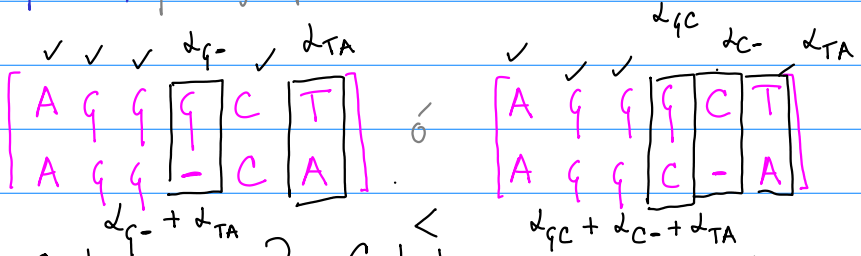
Más ejemplos de programación dinámica:

en un alfabeto fijo $\{A, C, T, G, -\} =: \Sigma$ (sucesiones de ADN).

Tenemos dos cadenas de símbolos (si se quiere de distinta longitud)

A G G G C T y A G G C A

Queremos alinearlos de la mejor manera posible, por ejemplo




Cuál es mejor? Calculamos el costo total de los errores, de acuerdo a reglas fijas $\alpha_{xy}$ para $x \in \{A, G, C, T, -\}$ y $\{A, G, C, T, -\}$.

**Problema:** [Sequence Alignment Problem]
Dadas palabras $X, Y$ encuentre un alignment de mínimo costo total. $\text{Seq}(X, Y)$ $\left[\begin{matrix}\alpha_{xy} \geq 0 \\ \alpha_{x-} > 0\end{matrix}\right]$

**Idea:** Suponga que $X = x_1 \ldots \boxed{x_m}$ es óptimo.
$Y = y_1 \ldots \boxed{y_m}$

Cómo es la última columna?

En un arrangement óptimo no sucede $\begin{bmatrix} x_m \\ y_m \end{bmatrix} = \begin{bmatrix} - \\ - \end{bmatrix}$,
(porque quitando esa columna el costo bajaría)

Nos quedan entonces las siguientes posibilidades:

**Caso 1:** $\begin{bmatrix} x_m \\ y_m \end{bmatrix} = \begin{bmatrix} - \\ s \end{bmatrix}$ con $s \in \Sigma$ último símbolo en $Y$

y las sucesiones $X, Y'$ se alinean óptimamente con las columnas anteriores.

$$Seq(X,Y) = \alpha_{-s} + Seq(X,Y')$$

**Caso 2:** $\begin{bmatrix} x_m \\ y_m \end{bmatrix} = \begin{bmatrix} s \\ - \end{bmatrix}$

$$Seq(X,Y) = \alpha_{s-} + Seq(X',Y)$$

**Caso 3:** $\begin{bmatrix} x_m \\ y_m \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}$

$$Seq(X,Y) = \alpha_{s_1 s_2} + Seq(X',Y')$$

**Teorema:** Ecuación de Bellman para $Seq(X,Y)$

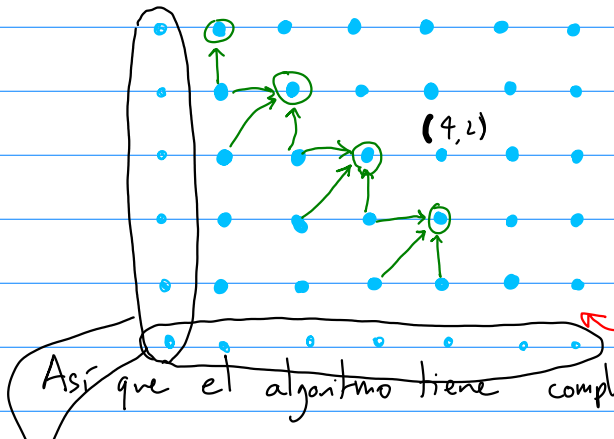
$$Seq(X,Y) = \max\left\{ \alpha_{-y_{last}} + Seq(X,Y'),\ \alpha_{x_{last}-} + Seq(X',Y),\ \alpha_{x_{last},y_{last}} + Seq(X',Y') \right\}$$

¿Qué estructura tiene nuestra colección de llamados?

