

Programación dinámica: un segundo ingrediente

En general, programación dinámica es usada para resolver problemas de optimización.

Programación dinámica: un segundo ingrediente

En general, programación dinámica es usada para resolver problemas de optimización.

Para que un problema de optimización pueda ser resuelto con esta técnica se debe cumplir el siguiente **principio de optimalidad para los sub-problemas**:

Una solución óptima para un problema contiene soluciones óptimas para sus sub-problemas

Programación dinámica: un segundo ingrediente

En general, programación dinámica es usada para resolver problemas de optimización.

Para que un problema de optimización pueda ser resuelto con esta técnica se debe cumplir el siguiente **principio de optimalidad para los sub-problemas**:

Una solución óptima para un problema contiene soluciones óptimas para sus sub-problemas

Vamos a ver un ejemplo de este principio que enfatiza otra característica de programación dinámica: en general los problemas deben ser resueltos de forma bottom-up.

Midiendo la distancia entre dos palabras

Sea $\Sigma = \{a, b, \dots, z\}$ el alfabeto español, el cual contiene 27 símbolos.

- Vamos a considerar las palabras (strings) sobre el alfabeto Σ

Midiendo la distancia entre dos palabras

Sea $\Sigma = \{a, b, \dots, z\}$ el alfabeto español, el cual contiene 27 símbolos.

- Vamos a considerar las palabras (strings) sobre el alfabeto Σ

Vamos a utilizar distancia de Levenshtein para medir cuán similares son dos palabras.

- Esta es una de las medidas de similitud de palabras más populares por lo que usualmente es llamada *edit distance*

Midiendo la distancia entre dos palabras

Sea $\Sigma = \{a, b, \dots, z\}$ el alfabeto español, el cual contiene 27 símbolos.

- ▶ Vamos a considerar las palabras (strings) sobre el alfabeto Σ

Vamos a utilizar distancia de Levenshtein para medir cuán similares son dos palabras.

- ▶ Esta es una de las medidas de similitud de palabras más populares por lo que usualmente es llamada *edit distance*

Dadas dos palabras $w_1, w_2 \in \Sigma^*$, utilizamos la notación $\text{ed}(w_1, w_2)$ para la edit distance entre w_1 y w_2

- ▶ Tenemos que $\text{ed} : \Sigma^* \times \Sigma^* \rightarrow \mathbb{N}$

Tres operadores sobre palabras

Sea $w \in \Sigma^*$ con $w = a_1 \cdots a_n$ y $n \geq 0$

- Si $n = 0$ entonces $w = \varepsilon$

Tres operadores sobre palabras

Sea $w \in \Sigma^*$ con $w = a_1 \cdots a_n$ y $n \geq 0$

► Si $n = 0$ entonces $w = \varepsilon$

Para $i \in \{1, \dots, n\}$ y $b \in \Sigma$ tenemos que:

$$\text{eliminar}(w, i) = a_1 \cdots a_{i-1} a_{i+1} \cdots a_n$$

$$\text{agregar}(w, i, b) = a_1 \cdots a_i b a_{i+1} \cdots a_n$$

$$\text{cambiar}(w, i, b) = a_1 \cdots a_{i-1} b a_{i+1} \cdots a_n$$

Además, tenemos que $\text{agregar}(w, 0, b) = bw$

Tres operadores sobre palabras

Sea $w \in \Sigma^*$ con $w = a_1 \cdots a_n$ y $n \geq 0$

► Si $n = 0$ entonces $w = \varepsilon$

Para $i \in \{1, \dots, n\}$ y $b \in \Sigma$ tenemos que:

$$\text{eliminar}(w, i) = a_1 \cdots a_{i-1} a_{i+1} \cdots a_n$$

$$\text{agregar}(w, i, b) = a_1 \cdots a_i b a_{i+1} \cdots a_n$$

$$\text{cambiar}(w, i, b) = a_1 \cdots a_{i-1} b a_{i+1} \cdots a_n$$

Además, tenemos que $\text{agregar}(w, 0, b) = bw$

Ejemplo

$\text{eliminar}(\text{hola}, 1)$	$=$	ola	$\text{eliminar}(\text{hola}, 3)$	$=$	hoa
$\text{agregar}(\text{hola}, 0, y)$	$=$	yhola	$\text{agregar}(\text{hola}, 3, z)$	$=$	holza
$\text{cambiar}(\text{hola}, 2, x)$	$=$	hxla			

