Programación Dinamica

2da Parte

TTPS - 30/Septiembre

Programación Dinamica

Definición

- Es una tecnica de resolución de problemas, cuyo objetivo es optimizar el computo y reducir el tiempo de ejecución algoritmos, y consiste en dividir el problema en subproblemas más simples y reutilizar los resultados de estos subproblemas en lugar de recalcularlos.

- Se puede aplicar a problemas que muestran tener dos propiedades fundamentales:

    - Overlapping subproblems

    - Optimal substructure

Overlapping subproblems

- Un problema se puede descomponer en subproblemas más pequeños que se repiten varias veces durante la ejecución . En lugar de resolver estos subproblemas múltiples veces guardamos los resultados parciales

Optimal substructure

- La solución optima del problema se puede construir en base a la solución optima de sus subproblemas. Es decir, resolver cada una sus partes de manera optima asegura que el problema se resuelve de manera optima.

Programación Dinamica

Problemas

- RSQ (RANGE SUM QUERY) -> Dado un arreglo A[ ] con N números y una cantidad de consultas M imprimir la respuesta a cada una de ellas. Las consultas están compuestas por un par de números L y R, y lo que se pide es la suma de los elementos de A en el rango [L, R]

- LIS (LONGEST INCREASING SUBSEQUENCE) -> Dada una secuencia de números, se desea encontrar una subsecuencia de la misma, tal que sus elementos mantengan su orden original y sea estrictamente creciente

LCS (Longest Common Subsequence)

Longest Common Subsequence

Problema

Se tienen dos strings S y T, y debemos determinar la subsecuencia común más larga

Ejemplo:

- S = “CADSA”
- T = “CBASAS”

Resultado:

- LCS = “CASA”

Longest Common Subsequence

Problema

- Debemos verificar que disponemos de las siguientes dos propiedades para resolver utilizando Programación Dinámica:
  - Optimal Substructure
  - Overlapping Subproblems

Optimal Substructure

LCS

- Tenemos que encontrar una función recursiva que resuelva el problema.
- Podríamos pensarlo como una función LCS(n, m) a la cual llamamos con los tamaños de los strings S y T respectivamente:
    - LCS(N, 0) = 0 para todo N
    - LCS(0, N) = 0 para todo N
    - Con estos dos casos podemos armar LCS(1, 1)

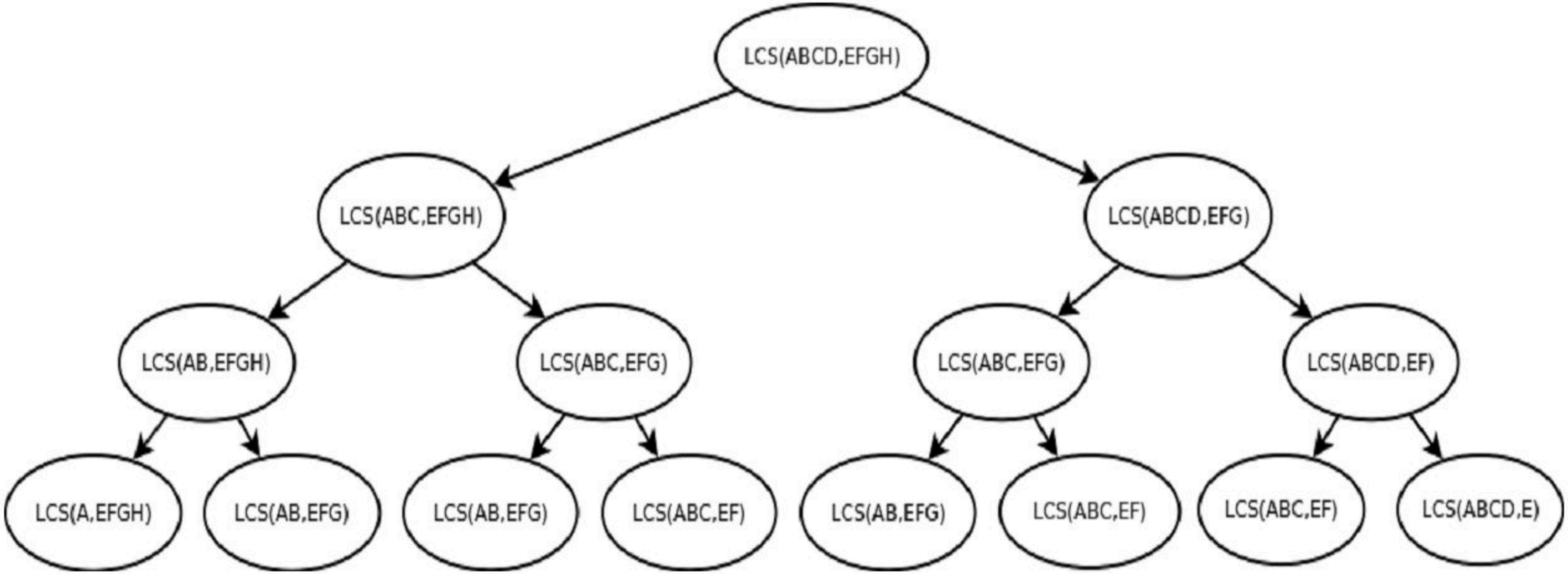
Optimal Substructure

LCS - Función Recursiva

$$
LCS(i, j) = \begin{cases} 0 & i == 0 \lor j == 0 \\ LCS(i-1, j-1) + 1 & S[i] == T[j] \\ \max(LCS(i-1, j), LCS(i, j-i)) & S[i] \mathrel{!=} T[j] \end{cases}
$$

Overlapping Subproblems

LCS



Que estructura podemos utilizar para implementar PD?

