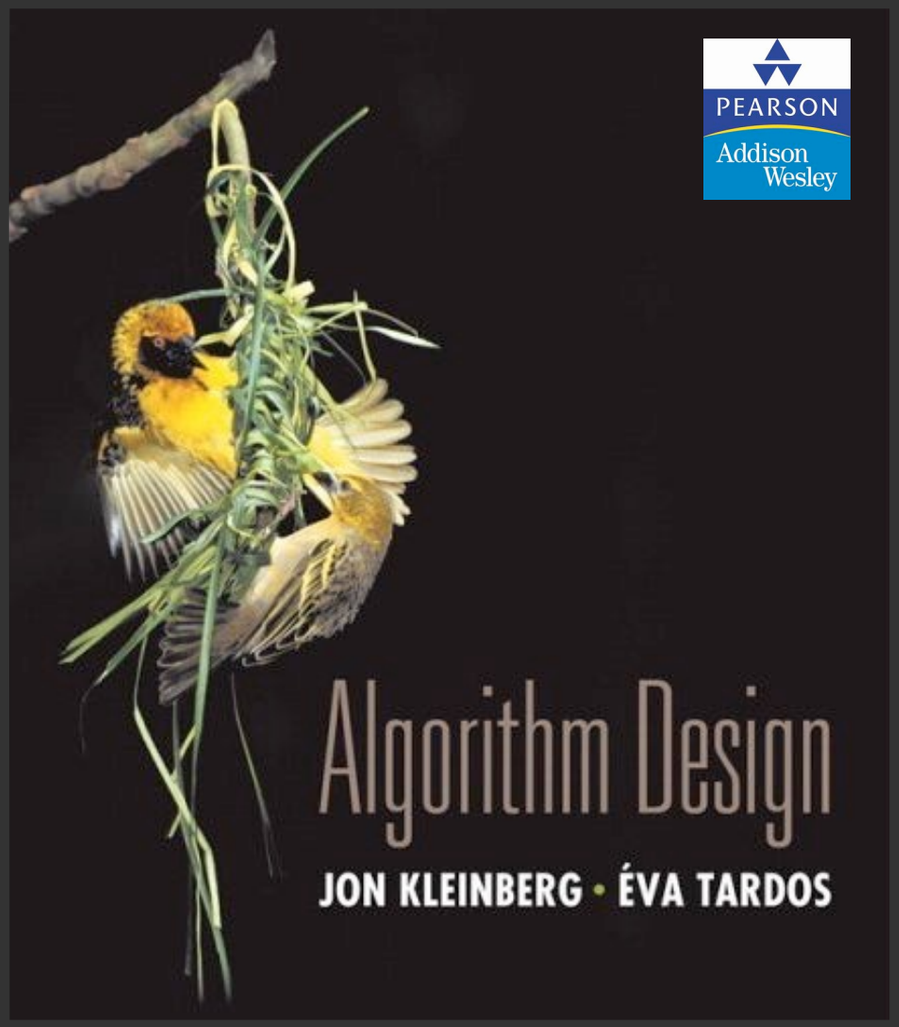


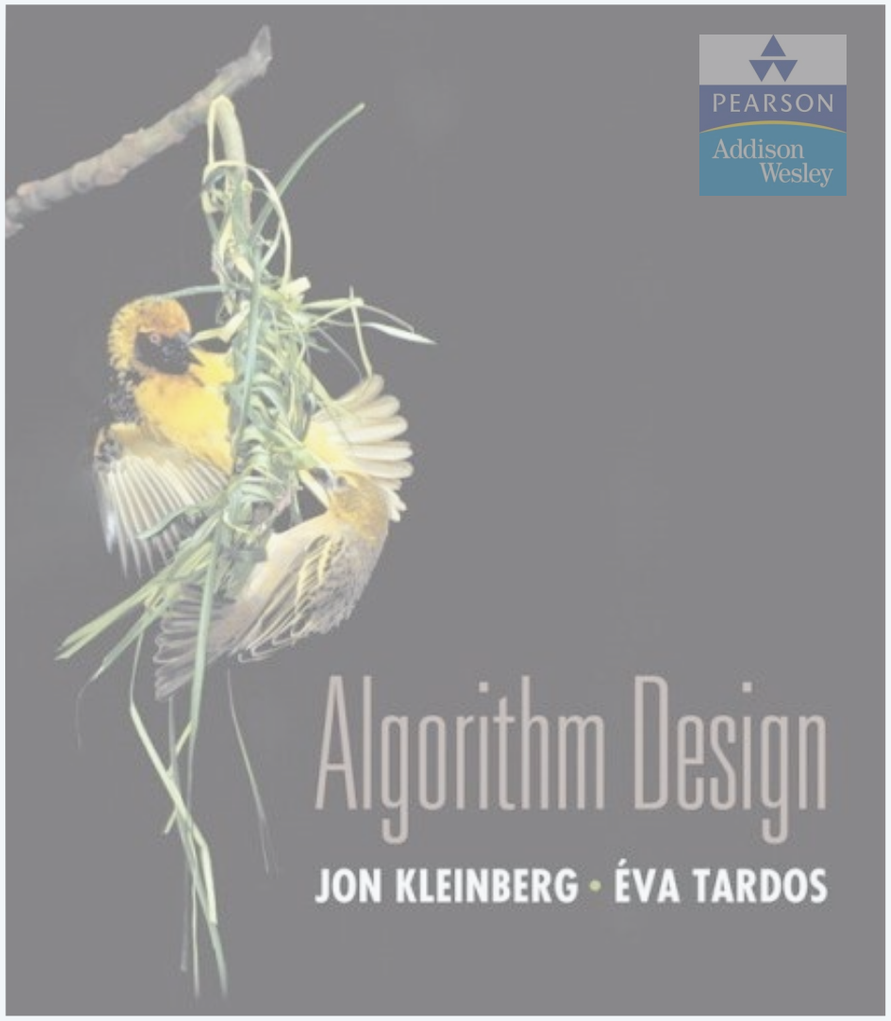
6. Programación Dinámica II

- *Estructura secundaria de ARN*
- *Alineamiento de secuencias*
- *Algoritmo de Bellman–Ford*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on 10/10/23 10:42



