# Programación Dinámica Bottom-up y Reconstrucción de Solución

Eric Brandwein Fernando Frassia Ferrari

1er Cuatrimestre 2024, TN

# 📝 Conceptitos

- ⬇️ **Top-down:** Ir de lo **más general** a lo **más específico**.
- ⬆️ **Bottom-up:** Ir de lo **más específico** a lo **más general**.

# Fibonacci

## Problema

Dado un entero no negativo $n$, obtener el $n$-ésimo número de la secuencia de Fibonacci.

# La formulita

## Problema

Dado un entero no negativo $n$, obtener el $n$-ésimo número de la secuencia de Fibonacci.

$$fib(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ fib(n-1) + fib(n-2) & \text{sino} \end{cases}$$

# Backtracking

$$fib(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ fib(n-1) + fib(n-2) & \text{sino} \end{cases}$$

# Backtracking

$$fib(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ fib(n-1) + fib(n-2) & \text{sino} \end{cases}$$

```python
def fib(n: int) -> int:
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

# Backtracking

$$fib(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ fib(n-1) + fib(n-2) & \text{sino} \end{cases}$$

```python
def fib(n: int) -> int:
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

¿Complejidad temporal?

# Backtracking

$$fib(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ fib(n-1) + fib(n-2) & \text{sino} \end{cases}$$

```python
def fib(n: int) -> int:
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

¿Complejidad temporal? $\mathcal{O}(2^n)$.

¿Complejidad espacial?

# Backtracking

$$fib(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ fib(n-1) + fib(n-2) & \text{sino} \end{cases}$$

```python
def fib(n: int) -> int:
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

- ¿Complejidad temporal? $\mathcal{O}(2^n)$.
- ¿Complejidad espacial? $\mathcal{O}(n)$.

# ⬇️ Top-down

# ⬇️ Top-down

```python
def fib(n: int, memoria: list = None) -> int:
    if memoria is None:
        memoria = [-1] * (n + 1)

    if memoria[n] == -1:
        if n <= 1:
            memoria[n] = n
        else:
            memoria[n] = \
                fib(n - 1, memoria) + fib(n - 2, memoria)

    return memoria[n]
```

# ⬇️ Top-down

```python
def fib(n: int, memoria: list = None) -> int:
    if memoria is None:
        memoria = [-1] * (n + 1)

    if memoria[n] == -1:
        if n <= 1:
            memoria[n] = n
        else:
            memoria[n] = \
                fib(n - 1, memoria) + fib(n - 2, memoria)

    return memoria[n]
```

🕑 ¿Complejidad temporal?

# Top-down

```python
def fib(n: int, memoria: list = None) -> int:
    if memoria is None:
        memoria = [-1] * (n + 1)

    if memoria[n] == -1:
        if n <= 1:
            memoria[n] = n
        else:
            memoria[n] = \
                fib(n - 1, memoria) + fib(n - 2, memoria)

    return memoria[n]
```

- ¿Complejidad temporal? $\mathcal{O}(n)$.
- ¿Complejidad espacial?

# ⬇️ Top-down

```python
def fib(n: int, memoria: list = None) -> int:
    if memoria is None:
        memoria = [-1] * (n + 1)

    if memoria[n] == -1:
        if n <= 1:
            memoria[n] = n
        else:
            memoria[n] = \
                fib(n - 1, memoria) + fib(n - 2, memoria)

    return memoria[n]
```

🕑 ¿Complejidad temporal? $\mathcal{O}(n)$.

🛸 ¿Complejidad espacial? $\mathcal{O}(n)$.

# ⬆️ Bottom-up

# Bottom-up

```
def fib(n: int) -> int:
    memoria = [-1] * (n + 1)
    memoria[0] = 0
    memoria[1] = 1

    for i in range(2, n + 1):
        memoria[i] = memoria[i-1] + memoria[i-2]

    return memoria[n]
```

# ⬆️ Bottom-up

```python
def fib(n: int) -> int:
    memoria = [-1] * (n + 1)
    memoria[0] = 0
    memoria[1] = 1

    for i in range(2, n + 1):
        memoria[i] = memoria[i-1] + memoria[i-2]

    return memoria[n]
```

🕑 ¿Complejidad temporal?

# Bottom-up

```python
def fib(n: int) -> int:
    memoria = [-1] * (n + 1)
    memoria[0] = 0
    memoria[1] = 1

    for i in range(2, n + 1):
        memoria[i] = memoria[i-1] + memoria[i-2]

    return memoria[n]
```

- ¿Complejidad temporal? $\mathcal{O}(n)$.
- ¿Complejidad espacial?