Solución LCS

Estructura

- Como la función dependerá de dos parámetros numéricos, la estructura que podemos usar es una matriz de dos dimensiones:

	NULL	C	A	R
NULL				
R				
O				
C				
A				

Solución LCS

Inicialización

- Completamos con los valores que ya conocemos:

	NULL	C	A	R
NULL	0	0	0	0
R	0			
O	0			
C	0			
A	0			

LCS entre
S = “CAR” y
T = “ ” es 0

Solución LCS

Proceso

	NULL	C	A	R
NULL	0	0	0	0
R	0			
O	0			
C	0			
A	0			

LCS entre S=”C” y T=”R”

Como “C” y “R” son diferentes se toma el máximo entre el LCS(“C”, NULL) (celda de arriba) y LCS(NULL, “R”) (celda izquierda)

Si hubiesen sido iguales se hubiese sumado uno al caso LCS(NULL, NULL) (celda diagonal superior izquierda) ya que es el mismo caso +1 (carácter coincidente)

Solución LCS

Resultado

	NULL	C	A	R
NULL	0	0	0	0
R	0	0	0	1
O	0	0	0	1
C	0	1	1	1
A	0	1	2	2

→ LCS("CAR", "ROCA")

Como podria obtener el str coincidente?

Solución LCS

Codigo

```
int LCS(string s, string t) {

  int n = s.length() + 1,
  int m = t.length() + 1
  int memo[n][m];

  // Inicialización
  for(int i = 0; i < n; i++) memo[i][0] = 0;
  for(int j = 0; j < m; j++) memo[0][j] = 0;

  for(int i = 1; i < n; i++) {
    for(int j = 1; j < m; j++) {
      if(s[i-1] == t[j-1]) {
        memo[i][j] = memo[i-1][j-1] + 1;
      } else {
        memo[i][j] = max(memo[i-1][j], memo[i][j-1]);
      }
    }
  }

  return memo[n-1][m-1];
}
```

ED (Edit Distance)

Edit Distance

Problema

Se tienen dos strings S y T. Debemos determinar la mínima cantidad de operaciones necesarias para transformar el primer string en el segundo. Las operaciones pueden tener costo y son: reemplazo, inserción, y eliminación

Costos:

- Inserción: 3
- Reemplazo: 4
- Delete: 2

Ejemplo:

- S = “CADSA”
- T = “CBASAS”

Resultado:

- ED = 6
- Insertar B en la posición 1 y S al final

Edit Distance

Problema

- Debemos verificar que disponemos de las siguientes dos propiedades para resolver utilizando Programación Dinámica:
  - Optimal Substructure
  - Overlapping Subproblems

Optimal Substructure

ED

- Tenemos que encontrar una función recursiva que resuelva el problema.
- Podríamos pensarlo como una función ED(n, m) a la cual llamamos con los tamaños de los strings S y T respectivamente.
  - ED(0, N) -> No quedan más letras del primer string, solo nos queda insertar las N restantes del segundo.
  - ED(N, 0) -> No quedan más letras del segundo string, solo nos queda eliminar las N sobrantes del primero.
  - ED(x, y) -> hay letras, tenemos que verificar cual es la operación más conveniente.