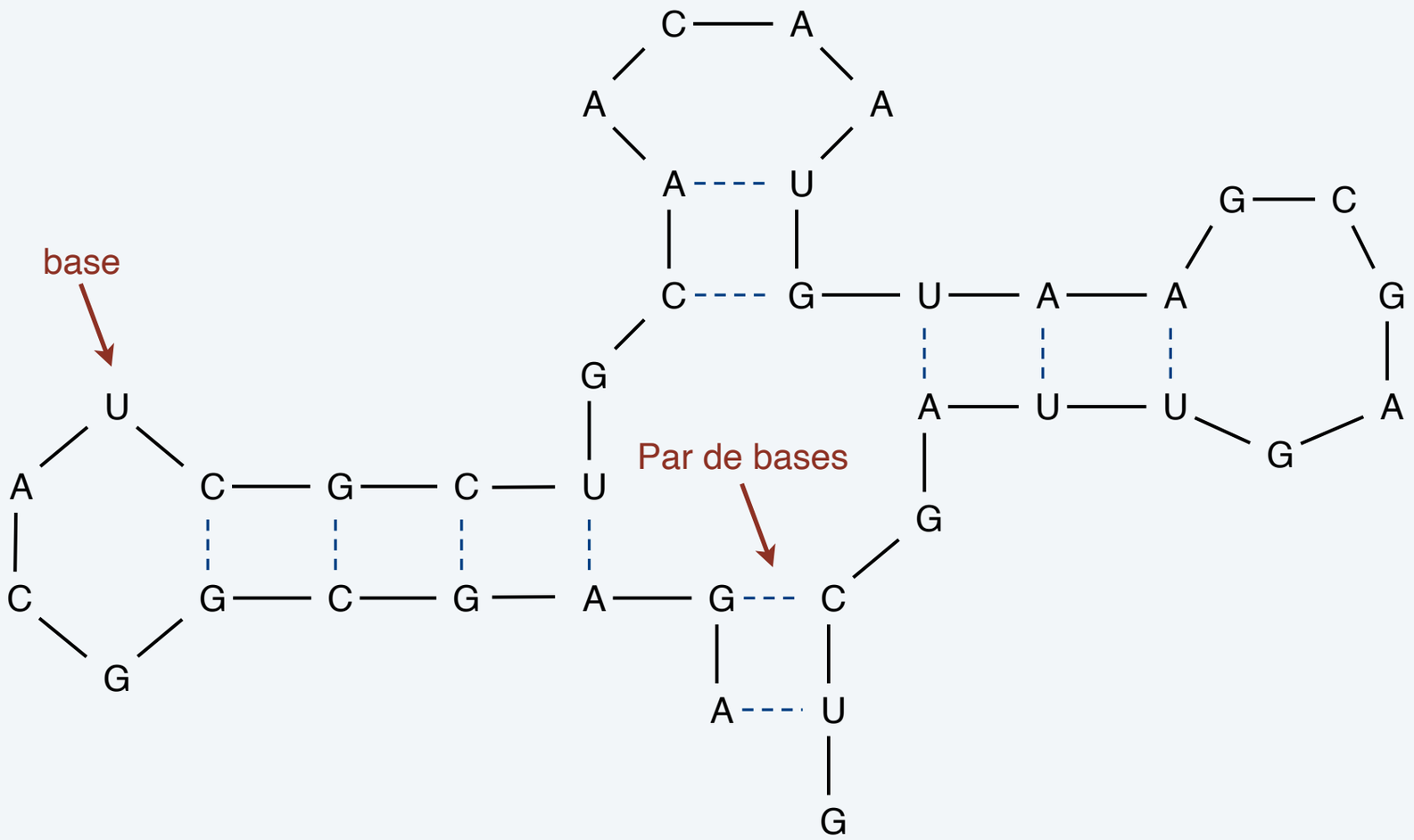
6. Programación Dinámica II

- *Estructura secundaria de ARN*
- *Alineamiento de secuencias*
- *Algoritmo de Bellman-Ford*

Estructura secundaria de ARN

ARN. Cadena $B = b_1 b_2 \ldots b_n$ sobre el alfabeto $\{ \mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{U} \}$.

Estructura secundaria. El ARN es un molécula de cadena simple. Tiende a enrollarse y formar pares de bases con sí misma.



