema principal del que sabemos respuesta

Programación Dinámica

- Algoritmo de Fibonacci + Memoización

```
function f(n):
  if n <= 1:
    return 1
  if memo[n] != -1:
    return memo[n]
  memo[n] = f(n-1) + f(n-2)
  return memo[n]
```

Programación Dinámica

Como saber que es DP

- DP solo funciona sobre DAGs (Grafos acíclicos dirigidos)
- Se requiere min/max/contar un problema concreto
- DP vs Greedy

Programación Dinámica

- Problemas ejemplo:
  - Contar de <u>cuántas</u> formas se puede llegar en la mínima cantidad de pasos (distancia manhattan) desde un punto a otro en una matriz
  - Para (0,0) → (2,2) existen 6 caminos
    1. (0,0) → (0,1) → (0,2) → (1,2) → (2,2)
    2. (0,0) → (0,1) → (1,1) → (1,2) → (2,2)
    3. (0,0) → (0,1) → (1,1) → (2,1) → (2,2)
    4. (0,0) → (1,0) → (1,1) → (1,2) → (2,2)
    5. (0,0) → (1,0) → (2,0) → (2,1) → (2,2)
    6. (0,0) → (1,0) → (1,1) → (2,1) → (2,2)

Programación Dinámica

- Detección de DP
  - ¿Se pide min/max/contar?
  - Selecciona un estado teniendo en cuenta los límites de memoria
  - ¿Los subproblemas pueden repetirse?
  - ¿Cuándo parar?

Programación Dinámica

- Detección de DP
  - ¿Se pide min/max/contar?
    - Si
  - Selecciona un estado teniendo en cuenta los límites de memoria
    - i,j con i,j >= 0 && i,j <= n,m
  - ¿Los subproblemas pueden repetirse?
    - Si, si consideramos i+1, j+1
  - ¿Cuándo parar?
    - Si i==n y j==m

Programación Dinámica

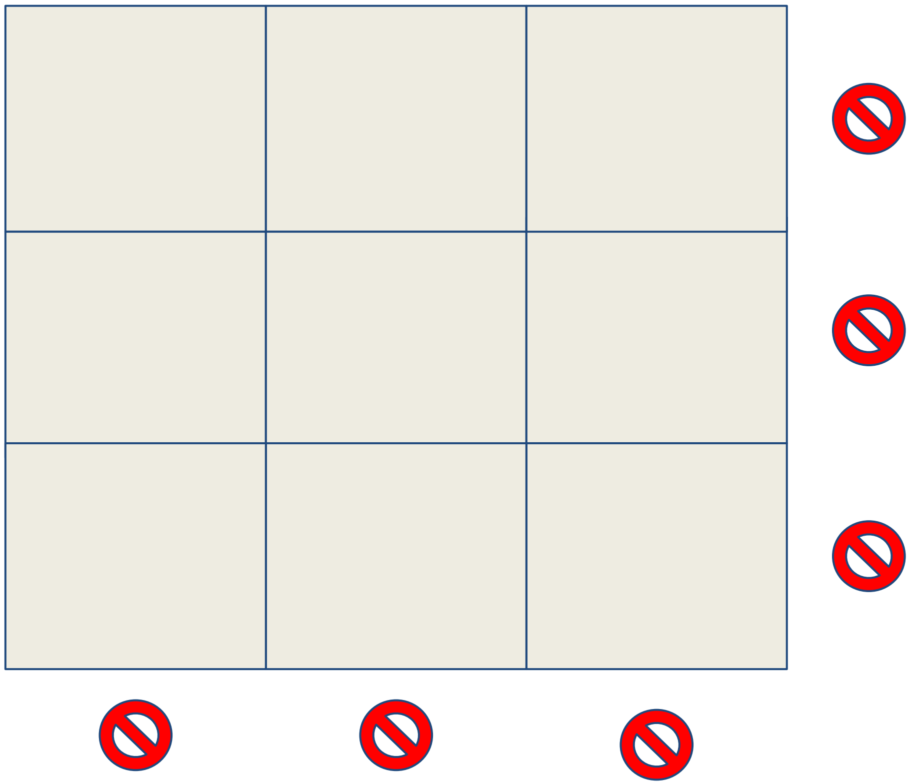
$N=3$

$F(2,2) = 1$

$F(0,0) = F(1, 0) + F(0, 1)$

$F(i, j) = F(i+1, j) + F(i, j+1)$

Programación Dinámica



N=3

F(2,2) = 1

F(0,0) = F(1, 0) + F(0, 1)

F(i, j) = F(i+1, j) + F(i, j+1)