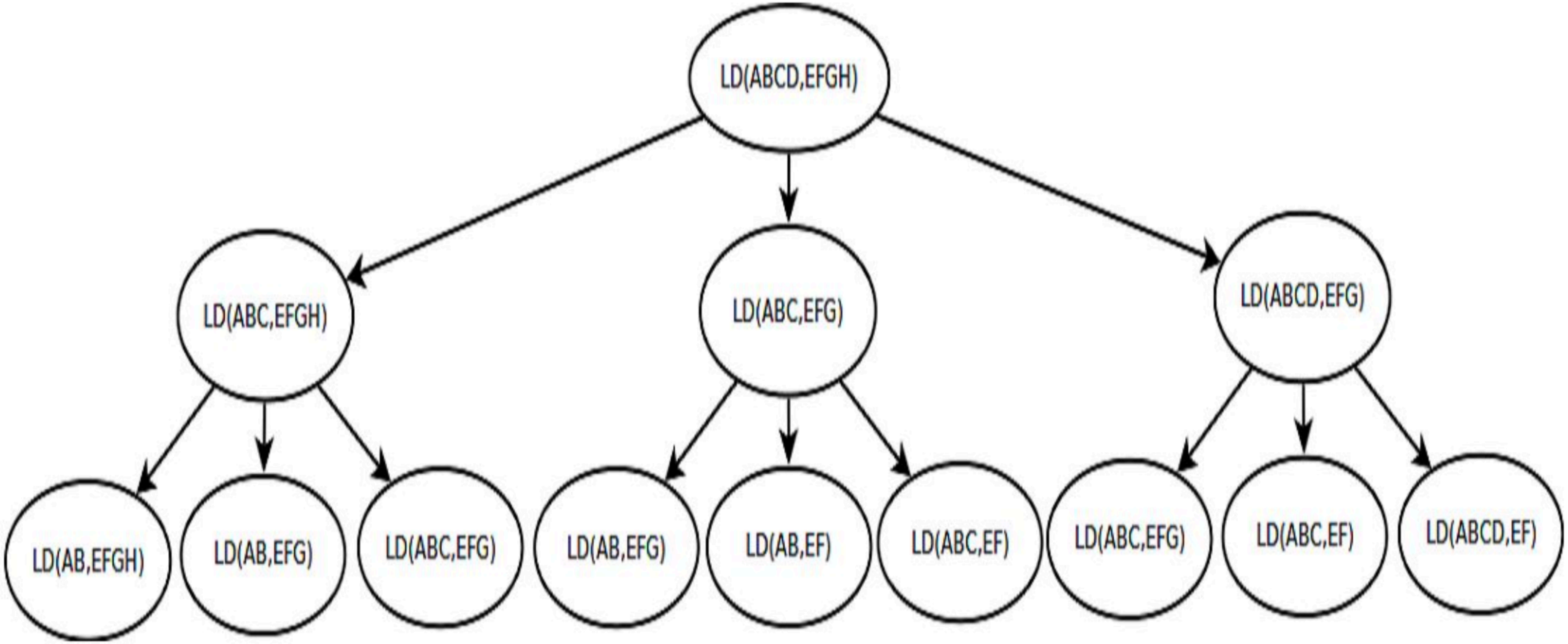
Optimal Substructure

ED - Función Recursiva

$$
ED(i, j) = \begin{cases}
i * cost\_insert & j == 0 \\
j * cost\_delete & i == 0 \\
\min(\, ED(i-1, j) + cost\_delete, & i > 0 \;\&\; j > 0 \\
\quad ED(i, j-1) + cost\_insert, & \\
\quad ED(i-1, j-1) + ((S[i] == T[j])\ ?\ 0 : cost\_replace) &
\end{cases}
$$

Overlapping Subproblems

ED



Que estructura podemos utilizar para implementar PD?

Solución ED

Estructura

- Como la función dependerá de dos parámetros numéricos, la estructura que podemos usar es una matriz de dos dimensiones:

	NULL	C	A	R
NULL				
R				
O				
C				
A				

Solución ED

Inicialización

- Completamos con los valores que ya conocemos:

	NULL	C	A	R
NULL	0	3	6	9
R	2			
O	4			
C	6			
A	8			

ED entre S = “ ” y T = “CAR” es 9 (3 inserciones)

ED entre S = “ROCA” y T = “ ” es 8 (4 borrados)

Solución ED

Proceso

	NULL	C	A	R
NULL	0	3	6	9
R	2			
O	4			
C	6			
A	8			

ED entre S="R" y T="C"

Se toma el mínimo entre:

- ED(NULL, "C") + DELETE
  = 3 + 2 = 5
- ED("R", NULL) + INSERT
  = 2 + 3 = 5
- ED(NULL, NULL) + REPLACE
  (sii "R" y "C" no son iguales)
  = 0 + 4 = 4

Que coloco en este caso?

Solución ED

Proceso

	NULL	C	A	R
NULL	0	3	6	9
R	2	4	7	6
O	4	6	8	8
C	6	4	7	10
A	8	6	4	7

Las flechas indican de qué celda se tomó el mínimo valor hasta llegar al 10. Hay dos caminos de flechas porque existen dos mínimos.

ED ("ROC", "CAR") = 10

- DELETE RO + INSERT AR
- INSERT CA + DELETE OC

Solución ED

Resultado

	NULL	C	A	R
NULL	0	3	6	9
R	2	4	7	6
O	4	6	8	8
C	6	4	7	10
A	8	6	4	7

→ ED("ROCA", "CAR")

Como podria obtener el conjunto de operaciones realizadas?

Solución ED

Codigo

```
int ED(string s, string t, int cost_delete, int cost_insert, int cost_replace) {

  int n = s.length() + 1;
  int m = t.length() + 1,
  int memo[n][m]

  // Inicialización
  for (i = 0; i < n; i++) memo[i][0] = i * cost_delete;
  for (j = 0; j < m; j++) memo[0][j] = j * cost_insert;

  for (i = 1; i < n; i++){
    for (j = 1; j < m; j++){
      if (S[i-1] == T[j-1]) {
        memo[i][j] = matrix[i-1][j-1]
      } else {
        memo[i][j] = matrix[i-1][j-1] + cost_replace;
        memo[i][j] = min(memo[i][j], memo[i-1][j] + cost_delete);
        memo[i][j] = min(memo[i][j], memo[i][j-1] + cost_insert);
       }
    }
  }
  return memo[n-1][m-1];
}
```