Una posible estructura secundaria para GUCGAUUGAGCGAAUGUAACAACGUGGCUACGGCGAGA

Estructura secundaria de ARN

Estructura secundaria. Conjunto de pares $S = \{ (i, j) \}$ que satisface:

- [Watson–Crick] S es un emparejamiento tal que todos los pares de bases que define, (b_i, b_j), son complementos de Watson–Crick: A–U, U–A, C–G, or G–C.
- [Sin ángulos filosos] Las bases apareadas están separadas por al menos 4 bases intermedias. Si $(i, j) \in S$, entonces $i < j - 4$.
- [Sin cruces] Si (i, j) y (k, ℓ) pertenecen a S, entonces no podemos tener $i < k < j < \ell$.

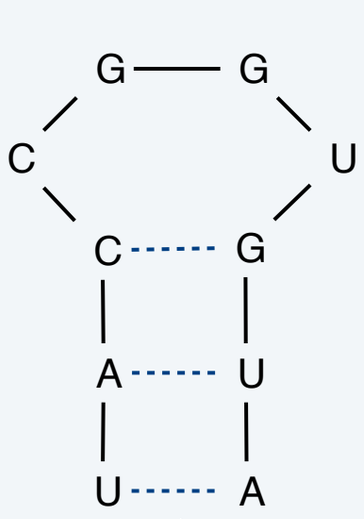
Energía libre. La hipótesis usual es que las estructuras secundarias que se forman minimizan la energía libre total.

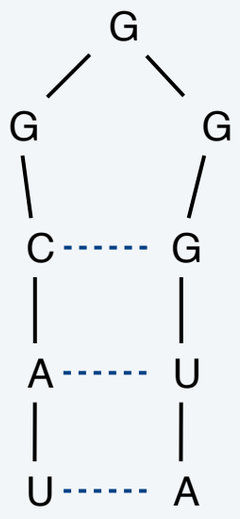
se aproxima por la cantidad de pares de bases

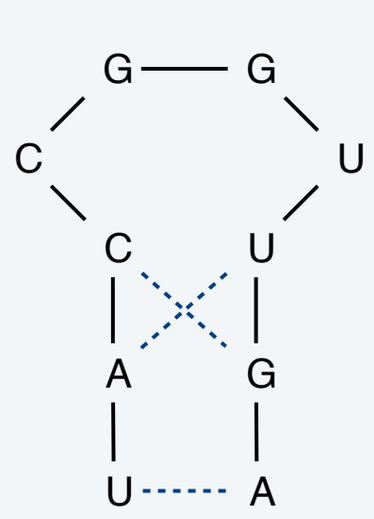
Objetivo. Dada una molécula de ARN $B = b_1 b_2 \ldots b_n$, encontrar una estructura secundaria S que maximiza la cantidad de pares de bases.

Estructura secundaria de ARN

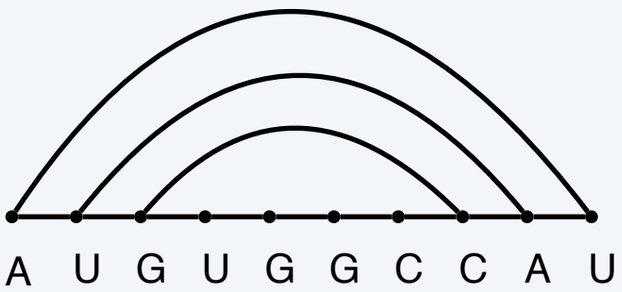
Ejemplos.



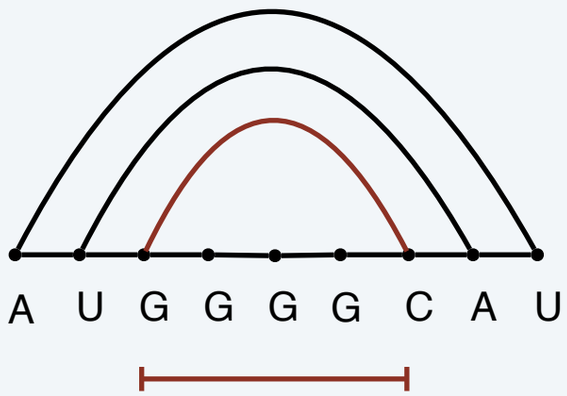




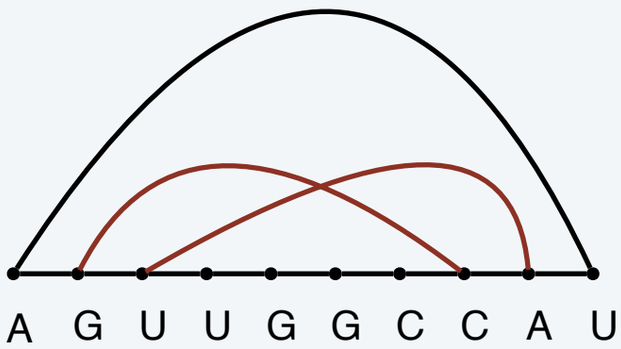
par de bases



OK



ángulo filoso
(≤4 bases intermedias)