# Bottom-up

```python
def fib(n: int) -> int:
    memoria = [-1] * (n + 1)
    memoria[0] = 0
    memoria[1] = 1

    for i in range(2, n + 1):
        memoria[i] = memoria[i-1] + memoria[i-2]

    return memoria[n]
```

- ¿Complejidad temporal? $\mathcal{O}(n)$.
- ¿Complejidad espacial? $\mathcal{O}(n)$.

# Bottom-up

```python
def fib(n: int) -> int:
    memoria = [-1] * (n + 1)
    memoria[0] = 0
    memoria[1] = 1

    for i in range(2, n + 1):
        memoria[i] = memoria[i-1] + memoria[i-2]

    return memoria[n]
```

- ¿Complejidad temporal? $\mathcal{O}(n)$.
- ¿Complejidad espacial? $\mathcal{O}(n)$.
- ¿Se puede mejorar?

# ⬆️ Bottom-up 2 – *menos memoria que Eric en final*™

```python
def fib(n: int) -> int:
    anterior = 0
    resultado = 1

    for _ in range(2, n + 1):
        nuevo = resultado + anterior
        anterior = resultado
        resultado = nuevo

    return resultado
```

# ⬆️ Bottom-up 2 – *menos memoria que Eric en final*™

```python
def fib(n: int) -> int:
    anterior = 0
    resultado = 1

    for _ in range(2, n + 1):
        nuevo = resultado + anterior
        anterior = resultado
        resultado = nuevo

    return resultado
```

🕑 ¿Complejidad temporal?

# ⬆️ Bottom-up 2 – *menos memoria que Eric en final*™

```python
def fib(n: int) -> int:
    anterior = 0
    resultado = 1

    for _ in range(2, n + 1):
        nuevo = resultado + anterior
        anterior = resultado
        resultado = nuevo

    return resultado
```

- 🕑 ¿Complejidad temporal? $\mathcal{O}(n)$.
- 🛸 ¿Complejidad espacial?

# ⬆️ Bottom-up 2 – *menos memoria que Eric en final*™

```python
def fib(n: int) -> int:
    anterior = 0
    resultado = 1

    for _ in range(2, n + 1):
        nuevo = resultado + anterior
        anterior = resultado
        resultado = nuevo

    return resultado
```

🕑 ¿Complejidad temporal? $\mathcal{O}(n)$.

🛸 ¿Complejidad espacial? $\mathcal{O}(1)$.

# ↔️ Diferencias

# ↔️ Diferencias

## ⬇️ Top-down

- Recursivo en general ☎️
- A veces más fácil de programar (agarrás el backtracking, le agregás memorización y listo el 🍗)

## ⬆️ Bottom-up

- Iterativo en general 🔃
- A veces usa menos memoria 💿
- A veces más rápido en la práctica (recursión vs. iteración) ⏱️

# Caminos en una matriz

## Problema

Se tiene una matriz $M$ de $m \times n$ números naturales. Se quiere llegar desde la posición $(0,0)$ de la matriz a la $(m-1, n-1)$, y sólo se pueden hacer movimientos para abajo o para la derecha en cada paso. Calcular el camino de la posición $(0,0)$ a la $(m-1, n-1)$ cuya suma de casilleros sea mínima.

# 🤔 ¿Cómo se piensan estos problemas?

1. Dibujar ejemplos.
2. Identificar casos borde.
3. Definir una función de recurrencia f.
4. Demostrar que efectivamente f resuelve el problema.
5. Implementarla y deducir la complejidad.

# Ejemplo

| | |
|---|---|
| 1 | 5 |
| 2 | 3 |

Camino 1 = [(0,0), (0,1), (1,1)], Sum(Camino 1) = 9
Camino 2 = [(0,0), (1,0), (1,1)], Sum(Camino 2) = 6

# Caminos en una matriz

## Definición

Un **camino** $P$ en la matriz $M$ desde el casillero $(a, b)$ al casillero $(a', b')$ es una secuencia de tamaño $l$ de tuplas $(i, j)$ con $i \in \{0, \ldots, m-1\}$ y $j \in \{0, \ldots, n-1\}$ que cumple las siguientes condiciones:

- $P_0$.fila $= a$ y $P_0$.col $= b$.
- $P_l$.fila $= a'$ y $P_l$.col $= b'$.
- Para todo $k \in \{2, \ldots, l\}$:
  - $P_k = P_{k-1} + (1, 0)$, o
  - $P_k = P_{k-1} + (0, 1)$.

