

Preparación Específica para Concursos ACM-ICPC.

Tema #5 : Programación Dinámica.

Conferencia #9: Introducción a la programación dinámica. Algoritmos Coin change, Edit distance, Subsecuencia de longitud máxima común.

Objetivos

- ❑ Caracterizar los principales elementos que distinguen a la programación dinámica.

Objetivos

- Caracterizar los algoritmos *coin change*, *edit distance* y *subsecuencia de longitud máxima común*.

Objetivos

- ❑ Identificar los problemas que se pueden solucionar aplicando los algoritmos *coin change*, *edit distance* y *subsecuencia de longitud máxima común*.

Contenidos

- ❑ Programación Dinámica.
- ❑ Algoritmo *coin change*.
- ❑ Algoritmo *edit distance*.
- ❑ Algoritmo *subsecuencia de longitud máxima común*.

Bibliografía

- *Manual de preparación para concursantes ACM-ICPC de la Universidad de Matanzas.*

Programación Dinámica

- La programación dinámica es un método para reducir el tiempo de ejecución de un algoritmo mediante la utilización de subproblemas superpuestos y subestructuras óptimas.

Algoritmo *coin change*

- La primera variante del *coin change* no servirá para determinar de cuantas formas podemos devolver un valor N usando una serie de K denominaciones de billetes.

Algoritmo *coin change*

- En otras palabras de cuanta formas podemos devolver 13 pesos teniendo una cantidad infinita de billetes con los valores de 1, 3, 5, 10, 15.

Algoritmo *coin change*

- Esta estructura lineal será un arreglo que denominaremos *way*.

Algoritmo *coin change*

- Inicialmente este arreglo será rellenado con cero cada posición , excepto la posición 0 la cual su valor será 1.

Algoritmo *coin change*

- Una vez planteado solo queda recorrer cada posición para cada denominación de billetes y plantear los siguiente $way[j+monedas[i]] += way[j]$ siempre y cuando $j+monedas[i]$ sea menor que N máximo.

Algoritmo *coin change*

- De esta forma la cantidad de forma de devolver un valor N con k denominaciones de billetes va estar en $way[N]$.

Algoritmo *coin change*

- La otra variante de esta técnica radica en determinar la mínima cantidad de billetes necesarios para devolver una cantidad N teniendo infinita cantidad de billetes de k denominaciones.

Algoritmo *coin change*

- De similar manera tendremos un arreglo que llenaremos inicialmente con valor bien grande. Luego iremos a cada posición que represente una de las denominaciones de los billetes disponibles y su valor será 1.

Algoritmo *coin change*

- Luego para cada posición que se cumpla que $\text{way}[i + \text{denominacion}[j]] > \text{way}[i] + 1$ actualizo $\text{way}[i + \text{denominacion}[j]] = \text{way}[i] + 1$.

Algoritmo *edit distance*

- Sean u y v dos cadenas de caracteres. Se desea transformar u en v con el mínimo número de operaciones básicas del tipo siguiente: eliminar un carácter, añadir un carácter, y cambiar un carácter.

Algoritmo *edit distance*

- Llamaremos m a la longitud de la cadena u , n a la longitud de la cadena v , y $OB(m,n)$ indicará el número de operaciones básicas mínimo para transformar una cadena u de longitud m en otra cadena v de longitud n .

Algoritmo *edit distance*

- Vamos a fijarnos en el último elemento de cada una de las cadenas. Si los dos son iguales, entonces tendremos que calcular el número de operaciones básicas necesarias para obtener de la primera cadena menos el último elemento, y la segunda cadena también sin el último elemento.

Algoritmo *edit distance*

□ $OB(m,n) = OB(m-1,n-1)$ si $u_m = v_n$

Algoritmo *edit distance*

- ❑ Pero si los últimos elementos fueran distintos habría que escoger la situación más beneficiosa de entre tres posibles:
- ❑ Considerar la primera cadena y la segunda pero sin el último elemento.