La distancia de Levenshtein

Dadas palabras $w_1, w_2 \in \Sigma^*$, definimos $ed(w_1, w_2)$ como el menor número de operaciones eliminar, agregar y cambiar que aplicadas desde w_1 generan w_2

Ejemplo

Tenemos que $ed(\text{casa}, \text{asado}) = 3$ puesto que:

agregar(casa, 4, d) = casad

eliminar(casad, 1) = asad

agregar(asad, 4, o) = asado

La distancia de Levenshtein

Ejercicios

1. Demuestre que $\text{ed}(w_1, w_2) \leq |w_1| + |w_2|$
2. Demuestre que ed es una función de distancia, vale decir, muestre lo siguiente:
 - 2.1 $\text{ed}(w_1, w_2) = 0$ si y sólo si $w_1 = w_2$
 - 2.2 $\text{ed}(w_1, w_2) = \text{ed}(w_2, w_1)$
 - 2.3 $\text{ed}(w_1, w_2) \leq \text{ed}(w_1, w_3) + \text{ed}(w_3, w_2)$
3. Dadas dos palabras w_1, w_2 del mismo largo, la distancia de Hamming entre w_1 y w_2 es definida como el número de posiciones en las que tienen distintos símbolos. Denote esta distancia como $\text{hd}(w_1, w_2)$
 - 3.1 Demuestre que $\text{ed}(w_1, w_2) \leq \text{hd}(w_1, w_2)$
 - 3.2 Encuentre palabras w_3 y w_4 tales que $\text{ed}(w_3, w_4) < \text{hd}(w_3, w_4)$

Calculando la distancia de Levenshtein

Para calcular $ed(w_1, w_2)$ no podemos considerar todas las posibles secuencias de operaciones que aplicadas desde w_1 generan w_2

- ▶ Incluso si consideramos las secuencias de largo a los más $|w_1| + |w_2|$ vamos a tener demasiadas posibilidades

Calculando la distancia de Levenshtein

Para calcular $ed(w_1, w_2)$ no podemos considerar todas las posibles secuencias de operaciones que aplicadas desde w_1 generan w_2

- Incluso si consideramos las secuencias de largo a los más $|w_1| + |w_2|$ vamos a tener demasiadas posibilidades

Para calcular $ed(w_1, w_2)$ utilizamos programación dinámica.

Notación preliminar

Dado $w \in \Sigma^*$ tal que $|w| = n$, y dado $i \in \{1, \dots, n\}$, definimos $w[i]$ como el símbolo de w en la posición i

► Tenemos que $w = w[1] \cdots w[n]$

Además, definimos el infijo de w entre las posiciones i y j como:

$$w[i, j] = \begin{cases} w[i] \cdots w[j] & 1 \leq i \leq j \leq n \\ \varepsilon & \text{en caso contrario} \end{cases}$$

La distancia y un principio optimalidad para sub-secuencias

Sean $w_1, w_2 \in \Sigma^*$, y suponga que $o_1, \dots, o_k$ es una secuencia óptima de operaciones que aplicadas desde w_1 generan w_2 con $k \geq 1$

- Tenemos que $\text{ed}(w_1, w_2) = k$

La distancia y un principio optimalidad para sub-secuencias

Sean $w_1, w_2 \in \Sigma^*$, y suponga que $o_1, \dots, o_k$ es una secuencia óptima de operaciones que aplicadas desde w_1 generan w_2 con $k \geq 1$

► Tenemos que $\text{ed}(w_1, w_2) = k$

Considere $0 \leq i \leq |w_1|$ y suponga que $o_1, \dots, o_\ell$ es la sub-secuencia de las operaciones en $o_1, \dots, o_k$ que son aplicadas a $w_1[1, i]$ con $1 \leq \ell \leq k$

La distancia y un principio optimalidad para sub-secuencias

Sean $w_1, w_2 \in \Sigma^*$, y suponga que $o_1, \dots, o_k$ es una secuencia óptima de operaciones que aplicadas desde w_1 generan w_2 con $k \geq 1$

► Tenemos que $\text{ed}(w_1, w_2) = k$

Considere $0 \leq i \leq |w_1|$ y suponga que $o_1, \dots, o_\ell$ es la sub-secuencia de las operaciones en $o_1, \dots, o_k$ que son aplicadas a $w_1[1, i]$ con $1 \leq \ell \leq k$

Estamos suponiendo que las operaciones sobre $w_1[1, i]$ son las primeras en ser aplicadas, ¿por qué podemos hacer este supuesto?