# Caminos en una matriz

## Definición

El **costo** de un camino en la matriz $M$ es la suma de los valores $M_{fila_i, columna_i}$ para todo $i \in \{1, \ldots, l\}$.

# Caminos en una matriz – traducción

**Problema**

Se tiene una matriz $M$ de $m \times n$ números naturales. ¿Cuál es el camino desde el casillero $(0, 0)$ al $(m-1, n-1)$ que tiene menor costo?

# Principio de Optimalidad

- Las **partes** de una **solución óptima** a un problema, deben ser **soluciones óptimas** de los correspondientes **subproblemas**.
- Permite obtener una solución óptima al problema original a partir de soluciones óptimas de los subproblemas.

# 🥇 Principio de Optimalidad

**Lema**

Sea $P$ un camino de mínimo costo en una matriz $M$ entre dos posiciones diferentes $(a, b)$ y $(c, d)$. Sea $Q$ un sufijo de $P$, y sea $(a', b')$ el primer elemento de $Q$. El camino $Q$ es de mínimo costo entre todos los caminos desde $(a', b')$ hasta $(c, d)$.

# Demostración Lema

Veámoslo por absurdo: Sea $P$ un camino de costo mínimo de $(a, b)$ a $(c, d)$, y $Q$ un sufijo de $P$ que empieza en $(a', b')$. Supongamos que $Q$ no es mínimo, y entonces existe un camino $Q'$ que va de $(a', b')$ a $(c, d)$ y tiene menor costo que $Q$. Llamemos $R$ al subcamino de $P$ que comienza en $(a, b)$ y termina en $(a', b')$. Construyamos entonces un camino $P'$ concatenando $R$ con $Q'$. Como $P' = R +\!\!+\, Q'$, y $P = R +\!\!+\, Q$, tenemos que $\text{costo}(P') = \text{costo}(R) + \text{costo}(Q')$, y $\text{costo}(P) = \text{costo}(R) + \text{costo}(Q)$, por lo tanto $\text{costo}(P') < \text{costo}(P)$ (por $\text{costo}(Q') < \text{costo}(Q)$). Absurdo! Pues comenzamos diciendo que $P$ era un camino mínimo, y construimos otro camino de costo estrictamente menor. El absurdo provino de asumir que $Q$ no era mínimo. $\square$

# 👀 Observación Clave

Un camino desde la posición (i,j) está formado por un camino desde (i + 1, j) o desde (i, j + 1).

# 👀 Observación Clave

Un camino desde la posición (i,j) está formado por un camino desde $(i + 1, j)$ o desde $(i, j + 1)$.

Podemos usar esto para pensar la función de recurrencia?

# 🔄 Función de Recurrencia

$$
minCamino(fila, col, M) =
$$

$$
\begin{cases}
M_{fila,col} & \text{si } fila = m \wedge col = n \\
M_{fila,col} + minCamino(fila, col+1, M) & \text{si } fila = m \wedge col < n \\
M_{fila,col} + minCamino(fila+1, col, M) & \text{si } fila < m \wedge col = n \\
M_{fila,col} + \min\{minCamino(fila, col+1, M), & \\
\qquad minCamino(fila+1, col, M)\} & \text{sino}
\end{cases}
$$

# ✅ Correctitud

**Teorema (Correctitud de *minCamino*)**

$minCamino(fila, col, M)$ es el costo de un camino mínimo desde $(fila, col)$ hasta $(m, n)$ en la matriz $M$ para todo $fila \in \{0, \dots, m\}$ y $col \in \{0, \dots, n\}$.

# Demostración Correctitud I

Sea $P$ un camino de costo mínimo de $(fila, col)$ a $(m, n)$. Queremos ver que $minCamino(fila, col, M) = \text{costo}(P)$. Vamos a demostrarlo por inducción en la estructura de $P$. Si $P$ tiene más de un elemento, llamamos $Q$ al sufijo de $P$ de longitud $|P| - 1$.

1. Si $(fila, col) = (m, n)$: $\exists!$ camino $P$ y $P =$ [(m,n)] $\implies$ $costo$(P) $=\mathrm{M}_{fila,col} = minCamino(fila, col, M)$.
2. Si $fila = m$ y $col < n$: Por definición de camino, el segundo elemento de $P$ debe ser $(fila, col + 1)$. Además, existe $Q$, y por el Lema, $Q$ es de costo mínimo entre los caminos desde $(fila, col + 1)$ hasta $(m, n)$. Por hipótesis inductiva, $minCamino(fila, col + 1, M)$ es el costo de $Q$. Además, por definición de costo, el costo de $P$ es el costo de $Q$ más $M_{fila,col}$, que es exactamente $minCamino(fila, col, M)$.