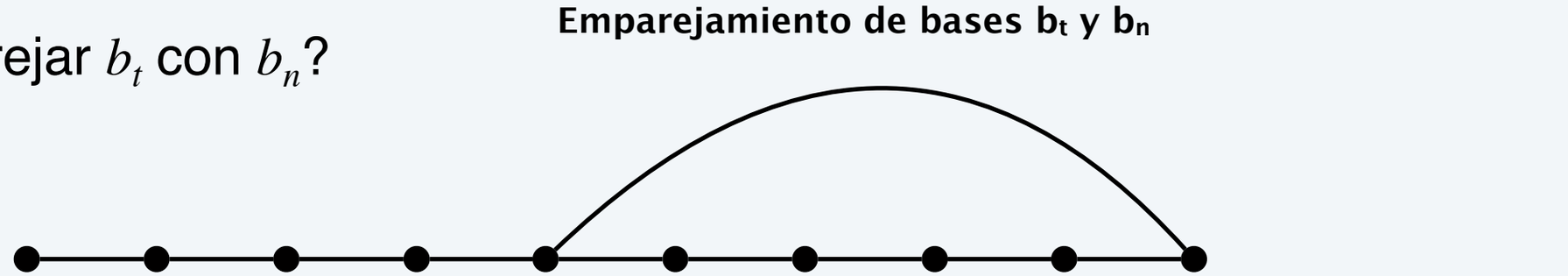
cruce

Estructura secundaria de ARN: subproblemas

Primer intento. $OPT(j)$ = máxima cantidad de pares de bases en una estructura secundaria para la subcadena $b_1 b_2 \ldots b_j$.

Objetivo. $OPT(n)$.

Elección. ¿Emparejar b_t con b_n?



No funciona. Aparecen dos subproblemas, pero uno no es de la forma necesaria.

- Encontrar estructura secundaria en $b_1 b_2 \ldots b_{t-1}$. ← OPT(t–1)
- Encontrar estructura secundaria en $b_{t+1} b_{t+2} \ldots b_{n-1}$. ← la primera base ya no es b_1)

Programación dinámica sobre intervalos

Definición. $OPT(i,\ j)$ = máxima cantidad de pares de bases en una estructura secundaria para la subcadena $b_i\, b_{i+1} \ldots b_j$.

Caso 1. Si $i \geq j - 4$.

- $OPT(i,\ j) = 0$ por la condición de que no haya ángulos filosos.

Caso 2. Base b_j no está apareada.

- $OPT(i,\ j) = \mathrm{OPT}(i,\ j-1)$.

Caso 3. Base b_j apareada con b_t para algún $t, i \leq t < j - 4$.

- Los subproblemas se optimizan independientemente por la condición de que no haya cruces.
- $\mathrm{OPT}(i, j) = 1 + \max_t \{\, OPT(i,\ t-1) + OPT(t+1,\ j-1) \,\}$.

máximo sobre los t tales que $i \leq t < j - 4$ y
b_t es el complemento de Watson–Crick de b_j

Programación dinámica sobre intervalos. Implementación bottom-up.

¿En qué orden resolver los subproblemas?

Primero los intervalos más cortos.

