CURSO DE PROGRAMACIÓN COMPETITIVA
URJC - 2019

Sesión 7 (12ª Semana)

David Morán (ddavidmorang@gmail.com)
Juan Quintana (juandavid.quintana@urjc.es)
Sergio Pérez (sergio.perez.pelo@urjc.es)
Jesús Sánchez-Oro (jesus.sanchezoro@urjc.es)

Contenidos

- Programación Dinámica
  - Ejemplos
    - Edit distance
    - Longest Common Subsequence
    - Longest Increasing Subsequence
  - Retorno con choice
    - +Ejemplos

Programación Dinámica

Clase pasada

- ¿Qué es la técnica de DP?
- ¿Cuándo usarla?
- ¿Diferencia entre memoización y tabulación?

Programación Dinámica

Definición de términos

- **Estado**: Estado actual en el que se encuentra el DP antes de tomar cualquier decisión
- **Transición**: Cambios que deben hacerse para ir a otro subproblema
- **Memo**: Estructura donde se guarda la respuesta de los subproblemas
- **Casos base**: Subproblema principal del que sabemos respuesta

Programación Dinámica

Problema de Edit Distance

- Dado un par de palabras y solo utilizando 3 operaciones (borrar letra, insertar letra, modificar letra), devolver la mínima cantidad de operaciones para cambiar la palabra A a B
    - La longitud de las palabras no excede 1000

Programación Dinámica

Algoritmo de Edit Distance

- Ejemplos
  - “GATA” v “PATATA”
  - “KITTEN” v “SITTING”
  - “KAMISAMA” v “CAMISA”

Programación Dinámica

Algoritmo de Edit Distance

- Ejemplo: “GATA” v “PATATA”
  - Insertar P en 0 (PGATA)
  - Insertar A en 1 (PAGATA)
  - Sustituir G por T (PATATA)
- 3 operaciones

Programación Dinámica

Algoritmo de Edit Distance

- Ejemplo: KITTEN v SITTING
  - Reemplazar K por S (SITTEN)
  - Reemplazar E por I (SITTIN)
  - Insertar G en 5 (SITTING)
- 3 operaciones

Programación Dinámica

Algoritmo de Edit Distance

- Ejemplo: KAMISAMA v CAMISA
  - Reemplazar K por C (CAMISAMA)
  - Eliminar M en 6 (CAMISAA)
  - Eliminar A en 6 (CAMISA)
- 3 operaciones

Programación Dinámica

¿Es DP?

- ¿Cómo manejamos los strings como estados? (No podemos memorizar strings)
  - ¡Discretizar strings!

Programación Dinámica

Definición de términos

- **Estado**: ?
- **Transición**: ?
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**:
  - Índice actual de las dos palabras
- **Transición**: ?
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: ?
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**:
  - Insertar (i, j+1) (1)
  - Borrar(i+1, j) (1)
  - Reemplazar (i+1, j+1) (1)
  - Dejar como está (i+1, j+1) (0)
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**:
  - Mínimo entre aplicar las 3 operaciones (si son aplicables)
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**: Conocido
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**: Conocido
- **Casos base**:
  - Si i1 >= |S1| (INF)
  - Si i2 >= |S2| (INF)
  - Si i1 == |S1| Y i2 == |S2| (0)

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**: Conocido
- **Casos base**: Conocido
- ¡Resolvamos el problema!

Programación Dinámica

```
function F(i, j):
    if i >= S.length && j >= T.length:
        return 0
    if i >= S.length or j >= T.length:
        return INF
    if memo[i][j] != -1
        return memo[i][j]
    if S[i] == T[j]:
        memo[i][j] = F(i+1, j+1)
    else:
        memo[i][j] = min(F(i+1, j)+1, F(i, j+1)+1, F(i+1,
    j+1)+1)
    return memo[i][j]
```

Programación Dinámica

Problema de Longest Common Subsequence

- Dado un par de secuencias A y B, determinar la subsecuencia más larga en A que existe en B
  - Las longitudes de las palabras no exceden 1000