# Demostración Correctitud II

3. Si $fila < m$ y $col = n$: Análogo al anterior.

4. Si $fila < m$ y $col < n$: El camino $P$ puede tener como segundo elemento una de dos cosas: $(fila + 1, col)$, o $(fila, col + 1)$. El subcamino $Q$ tendrá como primer elemento el segundo elemento de $P$, sea cual sea. Por el Lema, $Q$ es de costo mínimo entre los que van desde su primer elemento hasta $(m, n)$. Por hipótesis inductiva, entonces, su costo es o $minCamino(fila + 1, col, M)$, o $minCamino(fila, col + 1, M)$. Por lo tanto, por definición de costo, el costo de $P$ es igual o a $M_{fila,col} + minCamino(fila + 1, col, M)$, o a $M_{fila,col} + minCamino(fila, col + 1, M)$.

   Por otro lado, existen caminos con esos costos. Los mismos son formados agregándole el elemento $(fila, col)$ a los caminos óptimos desde $(fila + 1, col)$ y $(fila, col + 1)$, respectivamente. Esto quiere decir que el costo de $P$, al ser un camino mínimo, debe ser menor o

# Demostración Correctitud III

igual a los costos de estos dos caminos. Como el costo de $P$ debe ser uno de los dos costos anteriores, y debe ser menor o igual a los dos, forzosamente debe ser el menor de los dos. Entonces

$$
\begin{aligned}
&\text{costo}(P)\\
&\qquad = \min\{M_{fila,col} + minCamino(fila+1, col, M),\\
&\qquad\qquad\qquad M_{fila,col} + minCamino(fila, col+1, M)\}\\
&\qquad = M_{fila,col} + \min\{minCamino(fila+1, col, M),\\
&\qquad\qquad\qquad\qquad minCamino(fila, col+1, M)\}\\
&\qquad = minCamino(fila, col, M)
\end{aligned}
$$

Demostramos que sin importar qué valores tienen *fila* y *col*, $minCamino(fila, col, M)$ es el costo de un camino de costo mínimo entre $(fila, col)$ y $(m, n)$, que es lo que queríamos.

# ⬇️ Algoritmo Top-down

```python
def f(i: int, j: int, M, memo=None) -> int:
    filas = len(M)
    columnas = len(M[0])
    if memo is None:
        memo = [[-1] * columnas for _ in range(filas)]
    if memo[i][j] == -1:
        memo[i][j] = M[i][j]
        if i == filas-1 and j < columnas-1:
            memo[i][j] += f(i, j+1, M, memo)
        elif i < filas-1 and j == columnas-1:
            memo[i][j] += f(i+1, j, M, memo)
        elif i < filas-1 and j < columnas-1:
            memo[i][j] += min(f(i+1, j, M, memo), f(i, j+1, M, memo))
    return memo[i][j]
```

# ⬆️ Algoritmo Bottom-up

```python
def minCamino(M) -> dict:
    dic = {}
    m = len(M) - 1
    n = len(M[0]) - 1
    dic[(m,n)] = M[m][n] #caso 1
    for j in range(n-1, -1, -1): #caso 2
        dic[(m, j)] = M[m][j] + dic[(m, j+1)]

    for i in range(m-1, -1, -1): #caso 3
        dic[(i,n)] = M[i][n] + dic[(i+1, n)]

    for i in range(m-1, -1, -1): #caso 4
        for j in range(n-1, -1, -1):
            dic[(i,j)] = M[i][j] + min(dic[(i+1,j)], dic[(i, j+1)])
    return dic
```

# Reconstrucción I

```python
def reconstruir(M) -> list:
    costos = minCamino(M)
    m = len(M) - 1
    n = len(M[0]) - 1
    camino = [(0,0)]
    i = j = 0
    while i < m or j < n:
        if i == m: #caso 2
            j += 1
        elif j == n: #caso 3
            i += 1
        else: #caso 4
            if costos[(i+1,j)] < costos[(i, j+1)]:
                i += 1
            else:
```