$Seq(X,Y)$ solo necesita saber cuántos símbolos de cada cadena arrastra usar, es decir si

$X = AGGGCT \leftarrow$ 6 símbolos

$Y = AGGCA \leftarrow$ 5 símbolos




$Seq(4,2) = Seq(AGGG, AG)$

**Obs:** La estructura de la ecuación de Bellman nos dice como deberíamos arrastrar los problemas para calcularlos eficientemente.

Así que el algoritmo tiene complejidad $O(m_1 m_2)$

como función de las longitudes $m_1, m_2$ de las entradas.

$Seq(0,j) = j\alpha_-$

$Seq(i,0) = i\alpha_-$

Ejemplo: Suponga que hay un penalty de $\alpha_- = 1$ y un penalty $\alpha_{xy} = 2 \ \forall x \neq y$.

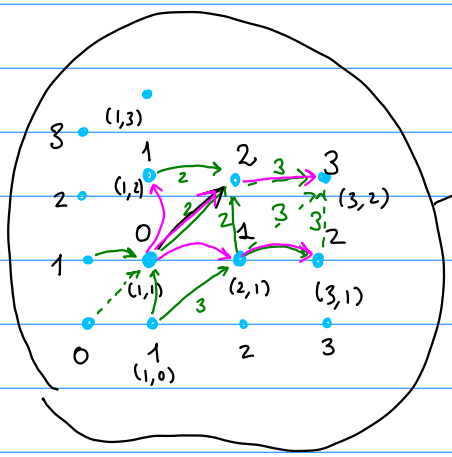
Seq (AGTACG, ACATAG)

$$S(2,2) = \max\left( \underset{[\,-\,]}{1 + S(1,2)}, \underset{[G\,-]}{1 + S(2,1)}, \underset{[G\,C]}{2 + S(1,1)} \right)$$

$$S(AG, AC) = c\left(\begin{bmatrix} - \\ C \end{bmatrix}\right) + S(AG, A)$$

$$\begin{bmatrix} AG \\ A- \end{bmatrix}$$

$\| \ S(2,2)$



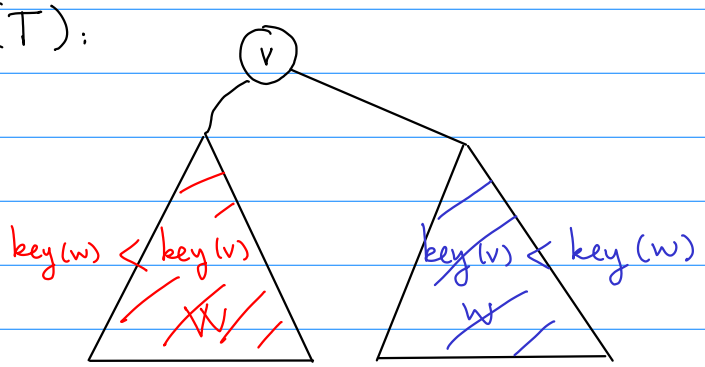

Puede imaginarse como un problema de Dijkstra de distancia mínima en el grafo de "estados".

✗ Dijkstra crece a lo largo de las diagonales

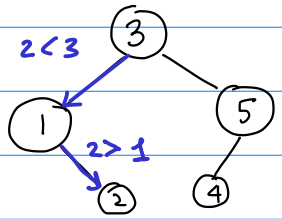
## Problema 4: [Árboles de búsqueda]

**Def:** Un "árbol de búsqueda" es un árbol binario cuyos nodos están marcados con números enteros (o más generalmente key(v) pertenece a un conjunto totalmente ordenado) y además cumple la siguiente propiedad:

$\forall v \in V(T)$:




Ejemplo:



Esto es muy útil porque al buscar z empezando por la raíz veo inmediatamente ↘ del diagrama arbol así que emplearé $O(\log(n))$ pasos de búsqueda para n elementos.