Algoritmo *edit distance*

- ❑ Pero si los últimos elementos fueran distintos habría que escoger la situación más beneficiosa de entre tres posibles:
- ❑ La primera cadena menos el último elemento y la segunda cadena.

Algoritmo *edit distance*

- ❑ Pero si los últimos elementos fueran distintos habría que escoger la situación más beneficiosa de entre tres posibles:
- ❑ Las dos cadenas sin el último elemento.

Algoritmo *edit distance*

$$OB(m,n)=1+\text{Min}\{OB(m,n-1),OB(m-1,n),OB(m-1,n-1)\}$$

Donde:

$$m \neq 0, n \neq 0 \text{ y } u_m \neq v_n$$

En cuanto a las condiciones iniciales, tenemos las tres siguientes:

$$OB(0,0) = 0, OB(m,0) = m \text{ y } OB(0,n) = n$$

Algoritmo *Subsecuencia de longitud máxima común*

- Dada una secuencia $X = \{x_1, x_2, \dots, x_m\}$, decimos que $Z = \{z_1, z_2, \dots, z_k\}$ es una subsecuencia de X (siendo $k \leq m$) si existe una secuencia creciente $\{i_1, i_2, \dots, i_k\}$ de índices de X tales que para todo $j = 1, 2, \dots, k$ tenemos $x_{i_j} = z_j$.

Algoritmo *Subsecuencia de longitud máxima común*

- Dadas dos secuencias X e Y , decimos que Z es una subsecuencia común de X e Y si es subsecuencia de X y subsecuencia de Y . Deseamos determinar la subsecuencia de longitud máxima común a dos secuencias.

Algoritmo *Subsecuencia de longitud máxima común*

- Llamaremos $L(i,j)$ a la longitud de la secuencia común máxima (LCS) de las secuencias X_i e Y_j , siendo X_i el i -ésimo prefijo de X (esto es, $X_i = \{x_1 x_2 \dots x_i\}$) e Y_j el j -ésimo prefijo de Y , ($Y_j = \{y_1 y_2 \dots y_j\}$).

Algoritmo *Subsecuencia de longitud máxima común*

$$L(i, j) = \begin{cases} 0 & \text{si } i = 0 \text{ o } j = 0 \\ L(i - 1, j - 1) + 1 & \text{si } i \neq 0, j \neq 0 \text{ y } x_i = y_j \\ \text{Max}\{L(i, j - 1), L(i - 1, j)\} & \text{si } i \neq 0, j \neq 0 \text{ y } x_i \neq y_j \end{cases}$$

Algoritmo *Subsecuencia de longitud máxima común*

- La solución recursiva resulta ser de orden exponencial, y por tanto Programación Dinámica va a construir una tabla con los valores $L(i, j)$ para evitar la repetición de cálculos.

Conclusiones.

- Para *coin change* la complejidad de ambos algoritmos es de $O(N * K)$ donde N es el valor máximo y K el número de denominaciones diferentes de billetes disponibles tener en cuenta.

Conclusiones.

- Como el algoritmo Edit distance se limita a dos bucles anidados que sólo incluyen operaciones constantes la complejidad de este algoritmo es de orden $O(mn)$.

Estudio Independiente

- Profundizar en los temas abordados con la lectura del capítulo Programación Dinámica del manual mencionado en la bibliografía del curso.

Estudio Independiente

- ❑ Solucionar de Juez Caribeño Online COJ los siguientes problemas.
- ❑ 1478 – Basic Edit Distance
- ❑ 3820 – Returning Coins
- ❑ 1103 – Coin Change
- ❑ 2616 – Easy Change
- ❑ 3105 – Filling the Hole

Preparación Específica para Concursos ACM-ICPC.

Tema #5 : Programación Dinámica.

Conferencia #9: Introducción a la programación dinámica. Algoritmos Coin change, Edit distance, Subsecuencia de longitud máxima común.