RNA-SECONDARY-STRUCTURE $(n, b_1, \ldots, b_n)$

FOR $k = 0$ TO 4

  FOR $i = 1$ TO $n - k$

    $j \leftarrow i + k.$

    $M[i, j] \leftarrow 0.$

FOR $k = 5$ TO $n - 1$

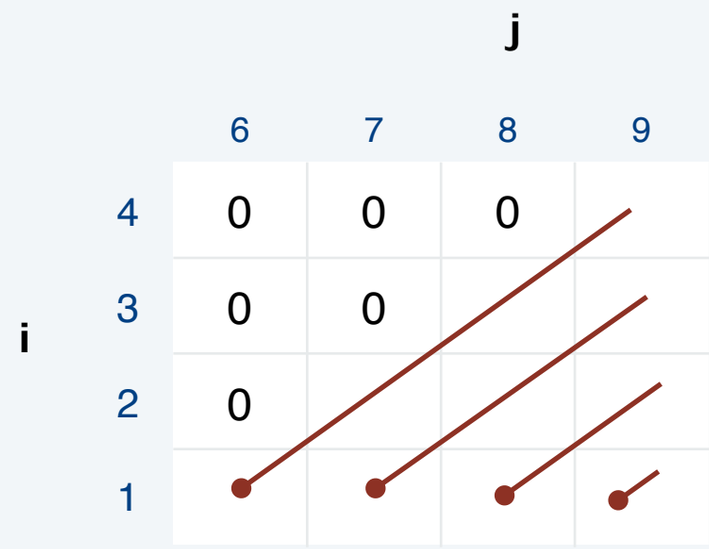
  FOR $i = 1$ TO $n - k$

    $j \leftarrow i + k.$

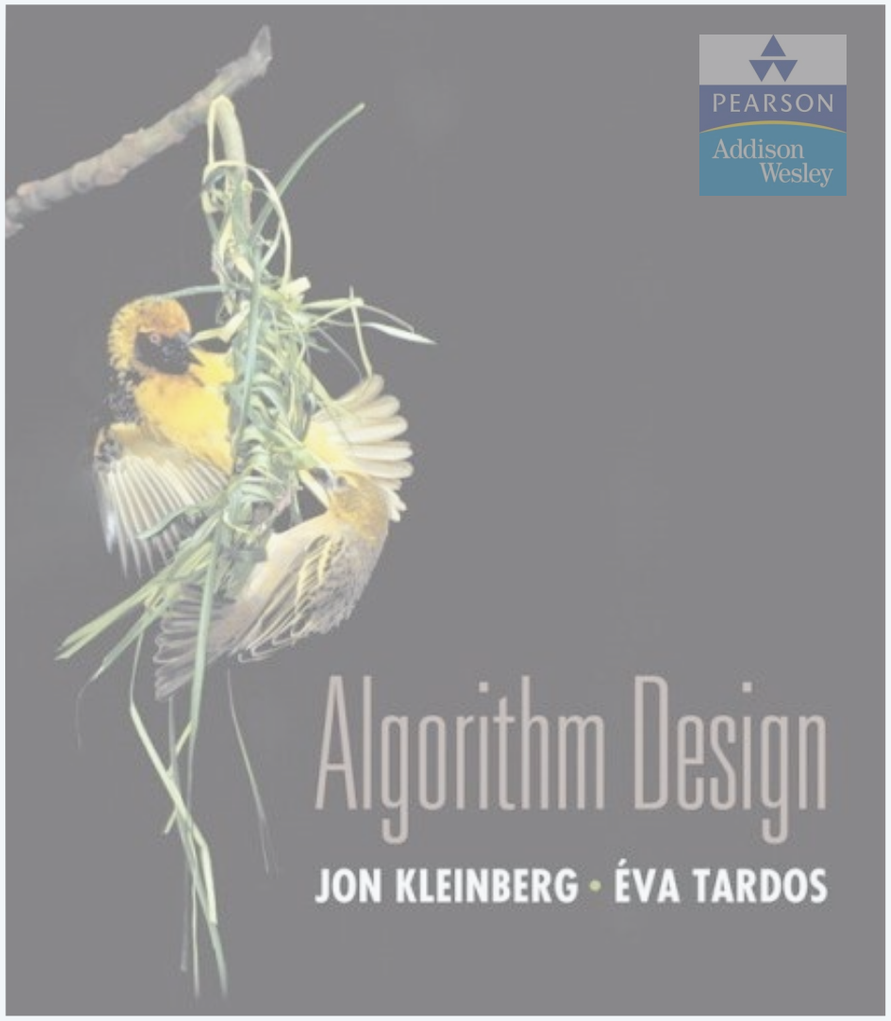
    Calcular $M[i, j]$ usando la recurrencia.

RETURN $M[1, n]$.

todos los valores necesarios fueron calculados previamente



orden en el cual se resuelven los subproblemas



6. Programación Dinámica II

- *Estructura secundaria de ARN*
- *Alineamiento de secuencias*
- *Algoritmo de Bellman-Ford*

Similitud de cadenas

¿Cómo medir similitud entre cadenas de texto?

Ejemplo. ocurrance and occurrence.

o	c	u	r	r	a	n	c	e	–
o	c	c	u	r	r	e	n	c	e

6 discrepancias, 1 hueco

o	c	–	u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

1 discrepancia, 1 hueco

o	c	–	u	r	r	–	a	n	c	e
o	c	c	u	r	r	e	–	n	c	e

0 discrepancias, 3 huecos

Distancia de edición

Distancia de edición. [Levenshtein 1966, Needleman–Wunsch 1970]

- Penalización por hueco δ; penalización por discrepancia α_{pq}.
- Costo = suma de las penalizaciones por todos los huecos y discrepancias.

costo = $\delta + \alpha_{CG} + \alpha_{TA}$

Aplicaciones. Unix diff, bioinformática, corrección de errores, ...

Alineamiento de secuencias

Objetivo. Dadas dos cadenas $x_1\, x_2 \ldots x_m$ y $y_1\, y_2 \ldots y_n$ encontrar un alineamiento de costo mínimo.

Def. Un alineamiento M es un emparejamiento entre $\{1, 2 \ldots m\}$ y $\{1, 2 \ldots n\}$ sin cruzamientos (escribimos $x_i — y_j$ o (x_i, y_j) para denotar un par (i,j) de M).

$x_i - y_j$ y $x_{i'} - y_{j'}$ se cruzan si $i < i'$ y $j > j'$

Def. El costo de un alineamiento M es:

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i : x_i \text{ unmatched}} \delta + \sum_{j : y_j \text{ unmatched}} \delta}_{\text{gap}}$$

x_1	x_2	x_3	x_4	x_5		x_6
C	T	A	C	C	–	G
–	T	A	C	A	T	G
y_1	y_2	y_3	y_4	y_5		y_6

alineamiento de CTACCG con TACATG:

$$M = \{ x_2–y_1, x_3–y_2, x_4–y_3, x_5–y_4, x_6–y_6 \}$$

Alineamiento de secuencias: descomposición en subproblemas

Def. $OPT(i, j)$ = mínimo costo de alineamiento entre $x_1\ x_2\ ...\ x_i$ y $y_1\ y_2\ ...\ y_j$.

Objetivo. $OPT(m, n)$.

Caso 1. $OPT(i, j)$ se alcanza emparejando $x_i - y_j$.

Costo de discrepancia de $x_i - y_j$ + costo mínimo de alinear $x_1\ x_2\ ...\ x_{i-1}$ con $y_1\ y_2\ ...\ y_{j-1}$.

Caso 2a. $OPT(i, j)$ se alcanza dejando x_i sin emparejar.

Costo del hueco en x_i + costo mínimo de alinear $x_1\ x_2\ ...\ x_{i-1}$ con $y_1\ y_2\ ...\ y_j$.

Caso 2b. $OPT(i, j)$ se alcanza dejando y_j sin emparejar.

Costo del hueco en y_j + costo mínimo de alinear $x_1\ x_2\ ...\ x_i$ con $y_1\ y_2\ ...\ y_{j-1}$.

$$OPT(i, j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}$$

Alineamiento de secuencias: implementación bottom-up

SEQUENCE-ALIGNMENT $(m, n, x_1, \ldots, x_m, y_1, \ldots, y_n, \delta, \alpha)$

FOR $i = 0$ TO m

$M[i, 0] \leftarrow i\,\delta$.