La distancia y un principio optimalidad para sub-secuencias

Podemos realizar el supuesto por la siguiente razón: si para $s \in \{1, \dots, \ell - 1\}$ tenemos que o_{s+1} es una operación sobre $w_1[1, i]$ pero o_s no lo es, entonces podemos permutar estas dos operaciones para generar o_{s+1}, o_s^* tal que:

- ▶ si o_{s+1} cambia un símbolo por otro, entonces $o_s^* = o_s$
- ▶ si o_{s+1} agrega un símbolo, entonces o_s^* se obtiene desde o_s cambiando la posición $t \in \{1, \dots, n\}$ mencionada en o_s por $t + 1$
- ▶ si o_{s+1} elimina un símbolo, entonces o_s^* se obtiene desde o_s cambiando la posición $t \in \{1, \dots, n\}$ mencionada en o_s por $t - 1$

La distancia y un principio optimalidad para sub-secuencias

Podemos realizar el supuesto por la siguiente razón: si para $s \in \{1, \dots, \ell - 1\}$ tenemos que o_{s+1} es una operación sobre $w_1[1, i]$ pero o_s no lo es, entonces podemos permutar estas dos operaciones para generar o_{s+1}, o_s^* tal que:

- ▶ si o_{s+1} cambia un símbolo por otro, entonces $o_s^* = o_s$
- ▶ si o_{s+1} agrega un símbolo, entonces o_s^* se obtiene desde o_s cambiando la posición $t \in \{1, \dots, n\}$ mencionada en o_s por $t + 1$
- ▶ si o_{s+1} elimina un símbolo, entonces o_s^* se obtiene desde o_s cambiando la posición $t \in \{1, \dots, n\}$ mencionada en o_s por $t - 1$

Finalmente, suponga que la aplicación de $o_1, \dots, o_\ell$ desde $w_1[1, i]$ genera $w_2[1, j]$ con $0 \leq j \leq |w_2|$

La distancia y un principio optimalidad para sub-secuencias

Entonces la sub-secuencia $o_1, \dots, o_\ell$ debe ser óptima para la generación de $w_2[1, j]$ a partir de $w_1[1, i]$

► Vale decir, $\text{ed}(w_1[1, i], w_2[1, j]) = \ell$

La distancia y un principio optimalidad para sub-secuencias

Entonces la sub-secuencia $o_1, \dots, o_\ell$ debe ser óptima para la generación de $w_2[1, j]$ a partir de $w_1[1, i]$

► Vale decir, $\text{ed}(w_1[1, i], w_2[1, j]) = \ell$

Suponga que lo anterior no es cierto, y suponga que $o'_1, \dots, o'_m$ es una secuencia de operaciones con $m < \ell$ que genera $w_2[1, j]$ desde $w_1[1, i]$

La distancia y un principio optimalidad para sub-secuencias

Entonces la sub-secuencia $o_1, \dots, o_\ell$ debe ser óptima para la generación de $w_2[1, j]$ a partir de $w_1[1, i]$

► Vale decir, $\text{ed}(w_1[1, i], w_2[1, j]) = \ell$

Suponga que lo anterior no es cierto, y suponga que $o'_1, \dots, o'_m$ es una secuencia de operaciones con $m < \ell$ que genera $w_2[1, j]$ desde $w_1[1, i]$

En la secuencia $o_1, \dots, o_k$ que genera w_2 desde w_1 podemos reemplazar $o_1, \dots, o_\ell$ por $o'_1, \dots, o'_m$

► La secuencia resultante se puede utilizar para generar w_2 desde w_1

La distancia y un principio optimalidad para sub-secuencias

Entonces la sub-secuencia $o_1, \dots, o_\ell$ debe ser óptima para la generación de $w_2[1, j]$ a partir de $w_1[1, i]$