Programación Dinámica

Longest Common Subsequence

- Ejemplos:
  - GATA v PATA
  - CALABAZA v MANITAS
  - METILENO v TOCHETO

Programación Dinámica

- GATA v PATA

Programación Dinámica

- CALABAZA v MANITAS

Programación Dinámica

- METILENO v TOCHETO

Programación Dinámica

Definición de términos

- **Estado**: ?
- **Transición**: ?
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**:
    - Índices de A y B
- **Transición**: ?
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: ?
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**:
  - Si A[i] == B[j], F(i+1, j+1)+1
  - Sino:
    - F(i+1, j) y F(i, j+1)
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**:
  - Si los dos son iguales, entonces F(i+1, j+1)+1
  - Sino, máximo entre F(i+1, j) y F(i, j+1)
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**: Conocido
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**: Conocido
- **Casos base**:
  - Si $i > |A|$ (0)
  - Si $j > |B|$ (0)

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**: Conocido
- **Casos base**: Conocido

Programación Dinámica

```
function LCS(i, j):
    if i >= S.length or j >= T.length:
        return 0
    if memo[i][j] != -1
        return memo[i][j]
    if S[i] == T[j]:
        memo[i][j] = F(i+1, j+1) + 1
    else:
        memo[i][j] = max(F(i+1, j), F(i, j+1))
    return memo[i][j]
```

versión bottom-up: `http://lcs-demo.sourceforge.net`

Programación Dinámica

Problema de Longest Increasing Subsequence

- Dada una lista de números, determinar la subsecuencia incremental más larga en dicha lista
  - La longitud de la lista no excede los 1000 caracteres

Programación Dinámica

Longest Increasing Subsequence

- Ejemplos:
  - { 1, 2, 3, 4, 5, 6 }
  - { 6, 5, 4, 3, 2, 1 }
  - { 1, 3, 5, 4, 2, 6 }

Programación Dinámica

- { 1, 2, 3, 4, 5, 6 }

Programación Dinámica

- { 6, 5, 4, 3, 2, 1 }

Programación Dinámica

- { 1, 3, 5, 4, 2, 6 }

Programación Dinámica

Definición de términos

- **Estado**: ?
- **Transición**: ?
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: ?
  - Índice actual
  - Índice previo de comparación
- **Transición**: ?
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: ?
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: ?
  - Continuar la secuencia si A[i] > A[j] con F(i+1,i)+1
  - Iniciar una nueva secuencia desde el índice actual con F(i+1, i)+0
  - Continuar la secuencia probando otro posible número F(i+1, j)
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**: ?
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**: ?
  - Si A[i] > A[j] entonces F(i+1, i) + 1
  - Sino:
    - Máximo entre F(i+1, j) y F(i+1, i)
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**: Conocido
- **Casos base**: ?

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**: Conocido
- **Casos base**:
  - Si el índice i llega hasta el final de la lista (0)

Programación Dinámica

Definición de términos

- **Estado**: Conocido
- **Transición**: Conocido
- **Memo**: Conocido
- **Casos base**: Conocido
- ¡Resolvamos el problema!

Programación Dinámica

```
function F(i, j):
    if i >= A.length:
        return 0
    if memo[i][j] != -1
        return memo[i][j]
    if A[i] > A[j]:
        memo[i][j] = F(i+1, i) + 1
    else:
        memo[i][j] = max(F(i+1, i), F(i+1, j))
    return memo[i][j]
```

Programación Dinámica

- Existe una variante más rápida de un LIS (NLgN)
  - Basada en búsquedas binarias dentro de un subconjunto ya encontrado
  - Se deja como tarea e investigación

Programación Dinámica

- Retorno con choice (elección)
- Nos pueden pedir que construyamos una solución que nos lleve a una respuesta óptima del problema
- Dicha construcción es más intuitiva de hacer con la técnica de Bottom-Up
- También se puede hacer con Top-Down

Programación Dinámica

- Necesitamos una estructura adicional exactamente igual a memo que nos guarde a que estado hemos tomado la decisión
  - Base: llamadas recursivas
- Requiere condicionales sobre qué llamada es mejor que otra
- Si A es mejor respuesta que B, definimos en nuestra elección del estado $E \rightarrow T_A$

Programación Dinámica

- Problema de la mochila
  - choice[objeto][peso] = ?
  - Ta = F(objeto + 1, peso)
  - Tb = F(objeto + 1, peso - P[objeto]) + V[objeto]
  - Si Ta > Tb → choice[objeto][peso] = (objeto+1, peso)
  - Sino → choice[objeto][peso] = (objeto+1, peso-P[objeto])

