# AED 3: Programación Dinámica

Agustín Santiago Gutiérrez

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Abril 2018

# AED 3, Práctica 3: Técnicas algorítmicas

En la primera parte de la materia, se explican varias **técnicas algorítmicas generales**:

- Repaso (AED2)
  - Recursión
  - Divide and Conquer
- Nuevas
  - Algoritmos golosos
  - **Programación dinámica** [Ejercicio elegido 3.8]
  - Backtracking

Objetivos de la práctica:

- Ejercitar el uso de las técnicas en la formulación de algoritmos.
- Ejercitar el análisis de los mismos
  - Correctitud
  - Complejidad

# AED 3, Práctica 3: Técnicas algorítmicas

En la primera parte de la materia, se explican varias **técnicas algorítmicas generales**:

- Repaso (AED2)
  - Recursión
  - Divide and Conquer
- Nuevas
  - Algoritmos golosos
  - **Programación dinámica** [Ejercicio elegido 3.8]
  - Backtracking

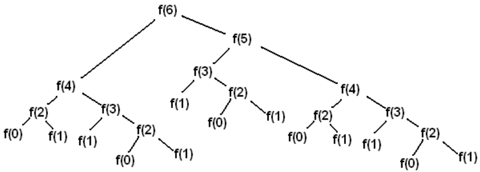
Objetivos de la práctica:

- Ejercitar el uso de las técnicas en la formulación de algoritmos.
- Ejercitar el análisis de los mismos
  - Correctitud
  - Complejidad

# ¿En qué consiste la programación dinámica?

- Es una técnica de solución de problemas **recursiva**.
- Al igual que DyC, se descompone en **subproblemas más pequeños de la misma especie**, para resolverlos recursivamente y combinar esas soluciones.
- La **diferencia** importante:
  - En DyC, los subproblemas son independientes entre sí, y se resuelven individualmente.
  - PD es aplicable cuando los subproblemas **no son independientes**.
- De realizar una implementación recursiva directa, se **recalcularían** los mismos subproblemas muchas veces.

Ejemplo Fibonacci:
$f(n) = f(n-1) + f(n-2)$



# ¿En qué consiste la programación dinámica? (2)

- Solución: **almacenar** las soluciones a subproblemas.
- De esta forma cada subproblema se calcula **una sola vez.**
- Se podría decir muy simplificadamente que: DyC $+$ Caching = PD
- Es común usar PD para problemas de **optimización**:
  - Encontrar una solución que **minimice** cierta función objetivo, en un espacio de soluciones posibles.
  - Como la técnica es recursiva, para poderse aplicar el problema debe cumplir el **principio del óptimo**.
- Principio del óptimo:
  - **Las partes de una solución óptima** a un problema, deben ser **soluciones óptimas de los correspondientes subproblemas**.
  - Es lo que permite obtener una solución óptima al problema original a partir de soluciones óptimas de los subproblemas.

# El esquema general [Según CLRS]

Los algoritmos de programación dinámica se pueden organizar típicamente en 4 pasos que responden al siguiente esquema general:

1. Caracterizar la estructura de una solución óptima.
2. Definir recursivamente el valor de una solución óptima.
3. Computar el **valor** de una solución óptima. Se calcula de manera *bottom-up*. [Según CLRS: Puede ser top-down en AED 3]
4. Construir una solución óptima a partir de la información obtenida en el paso 3

El paso 4 es optativo: se omite si solo interesa el valor o puntaje.