Programación Dinámica

$N=3$

$F(2,2) = 1$

$F(<0, <0) = 0$

$F(>N, >N) = 0$

$F(0,0) = F(1, 0) + F(0, 1)$

$F(i, j) = F(i+1, j) + F(i, j+1)$

Programación Dinámica

```
function f(i, j):
  if i >= n or j >= n:
    return 0
   if i == n-1 and j == n-1:
    return 1
  if memo[i,j] != -1:
    return memo[i,j]
  memo[i,j] = f(i+1, j) + f(i, j+1)
  return memo[i,j]
```

Programación Dinámica

- Problemas ejemplo:
    - Se tienen N objetos, cada objeto tiene un peso Pi y un valor Wi, se desea colocar en una mochila objetos cuyo valor sea máximo y no exceda el peso límite de la mochila
    - Peso de mochila: <= 1000
    - Número de objetos: <= 1000
    - Valor y peso de objetos: <= 500

Programación Dinámica

- Problemas ejemplo:
  - Peso de mochila: 26
  - Objetos: 5
  - [{ valor, peso }]
  - [{ 24, 12 }, { 13, 7 }, { 23, 11 }, { 15, 8 }, { 16, 9}]

Programación Dinámica

- Problemas ejemplo:
  - Peso de mochila: 26
  - Objetos: 5
  - [{ valor, peso }]
  - [{ 24, 12 }, { 13, 7 }, { 23, 11 }, { 15, 8 }, { 16, 9}]
  - Solución voraz:
    - [{ 24, 12 }, { 23, 11 }] = { 47, 23 }

Programación Dinámica

- Problemas ejemplo:
  - Peso de mochila: 26
  - Objetos: 5
  - [{ valor, peso }]
  - [{ 24, 12 }, { 13, 7 }, { 23, 11 }, { 15, 8 }, { 16, 9}]
  - Mejor solución:
    - [{ 13, 7 }, { 23, 11 }, { 15, 8 }] = { 51, 26 }

Programación Dinámica

- Problemas ejemplo:
  - **Estado**: ?
  - **Caso Base**: ?
  - **Transición**: ?
  - **Memo**: ?

Programación Dinámica

- Problemas ejemplo:
  - **Estado**:
    - Índice del i-ésimo objeto
    - Peso que lleva la mochila
    - Valor actual (Es nuestro objetivo, lo calculamos dentro de la función)
  - **Caso Base**: ?
  - **Transición**: ?
  - **Memo**: ?

Programación Dinámica

- Problemas ejemplo:
  - **Estado**:
    - Índice del i-ésimo objeto
    - Peso que lleva la mochila
  - **Caso Base**: ?
  - **Transición**: ?
  - **Memo**: ?

Programación Dinámica

- Problemas ejemplo:
  - **Estado**: Conocido
  - **Caso Base**: ?
  - **Transición**: ?
  - **Memo**: ?

Programación Dinámica

- Problemas ejemplo:
  - **Estado**: Conocido
  - **Caso Base**:
    - Si el peso excede nuestra capacidad
    - Si el peso es igual a nuestra capacidad
    - Si no hay más objetos que escoger
  - **Transición**: ?
  - **Memo**: ?

Programación Dinámica

- Problemas ejemplo:
  - **Estado**: Conocido
  - **Caso Base**:
    - Si el peso excede nuestra capacidad (-INF)
    - Si el peso es igual a nuestra capacidad (0)
    - Si no hay más objetos que escoger (0)
  - **Transición**: ?
  - **Memo**: ?

Programación Dinámica

- Problemas ejemplo:
  - **Estado**: Conocido
  - **Caso Base**: Conocido
  - **Transición**: ?
  - **Memo**: ?

Programación Dinámica

- Problemas ejemplo:
    - **Estado**: Conocido
    - **Caso Base**: Conocido
    - **Transición**:
        - Decidir poner el objeto en la mochila
        - Ignorar y seguir al siguiente objeto
    - **Memo**: ?

Programación Dinámica

- Problemas ejemplo:
  - **Estado**: Conocido
  - **Caso Base**: Conocido
  - **Transición**:
    - Decidir poner el objeto en la mochila
      - f(i+1, j + P[i]) + V[i]
    - Ignorar y seguir al siguiente objeto
      - f(i+1, j)
  - **Memo**: ?

Programación Dinámica

- Problemas ejemplo:
  - **Estado**: Conocido
  - **Caso Base**: Conocido
  - **Transición**: Conocido
  - **Memo**: ?

Programación Dinámica

- Problemas ejemplo:
  - **Estado**: Conocido
  - **Caso Base**: Conocido
  - **Transición**: Conocido
  - **Memo**:
    - Nos quedamos con el máximo resultado que nos produzcan las dos opciones

Programación Dinámica

- Problemas ejemplo:
  - **Estado**: Conocido
  - **Caso Base**: Conocido
  - **Transición**: Conocido
  - **Memo**: Conocido
- ¡Resolvamos el problema!