Programación Dinámica

```
function f(i, weight):
  if weight > W:
    return -INF
   if weight == W or i >= N:
    return 0
  if memo[i,weight] != -1:
    return memo[i,weight]
  memo[i,weight] = max(
    f(i+1, weight + W[i]) + V[i],
    f(i+1, weight))
  return memo[i,weight]
```

Programación Dinámica

```
function f(i, weight):
    if weight > W:
        return -INF
    if weight == W or i >= N:
        return 0
    if memo[i,weight] != -1:
        return memo[i,weight]
    T1 = f(i+1, weight + W[i]) + V[i]
    T2 = f(i+1, weight)
    if T1 > T2:
        memo[i, weight] = T1
        choice[i, weight] = (i+1, weight + W[i])
    else:
        memo[i, weight] = T2
        choice[i, weight] = (i+1, weight)
    return memo[i,weight]
```

Programación Dinámica

- Problema de edit distance:
  - PATA v GATA
  - Choice de tamaño 4x4 (AxB)

Programación Dinámica

- PATA v GATA

F(0,0) {P, G} = min(F(1,0)+1, F(1,1)+1, F(0,1)+1)

Programación Dinámica

- PATA v GATA

$F(1,0)\ \{A, G\} = \min(F(2,0)+1, F(2,1)+1, F(1,1)+1)$

Programación Dinámica

- PATA v GATA

F(2,0) {T, G} = min(F(3,0)+1, F(3,1)+1, F(2,1)+1)

Programación Dinámica

- PATA v GATA

F(3,0) {A, G} = min(F(4,0)+1, F(4,1)+1, F(3,1)+1)

Programación Dinámica

- PATA v GATA

F(4,0) {?, G} = INF

Programación Dinámica

- PATA v GATA

F(4,1) {?, A} = INF

Programación Dinámica

- PATA v GATA

F(3,1) {A, A} = F(4,2) = INF

Programación Dinámica

- PATA v GATA

F(3,0) {A, G} = INF

Programación Dinámica

- PATA v GATA

F(2,0) {T, G} = min(INF, F(3,1)+1, F(2,1)+1)

Programación Dinámica

- PATA v GATA

F(2,0) {T, G} = min(INF, INF, F(2,1)+1)

Programación Dinámica

- PATA v GATA

$$F(2,1)\ \{T, A\} = \min(F(3,1)+1,\ F(3,2)+1,\ F(2,2)+1)$$

Programación Dinámica

- PATA v GATA

F(2,1) {T, A} = min(INF, F(3,2)+1, F(2,2)+1)

Programación Dinámica

- PATA v GATA

F(3,2) {A, T} = min(F(4,2)+1, F(4,3)+1, F(3,3)+1)

Programación Dinámica

- PATA v GATA

F(3,2) {A, T} = min(INF, INF, F(3,3)+1)

Programación Dinámica

- PATA v GATA

F(3,3) {A, A} = F(4, 4) = **0**

Programación Dinámica

- PATA v GATA

			0

F(3,3) {A, A} = F(4, 4) = **0**

Programación Dinámica

- PATA v GATA

			0

F(3,2) {A, T} = min(INF, INF, 0+1)

Programación Dinámica

- PATA v GATA

		(3,3) + 1	0

F(3,2) {A, T} = min(INF, INF, 0+1)

Programación Dinámica

- PATA v GATA

		(3,3) + 1	0

F(2,1) {T, A} = min(INF, 1+1, F(2,2)+1)

Programación Dinámica

- PATA v GATA

		(3,3)+0	
		(3,3)+1	0

F(2,2) {T, T} = F(3,3) = 0

Programación Dinámica

- PATA v GATA

		(3,3)+0	
		(3,3)+1	0

F(2,1) {T, A} = min(INF, 2, 0+1)

Programación Dinámica

- PATA v GATA

	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(2,1) {T, A} = min(INF, 2, 1)

Programación Dinámica

- PATA v GATA

	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(2,0) {T, G} = min(INF, INF, 1+1)

Programación Dinámica

- PATA v GATA

F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(2,0) {T, G} = min(INF, INF, 2)

Programación Dinámica

- PATA v GATA

F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(1,0) {A, G} = min(F(2,0)+1, F(2,1)+1, F(1,1)+1)

F(1,0) {A, G} = min(2+1, 1+1, F(1,1)+1)