FOR $j = 0$ TO n

$M[0, j] \leftarrow j\,\delta$.

FOR i = 1 TO m

FOR j = 1 TO n

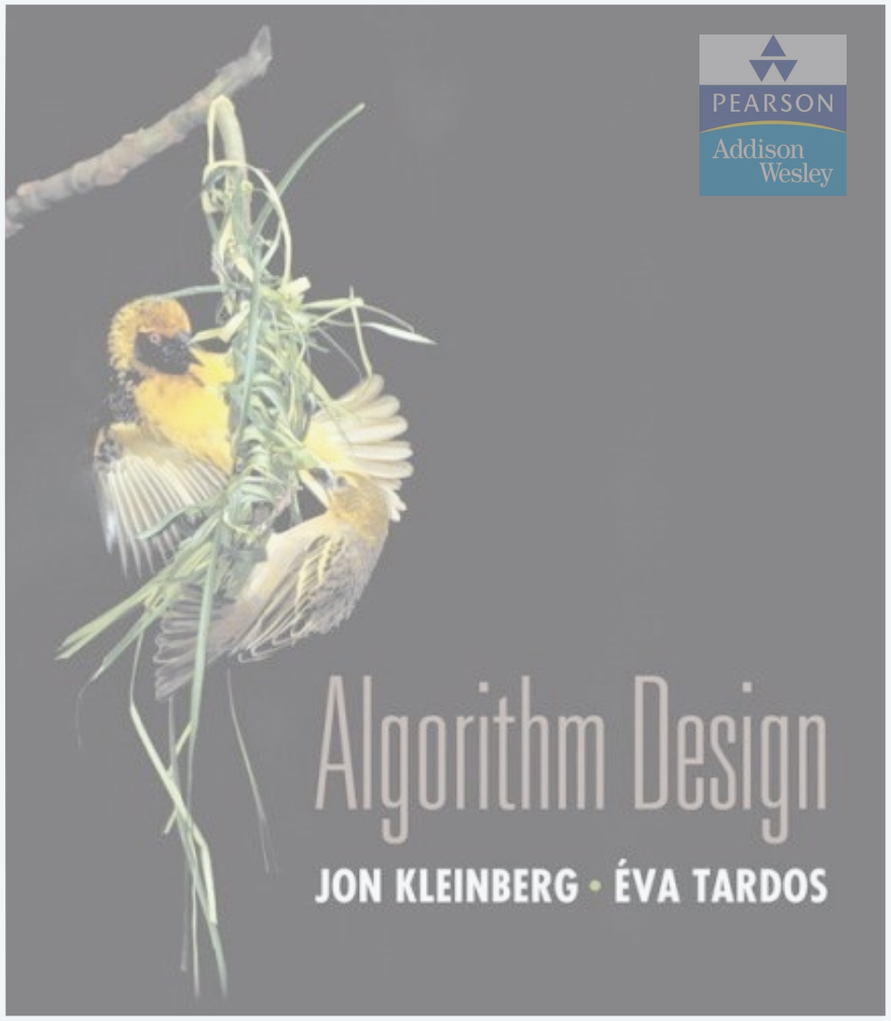
$M[i, j] \leftarrow \min \{\ \alpha[x_i, y_j] + M[i-1, j-1],$

$\delta + M[i-1, j],$

$\delta + M[i, j-1]).$

Calculado previamente

RETURN $M[m, n]$.