Programación Dinámica

```
function f(i, weight):
  if weight > W:
    return -INF
   if weight == W or i >= N:
    return 0
  if memo[i,weight] != -1:
    return memo[i,weight]
  memo[i,weight] = max(
    f(i+1, weight + W[i]) + V[i],
    f(i+1, weight))
  return memo[i,weight]
```

Programación Dinámica

Tabulación

- Ver el algoritmo de manera iterativa más que recursiva
- Enfoque de, a partir de una solución mínima, construir una solución más grande (Bottom-Up)
- Menos intuitiva (para algunos)
- Más rápida

Programación Dinámica

Tabulación: Ejemplos

- Fibonacci

```
function f(n):
    tab[0] = 1
    tab[1] = 1
    for i from 2 to n
        tab[i] = tab[i-1] + tab[i-2]
    return tab[n]
```

Programación Dinámica

Tabulación

- El estado se convierte en índices en tantos bucles como dimensiones tenga el array de tabulación (a veces tambien llamado memo)
- La transición se hace sobre cada estado
- Los casos base se guardan explícitamente en memo

Programación Dinámica

```
function f(i, j):
  if i >= n or j >= n:
    return 0
  if i == n-1 and j == n-1:
    return 1
  if memo[i,j] != -1:
    return memo[i,j]
  memo[i,j] = f(i+1, j) + f(i, j+1)
  return memo[i,j]
```

Programación Dinámica

$N=3$

$F(2,2) = 1$

$F(<0, <0) = 0$

$F(>N, >N) = 0$

$F(0,0) = F(1, 0) + F(0, 1)$

$F(i, j) = F(i+1, j) + F(i, j+1)$

Programación Dinámica

- **Casos base**: ?
- **Estado**: ?
- **Transición**: ?
- **Tabulación**: ?

Programación Dinámica

- **Casos base**:
  - tab[0, 0] = 1 (si partimos del origen, es válido)
  - if row < 0 or col < 0 = 0
- **Estado**: ?
- **Transición**: ?
- **Tabulación**: ?

Programación Dinámica

- **Casos base**: Conocido
- **Estado**: ?
- **Transición**: ?
- **Tabulación**: ?

Programación Dinámica

- **Casos base**: Conocido
- **Estado**: row, col (para cada celda de la matriz)
- **Transición**: ?
- **Tabulación**: ?

Programación Dinámica

- **Casos base**: Conocido
- **Estado**: Conocido
- **Transición**: ?
- **Tabulación**: ?

Programación Dinámica

- **Casos base**: Conocido
- **Estado**: Conocido
- **Transición**:
  - tab[i-1, j] y tab[i, j-1] (Tener cuidado con los índices)
- **Tabulación**: ?

Programación Dinámica

- **Casos base**: Conocido
- **Estado**: Conocido
- **Transición**: Conocido
- **Tabulación**: La suma de ambas transiciones

Programación Dinámica

- **Casos base**: Conocido
- **Estado**: Conocido
- **Transición**: Conocido
- **Tabulación**: Conocido

Programación Dinámica

```
function f(n):
  tab[0, 0] = 1
  for row from 0 to n:
    for col from 0 to n:
      prev_row = 0
      prev_col = 0
      if col-1 >= 0:
        prev_col = tab[row, col - 1]
      if row-1 >= 0:
        prev_row = tab[row - 1, col]
      tab[row, col] = prev_row + prev_col
  return tab[n-1, n-1]
```

Programación Dinámica

- Retorno con choice (elección)
- Nos pueden pedir que construyamos una solución que nos lleve a una respuesta óptima del problema (min/max)
- Dicha construcción es más intuitiva de hacer con la técnica de Bottom-Up
- También se puede hacer con Top-Down

Programación Dinámica

- Necesitamos una estructura adicional exactamente igual a **memo** que nos guarde a que estado hemos tomado la decisión
- Requiere condicionales sobre qué llamada es mejor que otra
- Si A es mejor respuesta que B, definimos en nuestra elección del estado $E \rightarrow T_A$

Programación Dinámica

- Problema de la mochila
  - choice[objeto][peso] = ?
  - Ta = F(objeto + 1, peso)
  - Tb = F(objeto + 1, peso - P[objeto]) + V[objeto]
  - Si Ta > Tb → choice[objeto][peso] = (objeto+1, peso)
  - Sino → choice[objeto][peso] = (objeto+1, peso-P[objeto])