TABLA tiempo de búsqueda:

| | t |
|---|---|
| 3 | 1 |
| 1 | 2 |
| 2 | 3 |
| 5 | 1 |
| 4 | 2 |

Bueno si 3 y 5 más probables

Malo si 1 y 2 más probables.

Queremos usar información estadística sobre los datos para construir buenas maneras de almacenarlos.

**Problema: [Optimal binary search trees]**

Dadas: Llaves ordenadas $k_1 < k_2 < \dots < k_n$ con Distrib de probabilidad sobre las llaves. $D(n)$

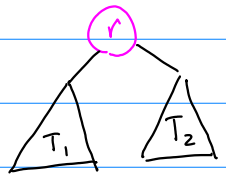
probabilidades $p_1 \; p_2 \; \dots \; p_n$ $(p_i \geq 0, \; \sum p_i = 1)$

Produzca un arbol binario T con llaves $k_i$ que minimice el tiempo esperado de búsqueda "$p_i$ de una llave con distribución $\mathbb{P}\{K = k_i\}$.

$\Lambda(D(n)) = \min \{ \mathbb{E}[t(T)] : T$ es árbol de búsqueda para $k_1 < \dots < k_n \}$

Suponga que $T^*$ es óptimo. Qué vértice esta en la raiz?

$$t(T) = \sum_{s \in V(T)} p_s \cdot \underbrace{\text{nivel}(s)}_{\text{número de pasos de búsqueda.}}$$




r podría ser cualquiera de las llaves

$T_1$ árbol de búsqueda para $k_1 < \dots < k_{r-1}$

$T_2$ árbol " $k_{r+1} < \dots < k_n$

$T_1$ y $T_2$ deben ser óptimos pues de lo contrario contradeciríamos la optimalidad de T

Calculemos el tiempo esperado de búsqueda:

$$\sum_{j \le r-1} p_j \left(1 + \text{tiempo de } j \text{ en } T_1\right) + p_r \cdot 1 + \sum_{j \ge r+1} p_j \left(1 + \text{tiempo de } j \text{ en } T_2\right)$$

$$= \sum_{j=1}^{n} p_j + t(T_1) + t(T_2)$$

$$\Lambda(D[1,n]) = \max_{1 \le j \le n} \left\{ \sum_{j=1}^{n} p_j + \Lambda(D[1,j-1]) + \Lambda(D[j+1,n]) \right\}$$

↑ Ecuación de Bellman!

donde

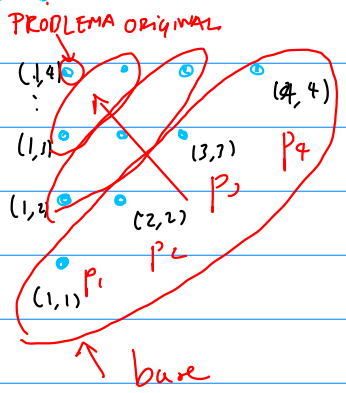
$D[i,j] :=$ Parte de la tabla formada por columnas $i, i+1, \ldots, j$, es decir

$k_i < k_{i+1} < \cdots < k_j$

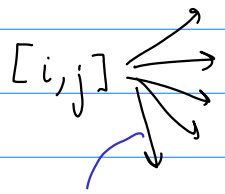
$p_i \quad p_{i+1} \quad p_j \ge 0$

Cuál es el espacio de estados?
El problema está completamente determinado por las parejas $[i,j]$ con $1 \le i \le j \le n$

Espacio de estados: Intervalos $[i,j]$ con $1 \le i \le j \le n$




$O(n^2)$ estados




varias costes dependiendo de por cuál partimos

El tiempo de ejecución es $O(n^3)$ porque cada mínimo tiene $O(n)$ términos y hay $O(n^2)$ estados.