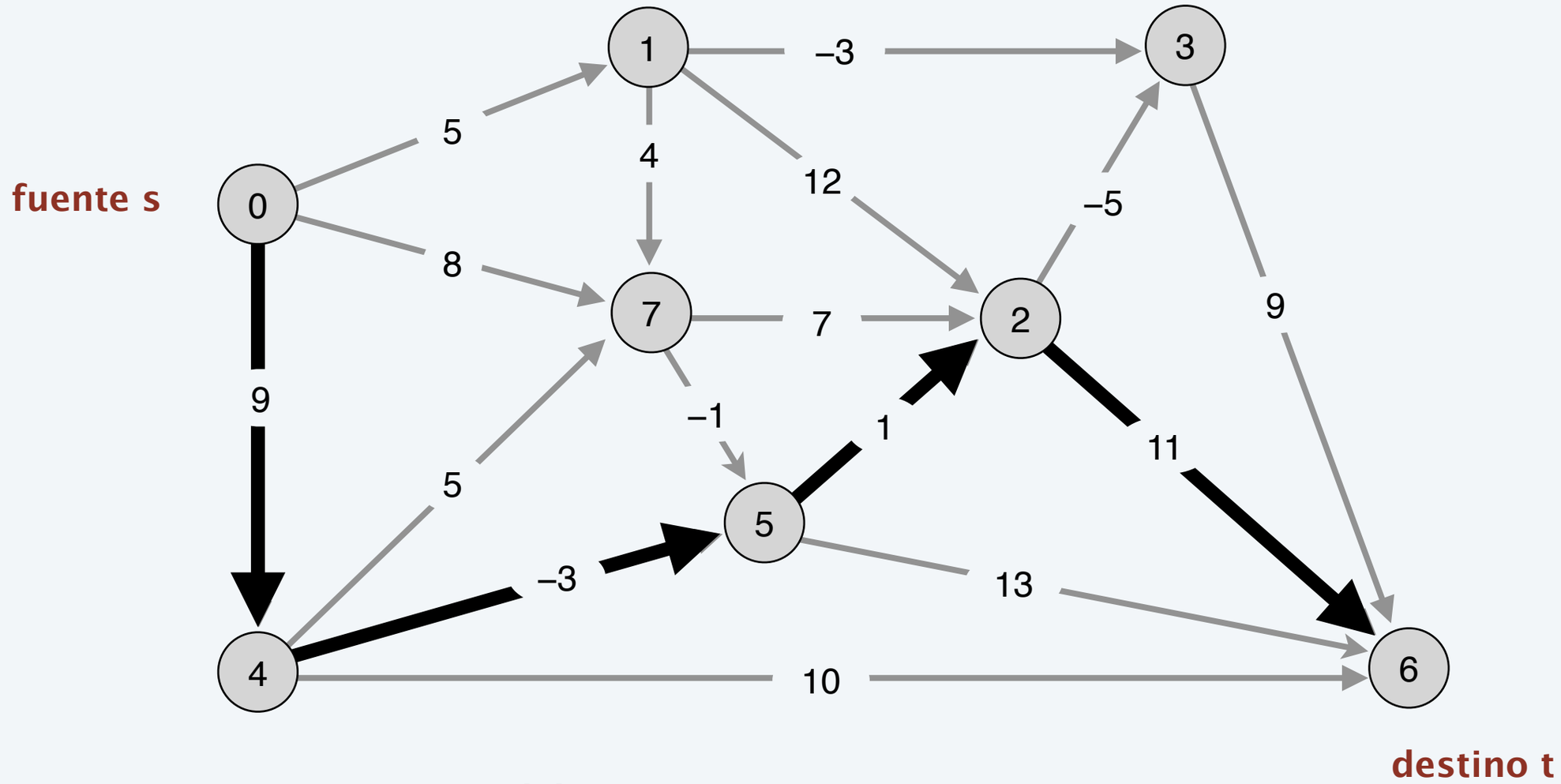


6. Programación Dinámica II

- *Estructura secundaria de ARN*
- *Alineamiento de secuencias*
- *Algoritmo de Bellman-Ford*

Caminos de costo mínimo

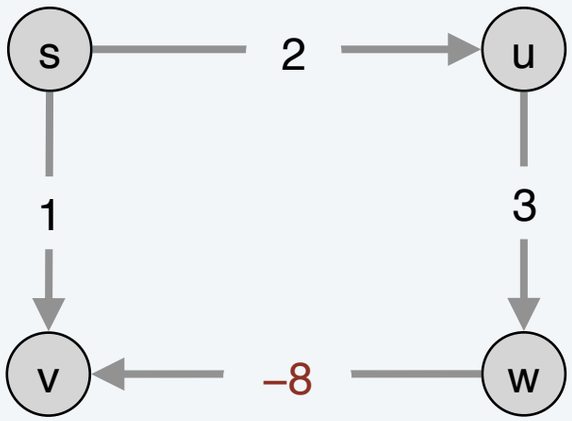
Problema del camino menos costoso. Dado un grafo dirigido $G = (V, E)$, con pesos o costos arbitrarios en las aristas, c_{vw}, encontrar el camino más barato de s a t.



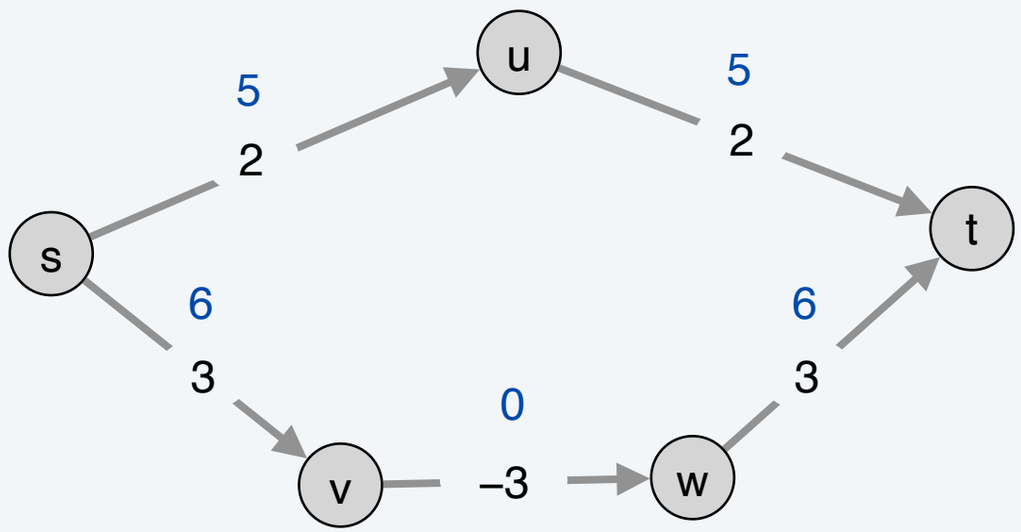
costo del camino = 9 − 3 + 1 + 11 = 18

Caminos de costo mínimo: intentos fallidos

Dijkstra. Puede fallar si hay costos negativos.



Traslación de pesos. Sumar una constante a los pesos no preserva el orden entre costo de los caminos (caminos más largos sufren mayores incrementos).