Programación Dinámica

```
function f(i, weight):
  if weight > W:
    return -INF
   if weight == W or i >= N:
    return 0
  if memo[i,weight] != -1:
    return memo[i,weight]
  memo[i,weight] = max(
    f(i+1, weight + W[i]) + V[i],
    f(i+1, weight))
  return memo[i,weight]
```

Programación Dinámica

```
function f(i, weight):
    if weight > W:
        return -INF
    if weight == W or i >= N:
        return 0
    if memo[i,weight] != -1:
        return memo[i,weight]
    T1 = f(i+1, weight + W[i]) + V[i]
    T2 = f(i+1, weight)
    if T1 > T2:
        memo[i, weight] = T1
        choice[i, weight] = (i+1, weight + W[i])
    else:
        memo[i, weight] = T2
        choice[i, weight] = (i+1, weight)
    return memo[i,weight]
```

Programación Dinámica

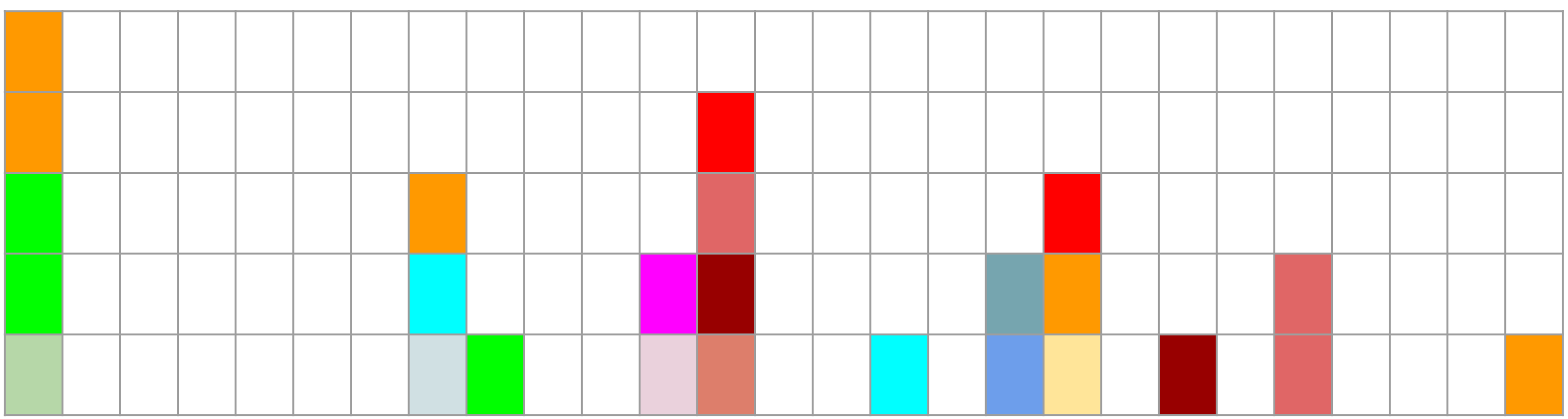
- Problemas ejemplo:
  - Peso de mochila: 26
  - Objetos: 5
  - [{ valor, peso }]
  - [{ 24, 12 }, { 13, 7 }, { 23, 11 }, { 15, 8 }, { 16, 9}]

Programación Dinámica

- Problemas de la mochila:
  - Choice tendrá tamaño 27x5 (pesos x índice)

Programación Dinámica

- Problemas de la mochila:
  - Choice tendrá tamaño 27x5 (pesos x índice)



Programación Dinámica

- Interpretando resultados de choice
  - Choice[0,0] → Choice[1,0]
  - Choice[1,0] → Choice[2,7]
  - Choice[2,7] → Choice[3,18]
  - Choice[3,18] → Choice[4,26]
  - Choice[4,26] → Choice[5,26]
- Si el segundo estado (las columnas) no cambian entre transición, no tomamos ese elemento (nuestro peso se mantiene igual)

Semana que viene

- Concurso final

Fin de Curso!

Hasta el 2022

Ante cualquier duda sobre el curso o sobre los problemas podéis escribirnos (preferiblemente con copia a algunos / todos los docentes)

- David Morán (david.moran@urjc.es)
- Sergio Pérez (sergio.perez.pelo@urjc.es)
- Jesús Sánchez-Oro (jesus.sanchezoro@urjc.es)
- Isaac Lozano (isaac.lozano@urjc.es)
- Raúl Martín (raul.martin@urjc.es)
- Jakub Jan (jakubjanluczyn@gmail.com)
- Antonio Gonzalez (antonio.gpardo@urjc.es)
- Iván Martín (ivan.martin@urjc.es)
- Leonardo Antonio Santella (leocaracas2010@gmail.com)