# Programación dinámica

## Práctica calificada II

Jarem Villalobos[1] Joaquín Aynaya[2]

Lunes 22 de septiembre de 2025

# Table of contents

# Longest palindrome Sequence: Definición

El siguiente algoritmo encuentra la longitud de la subsecuencia máxima de palíndromos que se pueden encontrar dentro de una secuencia de caracteres. Para ello usa un matriz n x n, siendo n la longitud de la secuencia dada

# Longest palindrome Sequence: Pseudocódigo

```
Procedimiento Subsecuencia(s)
    n <- longitud(s)
    crear matriz de longitud (n)x(n), inicializada en 0
    Para i desde 0 hasta n-1:
        dp[i][i] <- 1
    FinPara
    Para m desde 2 hasta n:
        Para i desde 0 hasta n-m:
            j <- i+m-1
            Si s[i] = s[j]:
                Si m = 2:
                    dp[i][j] <- 2
                Sino:
                    dp[i][j] <- 2+dp[i+1][j-1]
                FinSi
            Sino:
                dp[i][j] <- max(dp[i+1][j],dp[i][j-1])
            FinSi
        FinPara
    FinPara
    devolver dp[0][n-1]
FinProcedimiento
```

# Longest palindrome Sequence: Complejidad

- Inicializar la matriz nxn tiene como complejidad temporal $\mathcal{O}(n^2)$
- Se ejecuta un bucle for n veces, teniendo una complejidad $\mathcal{O}(n)$
- El bucle anidado:
  - El bucle externo se ejecuta desde 2 hasta n, dando n-1 iteraciones -¿ O(n).
  - El bucle interno se recorre desde 0 hasta n-tamaño -¿ O(n)
  - Cada operación dentro de los bucle tiene complejidad temporal O(1)
- El bucle anidado tiene complejidad $\mathcal{O}(n^2)$

$$T(n) = \mathcal{O}(n^2) + \mathcal{O}(n) + \mathcal{O}(n^2) = \mathcal{O}(n^2)$$

# Ordenación de multiplicación de matrices: Definición

El siguiente algoritmo encuentra el orden optimo de multiplicación de una secuencia de multiplicaciones matriciales. Para ello recibe un arreglo de las dimensiones de cada matriz, siendo los elementos i-1 e i el numero de filas y columnas respectivamente de la matriz i.

# Ordenación de multiplicación de matrices: Pseudocódigo

```
Procedimiento MenorNumeroDeProductos(p):
    n <- longitud(p - 1)
    Crear matriz dp de longitud (n+1) x (n+1), inicializada en 0

    Para m desde 2 hasta n:
        Para i desde 1 hasta n - m + 1:
            j <- i + m - 1
            dp[i][j] <- infinito
            Para k desde i hasta j - 1:
                costo <- dp[i][k] + dp[k+1][j] + p[i-1] * p[k] * p[j]
                Si costo < dp[i][j]:
                    dp[i][j] <- costo
                FinSi
            FinPara
        FinPara
    FinPara
    Devolver dp[1][n]
FinProcedimiento
```

# Ordenación de multiplicación de matrices: Complejidad

- Inicializar la matriz nxn tiene como complejidad temporal $\mathcal{O}(n^2)$.
- Se ejecuta el bucle externo $n-1$ veces.
- El bucle sobre el índice $i$ se ejecuta $n-m+1$ veces. En el peor caso es $\mathcal{O}(n)$.
- El bucle sobre el punto de partición $k$ se ejecuta $m-1$ veces. En el peor caso es $\mathcal{O}(n)$.
- Cada operación dentro de los bucle tiene complejidad temporal $\mathcal{O}(1)$.

$\Longrightarrow$ Los bucles anidados nos dan una complejidad total $\mathcal{O}(n^3)$.

$$T(n) = \mathcal{O}(n^2) + \mathcal{O}(n^3) = \mathcal{O}(n^3)$$

# Optimal Binary Search Tree: Definición

Dado un arreglo de elementos **ordenados** $K[1...n]$ y otro arreglo $V[1...n]$ que indica la frecuencia de cada elemento (también definida como la probabilidad de aparición de cada elemento), Se desea buscar un árbol de búsqueda binaria tal que la métrica:

$$M_K = \sum_{i=1}^{n} (V[i] \times \ell(K[i]))$$

Donde $\ell(v)$ representa el nivel de un nodo en el árbol binario, sea mínima

# Optimal Binary Search Tree: Solución

Se presenta recursión en el problema: Dados $i, j, l : 1 \leq i \leq l \leq j \leq n$, podemos construir árboles del subarreglo $K[i...j]$ con raíz $K[l]$ y subárboles derechos e izquierdos con nodos dentro de los subarreglos $K[i...l-1]$ y $K[l+1...i]$. Se tendría que su métrica puede ser expresada como:

$$M_{i,j} = M_{i,l-1} + M_{l+1,j} + \sum_{k=i}^{j} V[k]$$

Dado que se busca el mínimo entre todas estas posibilidades, este sería:

$$M_{i,j}^{\text{min}} = \min_{i \leq l \leq j} \{M_{i,l-1} + M_{l+1,j}\} + \sum_{k=i}^{j} V[k]$$

Junto a los casos triviales

$$M_{i,i}^{\text{min}} = V[i] \quad , \quad M_{i,j}^{\text{min}} = 0 \quad , \quad \forall j < 1$$