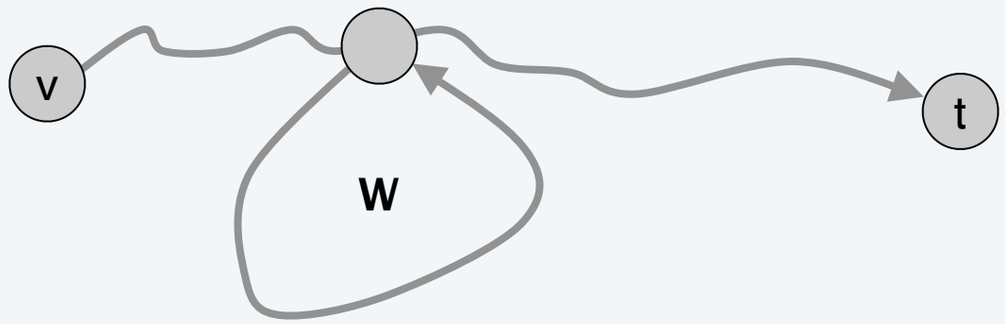


Caminos de costo mínimo / ciclos de costo negativo

Lema. Si G no tiene ciclos de costo negativo, existe un camino de costo mínimo de v a t que es simple (como no repite vértices tiene a lo sumo $n - 1$ aristas).

Demostración.

- Considerar un camino P de $v \rightsquigarrow t$ de costo mínimo con la menor cantidad de aristas posible.
- Si P contiene un ciclo W, podemos removerlo de P sin incrementar el costo. ▪



Caminos de costo mínimo: programación dinámica

Def. $OPT(i, v)$ = costo mínimo de camino $v \rightsquigarrow t$ usando a lo sumo i aristas.

- Caso 1: El costo mínimo se alcanza usando menos de i aristas.
  - $OPT(i, v) = \text{OPT}(i - 1, v)$

- Caso 2: El costo mínimo lo alcanza un camino que usa exactamente i aristas.
  - Si (v, w) es la primera arista de tal camino, a continuación se sigue por el mejor camino $w \rightsquigarrow t$ usando a lo sumo $i - 1$ aristas.

$$OPT(i, v) = \infty, \qquad i = 0, v \neq t$$

$$OPT(i, v) = 0, \qquad i = 0, v = t$$

$$OPT(i, v) = \min \left\{ OPT(i-1, v), \min_{w:(v,w)\in E} \{OPT(i-1, w) + c_{vw}\} \right\}, \text{ en otro caso}$$

Objetivo. Si no hay ciclos de costo negativo, $OPT(n - 1, v)$ es el costo mínimo de un camino $v \rightsquigarrow t$.

Caminos de costo mínimo: implementación bottom-up

SHORTEST-PATHS (V, E, c, s, t)

FOREACH node $v \in V$

  $M[0, v] \leftarrow \infty$.

$M[0, t] \leftarrow 0$.

FOR i = 1 TO $n - 1$

  FOREACH node $v \in V$

    $M[i, v] \leftarrow M[i-1, v]$.

    FOREACH edge $(v, w) \in E$

      $M[i, v] \leftarrow \min \{ M[i, v],\ M[i-1, w] + c_{vw} \}$.

RETURN $M[n\text{-}1, s]$.

Caminos de costo mínimo: extras

Obtención de camino de costo mínimo.

- Opción 1: Mantener un puntero $siguiente(i, v)$ que apunta al siguiente nodo en el camino $v \rightsquigarrow t$ usando a lo sumo i aristas.
- Opción 2: Calcular los costos $M[i, v]$ y desde v considerar solo las aristas que satisfacen $M[i, v] = M[i-1, w] + c_{vw}$.

Optimización de espacio

- Mantener vectores unidimensionales $M[v]$, $siguiente(v)$, y actualizarlos en cada iteración in situ sin mantener los valores de iteraciones anteriores.
- Actualizar $siguiente(v)$ cada vez que $M[v]$ decrece estrictamente.

Programación dinámica: Resumen

Técnicas generales.

- Elección binaria (ej. planificación de intervalos)
- Elección de múltiples vías (ej. mínimos cuadrados segmentado)
- Agregado de variable adicional (ej. problema de la mochila).
- Descomposición en subintervalos (ej. estructuras secundarias en ARN)

Otros ejemplos.

- Alineamiento de secuencias.
- Caminos de costo mínimo en un grafo dirigido sin ciclos de costo negativo.

Programación dinámica: Algunos puntos importantes a recordar

Descomposición del problema.

- Definir semánticamente una función para descomponer el problema. Por ejemplo, "$OPT(j)$ = Máximo valor de …."
- Definir cómo se obtiene el valor que buscamos para el problema completo. Por ejemplo, “la solución que buscamos es $OPT(n)$”
- Definir una relación de recurrencia para OPT, especificando claramente los casos en que aplica cada caso de la recurrencia.

Algoritmo para calcular el valor de una solución óptima.

- Llenar los valores de la tabla correspondientes a casos base.
- Establecer un orden de llenado de la tabla tal que, al momento de calcular una entrada de la tabla, todos los términos de la relación de recurrencia necesarios para el cálculo ya fueron calculados previamente.
- Obtener el valor para el problema completo de la tabla. Por ejemplo, $M[\ n\ ]$.

Algoritmo para obtener una solución óptima.

- Reconstruir las decisiones que llevan a un valor óptimo, partiendo desde el valor para el problema completo y avanzando en orden inverso al que se llenó la tabla.