Programación Dinámica

- PATA v GATA

F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(1,1) {A, A} = F(2,2)+0

Programación Dinámica

- PATA v GATA

	(2,2)+0		
F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(1,1) {A, A} = F(2,2)+0

Programación Dinámica

- PATA v GATA

(1,1)+1	(2,2)+0		
F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(1,0) {A, G} = min(F(2,0)+1, F(2,1)+1, F(1,1)+1)

F(1,0) {A, G} = min(2+1, 1+1, 0+1)

Programación Dinámica

- PATA v GATA

(1,1)+1	(2,2)+0		
F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(0,0) {P, G} = min(F(1,0)+1, F(1,1)+1, F(0,1)+1)

F(0,0) {P, G} = min(1+1, 0+1, F(0, 1)+1)

Programación Dinámica

- PATA v GATA

(1,1)+1	(2,2)+0		
F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(0,1) {P, A} = min(F(1,1)+1, F(1,2)+1, F(0,2)+1)

Programación Dinámica

- PATA v GATA

(1,1)+1	(2,2)+0		
F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(0,2) {P, T} = min(F(1,2)+1, F(1,3)+1, F(0,3)+1)

Programación Dinámica

- PATA v GATA

(1,1)+1	(2,2)+0		
F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(0,2) {P, T} = min(F(1,2)+1, F(1,3)+1, F(0,3)+1)

F(0,2) {P, T} = min(INF, INF, INF)

Programación Dinámica

- PATA v GATA

(1,1)+1	(2,2)+0		
F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(0,2) {P, T} = min(F(1,2)+1, F(1,3)+1, F(0,3)+1)

F(0,2) {P, T} = min(INF, INF, INF)

Programación Dinámica

- PATA v GATA

(1,1)+1	(2,2)+0		
F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(0,1) {P, A} = min(F(1,1)+1, F(1,2)+1, F(0,2)+1)

F(0,1) {P, A} = min(0+1, INF, INF)

Programación Dinámica

- PATA v GATA

	(1,1)+1		
(1,1)+1	(2,2)+0		
F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(0,1) {P, A} = min(F(1,1)+1, F(1,2)+1, F(0,2)+1)

F(0,1) {P, A} = min(0+1, INF, INF)

Programación Dinámica

- PATA v GATA

	(1,1)+1		
(1,1)+1	(2,2)+0		
F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(0,0) {P, G} = min(F(1,0)+1, F(1,1)+1, F(0,1)+1)

F(0,0) {P, G} = min(1+1, 0+1, 1+1)

Programación Dinámica

- PATA v GATA

(1,1)+1	(1,1)+1		
(1,1)+1	(2,2)+0		
F(2,1)+2	(2,2)+1	(3,3)+0	
		(3,3)+1	0

F(0,0) {P, G} = min(F(1,0)+1, F(1,1)+1, F(0,1)+1)

F(0,0) {P, G} = min(2, 1, 2)

Práctica

- Finalmente, bastaría construir la solución a partir del primer estado conocido de la función F (en este caso, sería {0,0})
- A partir de {0,0} tenemos tres posibilidades
  - {1,0} (Agregar letra en B)
  - {0,1} (Borrar letra en B)
  - {1,1} (Reemplazar)

Práctica

- Estas transiciones solo son posibles si A[i] != B[j], por lo que si suma 1, está sucediendo una de estas opciones
- Al ir de (0,0) a (1,1)+1 estamos reemplazando
- Ir de (1,1) a (2,2)+0 no hacemos nada
- Ir de (2,2) a (3,3)+0 no hacemos nada
- Ir de (3,3) a (4,4)+0 no hacemos nada

Práctica

- Práctica siguiente (12/04):
  - Práctica online
  - Disponible hasta el 25/04
- Eliminatoria del SWERC (26/04)
  - De 16:00 ~ 20:00 (16:30 ~ 19:30)
  - Equipos de 3 y solo 3
  - 1 cupo garantizado

¡Hasta la próxima semana!

Ante cualquier duda sobre el curso o sobre los problemas podéis escribirnos (preferiblemente copia a los tres)

David Morán (ddavidmorang@gmail.com)
Juan Quintana (juandavid.quintana@urjc.es)
Sergio Pérez (sergio.perez.pelo@urjc.es)
Jesús Sánchez-Oro (jesus.sanchezoro@urjc.es)