# Problema del recorrido óptimo en una matriz

Ejercicio 3.8, práctica 3 de Algoritmos y Estructuras de Datos 3:

Sea $M \in \mathbb{N}^{m \times n}$ una matriz de números naturales. Se desea obtener un camino que empiece en la casilla superior izquierda $(1, 1)$, termine en la casilla inferior derecha $(m, n)$ y tal que minimice la suma de los valores de las casillas por las que pasa. En cada casilla $(i, j)$ hay dos movimientos posibles: ir hacia abajo (a la casilla $(i + 1, j)$), o ir hacia la derecha (a la casilla $(i, j + 1)$).

a Diseñar un algoritmo eficiente basado en programación dinámica que resuelva este problema.

b Determinar la complejidad del algoritmo propuesto (temporal y espacial).

c Exhibir el comportamiento del algoritmo sobre la matriz que aparece a continuación.

$$\begin{bmatrix} 2 & 8 & 3 & 4 \\ 5 & 3 & 4 & 5 \\ 1 & 2 & 2 & 1 \\ 3 & 4 & 6 & 5 \end{bmatrix}$$

# Relevancia del ejercicio elegido

- Patrón de recursión sencillo (solo 2 llamadas), pero no trivial.
- Los subproblemas se pueden "ver" en la misma matriz.
- Camino mínimo en un DAG (reveer el ejercicio más adelante).
- Pide complejidad y reconstrucción del camino: es un ejercio completo, que ejercita todos los pasos generales de PD.
- Por todo esto, es un buen ejercicio para usar como primer problema de optimización en el que los subproblemas se definan con 2 variables ($i$,$j$), luego de haber visto ejemplos de PD unidimensionales.

# Fórmula recursiva

La matriz de resultados parciales almacena en $best(i,j)$ la mínima longitud de un camino que empiece en $(i,j)$ y llegue a $(m,n)$, haciendo solo movimientos hacia abajo y hacia la derecha.

- $best(i,j) = M_{i,j} + min(best(i+1,j), best(i,j+1))$
  para $1 \leq i < m$ y $1 \leq j < n$
- $best(i,n) = M_{i,n} + best(i+1,n)$ para $1 \leq i < m$
- $best(m,j) = M_{m,j} + best(m,j+1)$ para $1 \leq j < n$
- $best(m,n) = M_{m,n}$

La longitud del mínimo camino entre esquinas, que constituye la solución al problema, viene dada por $best(1,1)$. Con esto ya podríamos implementar una solución top-down recursiva (Memoization):

- Si para calcular un $best(i,j)$ necesitamos un resultado ya calculado, lo usamos directamente.
- Sino, lo calculamos recursivamente, almacenamos su valor en la tabla de resultados y luego lo utilizamos.

# Algoritmo bottom-up

```
longitudCaminoMinimo(Matriz, m, n):
    best <- matrizDeEnteros(m,n)
    best[m,n] = Matriz[m,n]
    FOR i <- m-1 DOWNTO 1 DO
        best[i,n] = Matriz[i,n] + best[i+1,n]
    FOR j <- n-1 DOWNTO 1 DO
        best[m,j] = Matriz[m,j] + best[m,j+1]
    FOR i <- m-1 DOWNTO 1 DO
        FOR j <- n-1 DOWNTO 1 DO
            best[i,j] = Matriz[i,j] + min(best[i+1,j] + best[i,j+1])
    RETURN best[1,1]
```

La complejidad del algoritmo resultante es $O(nm)$, tanto espacial como temporal. Se puede bajar la complejidad espacial a $O(min(n,m))$ si no interesa reconstruir el camino sino solo su longitud.

# Cálculo en el ejemplo

Matriz de entrada:

$$\begin{bmatrix} \mathbf{2} & 8 & 3 & 4 \\ \mathbf{5} & 3 & 4 & 5 \\ \mathbf{1} & \mathbf{2} & \mathbf{2} & \mathbf{1} \\ 3 & 4 & 6 & \mathbf{5} \end{bmatrix}$$

Matriz de best:

$$\begin{bmatrix} \mathbf{18} & 21 & 15 & 15 \\ \mathbf{16} & 13 & 12 & 11 \\ \mathbf{11} & \mathbf{10} & \mathbf{8} & \mathbf{6} \\ 18 & 15 & 11 & \mathbf{5} \end{bmatrix}$$

En ambas matrices, se indica un camino óptimo en negrita.

# Referencias


- Introduction to Algorithms, 2nd Edition. MIT Press. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein Página 323: 15 Dynamic Programming