Se obtiene la solución para $M^{min} = M_{1,n}$

# Optimal Binary Search Tree: Pseudocódigo

```
Procedimiento OptimalBST(K,V)
    dp = construir_dp(V)

    Para d Desde 1 Hasta n-1 Hacer
        Para i Desde 1 Hasta n-d Hacer
            j = d + i
            min <- -inf
            Para l Desde i hasta j Hacer
                m = dp[i-1][l-1] + dp[l][j]
                Si m < min Entonces
                    min = m
                FinSi
            FinPara
            Para k Desde i Hasta j Hacer
                min <- min + V[k-1]
            FinPara
            dp[i-1][j] = min
        FinPara
    FinPara
    Retornar dp[0][n]
FinProcedimiento
```

```
Procedimiento construir_dp(V)
    n <- V.length
    dp <- Matriz[n+1][n+1]
        todas entradas 0
    Para i Desde 0 Hasta n-1
        dp[i][i+1] = V[i]
    FinPara
    Retornar dp
FinProcedimiento
```

# Optimal Binary Search Tree: Complejidad

Partimos de la premisa que, dado un arreglo de $n$ entradas, hay $n - m + 1$ posibles subarreglos de longitud $m$. Por cada subarreglo, se realizan $\mathcal{O}(m)$ operaciones buscando la métrica mínima. Entonces:

$$
\begin{aligned}
T(n) &= \sum_{m=1}^{n} (n - m + 1) \times cm \\
&= c(n+1)\sum_{k=1}^{n} k - c\sum_{k=1}^{n} k^2 \\
&= c\frac{n(n+1)(n+2)}{6} \\
&= \mathcal{O}(n^3)
\end{aligned}
$$

Por lo tanto su complejidad es $\mathcal{O}(n^3)$

# Alternating Coin Game: Definición del problema

Dado un arreglo $C[1...n]$ que representa los valores de una fila de $n$ monedas, se propone un juego entre dos jugadores con las siguientes reglas:

- Cada jugador toma turnos intercalados
- Cada jugador puede sacar una moneda sólo de un extremo de la fila de monedas por turno
- Si se saca una moneda, no puede ser regresada a la fila

El problema consiste en determinar la suma máxima que puede acumular el primer jugador ($S^{\text{max}}$), tomando en cuenta que el adversario *es igual de inteligente que el jugador*

# Alternating Coin Game: Solución

Como sabemos que el oponente es igual de inteligente que nosotros, asumimos que el oponente **siempre tratará de tomar la moneda que nos conduza a tener la menor suma posible**.
Supongamos que han transcurrido algunos turnos y tenemos el subarreglo $C[i, ..., j]$ del arreglo original de monedas. Tenemos dos casos:

- Tomamos $C[i]$: Entonces el oponente tiene dos opciones, tomar C[i+1] o C[j]. Dado que el oponente buscará tomar la alternativa que nos conduzca a tomar el mínimo valor posible, entonces el máximo valor que podemos recolectar es:

$$S_{i,j} = C[i] + \min\{S_{i+2,j}, S_{i+1,j-1}\}$$

- Tomamos $C[j]$: Análogamente al caso anterior, el oponente hará su jugada y tendremos que el valor máximo recolectable es

$$S_{i,j} = C[j] + \min\{S_{i,j-2}, S_{i+1,j-1}\}$$

# Alternating Coin Game: Solución

El resultado que maximize nuestra ganancia es el máximo entre estos dos casos. Así:

$$S_{i,j}^{\max} = \max\{C[i] + \min\{S_{i+2,j}^{\max}, S_{i+1,j-1}^{\max}\}, C[j] + \min\{S_{i,j-2}^{\max}, S_{i+1,j-1}^{\max}\}\}$$

Junto a los casos triviales

$$S_{i,i}^{\max} = C[i]$$
$$S_{i,j}^{\max} = 0 \quad , \quad \forall j < i$$

Tendríamos que la respuesta al problema se obtendría computando:

$$S^{\max} = S_{1,n}^{\max}$$

# Alternating Coin Game: Pseudocódigo

```
Procedimiento maxGameResult(C)
    n <- C.length
    dp <- Matriz[n+1][n+1]
    construir(dp)
    Para d Desde 2 Hasta n Hacer
        j = d+i
        Para i Desde 0 Hasta n-d Hacer
            s_1 = C[i] +
            min(dp[i+2][j],dp[i+1][j-1])
            s_2 = C[j-1] +
            min(dp[i][j-2],dp[i+1][j-1])
            dp[i][j] = max(s_1,s_2)
        FinPara
        Retornar dp[0][n]
    FinPara
FinProcedimiento
```

```
Procedimiento construir_dp(dp)
    n <- K.length
    Para i Desde 0 Hasta n
        Para j Desde 0 Hasta i
            dp[i][j] = 0
        FinPara
        Si i < n Entonces
            dp[i][i+1] = V[i]
        FinSi
    FinPara
    Retornar dp
FinProcedimiento
```

# Alternating Coin Game: Complejidad

La cantidad de subarreglos obtenibles de $C[1...n]$ de longitud $m$ es $n - m + 1$ Por cada subarreglo, se ejecutan $\mathcal{O}(1)$ operaciones para obtener la solución. Así:

$$
\begin{aligned}
T(n) &= \sum_{m=1}^{n} (n - m + 1) \times c \\
&= c\left(n(n+1) - \frac{n(n+1)}{2}\right) \\
&= c\frac{n(n+1)}{2} \\
&= \mathcal{O}(n^2)
\end{aligned}
$$

Finalmente, el algoritmo tiene complejidad $\mathcal{O}(n^2)$