► Vale decir, $\text{ed}(w_1[1, i], w_2[1, j]) = \ell$

Suponga que lo anterior no es cierto, y suponga que $o'_1, \dots, o'_m$ es una secuencia de operaciones con $m < \ell$ que genera $w_2[1, j]$ desde $w_1[1, i]$

En la secuencia $o_1, \dots, o_k$ que genera w_2 desde w_1 podemos reemplazar $o_1, \dots, o_\ell$ por $o'_1, \dots, o'_m$

► La secuencia resultante se puede utilizar para generar w_2 desde w_1

Concluimos que $\text{ed}(w_1, w_2) \leq (k - \ell) + m = k - (\ell - m) < k$, lo que contradice el supuesto inicial.

Una definición recursiva de la distancia de Levenshtein

Fije dos strings $w_1, w_2 \in \Sigma^*$ tales que $|w_1| = m$ y $|w_2| = n$

Dados $i \in \{0, \dots, m\}$ y $j \in \{0, \dots, n\}$, definimos

$$\text{ed}(i, j) = \text{ed}(w_1[1, i], w_2[1, j])$$

Observe que $\text{ed}(w_1, w_2) = \text{ed}(m, n)$

Además, definimos el valor $\text{dif}(i, j)$ como 0 si $w_1[i] = w_2[j]$, y como 1 en caso contrario.

Una definición recursiva de la distancia de Levenshtein

Del principio de optimalidad para sub-secuencias obtenemos la siguiente definición recursiva para la función ed :

$$ed(i, j) = \begin{cases} \text{máx}\{i, j\} & i = 0 \text{ o } j = 0 \\ \text{mín}\{1 + ed(i - 1, j), \\ \quad 1 + ed(i, j - 1), \\ \quad \text{dif}(i, j) + ed(i - 1, j - 1)\} & i > 0 \text{ y } j > 0 \end{cases}$$

Una definición recursiva de la distancia de Levenshtein

Del principio de optimalidad para sub-secuencias obtenemos la siguiente definición recursiva para la función ed :

$$ed(i, j) = \begin{cases} \text{máx}\{i, j\} & i = 0 \text{ o } j = 0 \\ \text{mín}\{1 + ed(i - 1, j), \\ \quad 1 + ed(i, j - 1), \\ \quad \text{dif}(i, j) + ed(i - 1, j - 1)\} & i > 0 \text{ y } j > 0 \end{cases}$$

Tenemos entonces una forma de calcular la función ed que se basa en resolver sub-problemas más pequeños

Una definición recursiva de la distancia de Levenshtein

Del principio de optimalidad para sub-secuencias obtenemos la siguiente definición recursiva para la función ed:

$$\text{ed}(i, j) = \begin{cases} \text{máx}\{i, j\} & i = 0 \text{ o } j = 0 \\ \text{mín}\{1 + \text{ed}(i - 1, j), \\ \quad 1 + \text{ed}(i, j - 1), \\ \quad \text{dif}(i, j) + \text{ed}(i - 1, j - 1)\} & i > 0 \text{ y } j > 0 \end{cases}$$

Tenemos entonces una forma de calcular la función ed que se basa en resolver sub-problemas más pequeños

- ▶ Estos sub-problemas están traslapados y se tiene un número polinomial de ellos, podemos aplicar entonces programación dinámica

Una implementación recursiva de la distancia de Levenshtein

```
EditDistance( $w_1, i, w_2, j$ )  
  if  $i = 0$  then return  $j$   
  else if  $j = 0$  then return  $i$   
  else  
     $r := \text{EditDistance}(w_1, i - 1, w_2, j)$   
     $s := \text{EditDistance}(w_1, i, w_2, j - 1)$   
     $t := \text{EditDistance}(w_1, i - 1, w_2, j - 1)$   
    if  $w_1[i] = w_2[j]$  then  $d := 0$   
    else  $d := 1$   
    return  $\min\{1 + r, 1 + s, d + t\}$ 
```

Una implementación recursiva de la distancia de Levenshtein

```
EditDistance( $w_1, i, w_2, j$ )  
  if  $i = 0$  then return  $j$   
  else if  $j = 0$  then return  $i$   
  else  
     $r := \text{EditDistance}(w_1, i - 1, w_2, j)$   
     $s := \text{EditDistance}(w_1, i, w_2, j - 1)$   
     $t := \text{EditDistance}(w_1, i - 1, w_2, j - 1)$   
    if  $w_1[i] = w_2[j]$  then  $d := 0$   
    else  $d := 1$   
    return  $\min\{1 + r, 1 + s, d + t\}$ 
```

¿Es esta una buena implementación de **EditDistance**?