# Reconstrucción II

```
                j += 1
        camino.append((i, j))

    return camino
```

# 🏗️ Reconstrucción (otra forma)

```
ABAJO = 1
DERECHA = 2

def minCamino(M, direcciones) -> dict:
    dic = {}
    m = len(M) - 1
    n = len(M[0]) - 1

    dic[(m,n)] = M[m][n]
    direcciones[(m,n)] = -1 #caso 1

    for j in range(n-1, -1, -1): #caso 2
        dic[(m, j)] = M[m][j] + dic[(m, j+1)]
        direcciones[(m,j)] = DERECHA
    ...
```

# 🏗 Reconstrucción (otra forma)

```
...
    for i in range(m-1, -1, -1): #caso 3
        dic[(i,n)] = M[i][n] + dic[(i+1, n)]
        direcciones[(i,n)] = ABAJO

    for i in range(m-1, -1, -1): #caso 4
        for j in range(n-1, -1, -1):
            dic[(i,j)] = M[i][j] + min(dic[(i+1,j)], dic[(i, j+1)])
            if dic[(i+1,j)] < dic[(i, j+1)]:
                direcciones[(i,j)] = ABAJO
            else:
                direcciones[(i,j)] = DERECHA

    return dic
```

# Reconstrucción (otra forma)

```python
def reconstruir(direcciones) -> list:
    camino = [(0, 0)]

    while direcciones[camino[-1]] != -1:
        i, j = camino[-1]
        if direcciones[(i,j)] == ABAJO:
            camino.append((i+1, j))
        else:
            camino.append((i, j+1))

    return camino
```

# 🏗 Reconstrucción (otra forma)

```
M = [[1,2,3], [4,5,6], [7,8,9], [10,11,12]]
direcciones = [[-1] * len(M[0]) for _ in range(len(M))]

minCamino(M, direcciones)
camino = reconstruir(direcciones)
print(camino)
# [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2)]
```

# 😵‍💫 Complejidades

- ⬇️ Algoritmo Top-down:
  - 🕑 Temporal:
  - 🛸 Espacial:
- ⬆️ Algoritmo Bottom-up:
  - 🕑 Temporal:
  - 🛸 Espacial:

- 🏗️ Reconstrucción 1:
  - 🕑 Temporal:
  - 🛸 Espacial:
- 🏗️ Reconstrucción 2:
  - 🕑 Temporal:
  - 🛸 Espacial:

# 😵‍💫 Complejidades

- ⬇️ Algoritmo Top-down:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial:
- ⬆️ Algoritmo Bottom-up:
  - 🕑 Temporal:
  - 🛸 Espacial:
- 🏗️ Reconstrucción 1:
  - 🕑 Temporal:
  - 🛸 Espacial:
- 🏗️ Reconstrucción 2:
  - 🕑 Temporal:
  - 🛸 Espacial:

# 😵‍💫 Complejidades

- ⬇️ Algoritmo Top-down:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial: $\mathcal{O}(mn)$
- ⬆️ Algoritmo Bottom-up:
  - 🕑 Temporal:
  - 🛸 Espacial:
- 🏗️ Reconstrucción 1:
  - 🕑 Temporal:
  - 🛸 Espacial:
- 🏗️ Reconstrucción 2:
  - 🕑 Temporal:
  - 🛸 Espacial:

# 😵‍💫 Complejidades

- ⬇️ Algoritmo Top-down:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial: $\mathcal{O}(mn)$
- ⬆️ Algoritmo Bottom-up:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial:
- 🏗️ Reconstrucción 1:
  - 🕑 Temporal:
  - 🛸 Espacial:
- 🏗️ Reconstrucción 2:
  - 🕑 Temporal:
  - 🛸 Espacial:

# Complejidades

- Algoritmo Top-down:
  - Temporal: $\mathcal{O}(mn)$
  - Espacial: $\mathcal{O}(mn)$
- Algoritmo Bottom-up:
  - Temporal: $\mathcal{O}(mn)$
  - Espacial: $\mathcal{O}(mn)$

- Reconstrucción 1:
  - Temporal:
  - Espacial:
- Reconstrucción 2:
  - Temporal:
  - Espacial:

# 😵‍💫 Complejidades

⬇️ Algoritmo Top-down:
- 🕑 Temporal: $\mathcal{O}(mn)$
- 🛸 Espacial: $\mathcal{O}(mn)$

⬆️ Algoritmo Bottom-up:
- 🕑 Temporal: $\mathcal{O}(mn)$
- 🛸 Espacial: $\mathcal{O}(mn)$