Una implementación recursiva de la distancia de Levenshtein

```
EditDistance( $w_1, i, w_2, j$ )  
  if  $i = 0$  then return  $j$   
  else if  $j = 0$  then return  $i$   
  else  
     $r := \text{EditDistance}(w_1, i - 1, w_2, j)$   
     $s := \text{EditDistance}(w_1, i, w_2, j - 1)$   
     $t := \text{EditDistance}(w_1, i - 1, w_2, j - 1)$   
    if  $w_1[i] = w_2[j]$  then  $d := 0$   
    else  $d := 1$   
    return  $\min\{1 + r, 1 + s, d + t\}$ 
```

¿Es esta una buena implementación de **EditDistance**?

- No porque tenemos muchas llamadas recursivas repetidas, es mejor evaluar esta función utilizando un enfoque bottom-up

Una evaluación bottom-up de la distancia de Levenshtein

Para determinar los valores de la función ed construimos una tabla siguiendo un orden lexicográfico para los pares (i, j) :

$$(i_1, j_1) < (i_2, j_2) \quad \text{si y sólo si} \quad i_1 < i_2 \text{ o } (i_1 = i_2 \text{ y } j_1 < j_2)$$

Una evaluación bottom-up de la distancia de Levenshtein

Para determinar los valores de la función ed construimos una tabla siguiendo un orden lexicográfico para los pares (i, j) :

$$(i_1, j_1) < (i_2, j_2) \quad \text{si y sólo si} \quad i_1 < i_2 \text{ o } (i_1 = i_2 \text{ y } j_1 < j_2)$$

Por ejemplo, para determinar el valor de $ed(\text{casa}, \text{asado})$ construimos la siguiente tabla:

		a	s	a	d	o
c a s a	0	1	2	3	4	5
	1	1	2	3	4	5
	2	1	2	2	3	4
	3	2	1	2	3	4
	4	3	2	1	2	3

Una evaluación bottom-up de la distancia de Levenshtein

A partir de la tabla podemos obtener una secuencia óptima de operaciones para transformar casa en asado:

		a	s	a	d	o
c a s a	0	1	2	3	4	5
	1	1	2	3	4	5
	2	2	1	2	3	4
	3	3	2	1	2	3
a	4	4	3	2	1	2
	5	5	4	3	2	1
	6	6	5	4	3	2
	7	7	6	5	4	3

Una evaluación bottom-up de la distancia de Levenshtein

A partir de la tabla podemos obtener una secuencia óptima de operaciones para transformar casa en asado:

		a	s	a	d	o
c	0	1	2	3	4	5
a	1	1	2	2	3	4
s	2	1	2	2	3	4
a	3	2	1	2	3	4
	4	3	2	1	2	3

Estas operaciones son las siguientes:

eliminar(casa, 1) = asa
agregar(asa, 3, d) = asad
agregar(asad, 4, o) = asado

El costo de calcular la distancia de Levenshtein

¿Qué operaciones debemos contar al analizar la complejidad del algoritmo discutido en las transparencias anteriores?

El costo de calcular la distancia de Levenshtein

¿Qué operaciones debemos contar al analizar la complejidad del algoritmo discutido en las transparencias anteriores?

- ▶ Consideramos como operaciones básicas acceder a una celda de la tabla (para leer o escribir), realizar operaciones con elementos de la tabla (sumar o comparar) y comparar elementos de las palabras de entrada

El costo de calcular la distancia de Levenshtein

¿Qué operaciones debemos contar al analizar la complejidad del algoritmo discutido en las transparencias anteriores?

- Consideramos como operaciones básicas acceder a una celda de la tabla (para leer o escribir), realizar operaciones con elementos de la tabla (sumar o comparar) y comparar elementos de las palabras de entrada

Corolario

Utilizando programación dinámica es posible construir un algoritmo para calcular $ed(w_1, w_2)$ que en el peor caso es $\Theta(|w_1| \cdot |w_2|)$