🏗️ Reconstrucción 1:
- 🕑 Temporal: $\mathcal{O}(mn)$
- 🛸 Espacial:

🏗️ Reconstrucción 2:
- 🕑 Temporal:
- 🛸 Espacial:

# 😵‍💫 Complejidades

- ⬇️ Algoritmo Top-down:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial: $\mathcal{O}(mn)$
- ⬆️ Algoritmo Bottom-up:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial: $\mathcal{O}(mn)$
- 🏗️ Reconstrucción 1:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial: $\mathcal{O}(mn)$
- 🏗️ Reconstrucción 2:
  - 🕑 Temporal:
  - 🛸 Espacial:

# Complejidades

- Algoritmo Top-down:
  - Temporal: $\mathcal{O}(mn)$
  - Espacial: $\mathcal{O}(mn)$
- Algoritmo Bottom-up:
  - Temporal: $\mathcal{O}(mn)$
  - Espacial: $\mathcal{O}(mn)$
- Reconstrucción 1:
  - Temporal: $\mathcal{O}(mn)$
  - Espacial: $\mathcal{O}(mn)$
- Reconstrucción 2:
  - Temporal: $\mathcal{O}(mn)$
  - Espacial:

# 😵‍💫 Complejidades

- ⬇️ Algoritmo Top-down:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial: $\mathcal{O}(mn)$
- ⬆️ Algoritmo Bottom-up:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial: $\mathcal{O}(mn)$

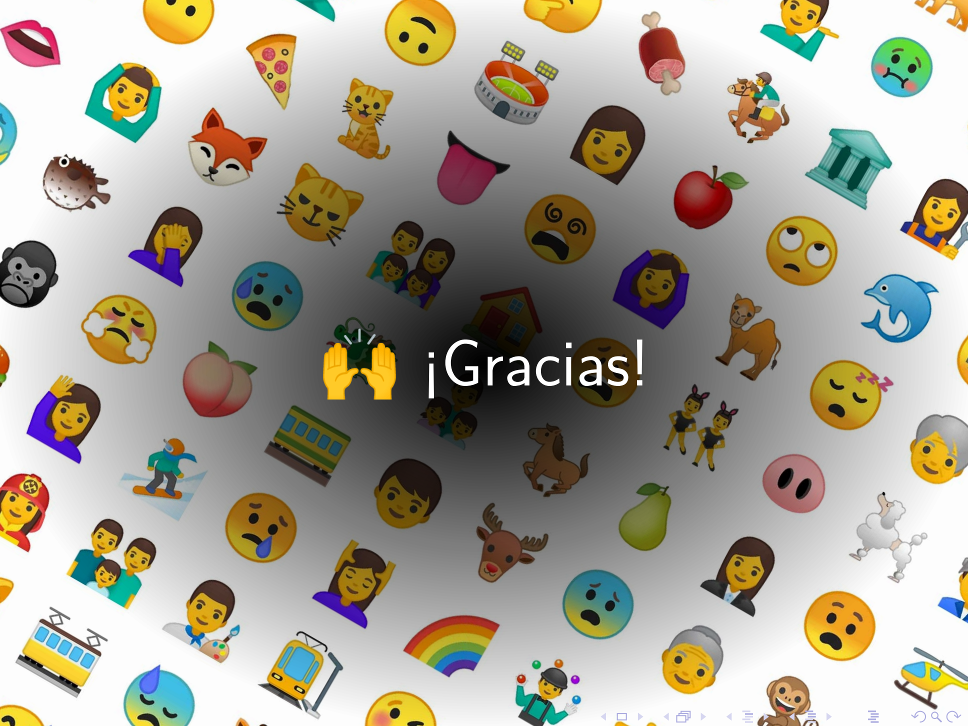
- 🏗️ Reconstrucción 1:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial: $\mathcal{O}(mn)$
- 🏗️ Reconstrucción 2:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial: $\mathcal{O}(mn)$

# 😵‍💫 Complejidades

- ⬇️ Algoritmo Top-down:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial: $\mathcal{O}(mn)$
- ⬆️ Algoritmo Bottom-up:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial: $\mathcal{O}(mn)$

- 🏗️ Reconstrucción 1:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial: $\mathcal{O}(mn)$
- 🏗️ Reconstrucción 2:
  - 🕑 Temporal: $\mathcal{O}(mn)$
  - 🛸 Espacial: $\mathcal{O}(mn)$

¿Se puede mejorar alguno?


🙌 ¡Gracias!