Programación Dinámica

Problema 1: Cálculo de funciones

Implementa un algoritmo Basado en Programación dinámica que calcule la siguiente función:

$$f(n) = \begin{cases} 0 & si\ n = 0 \\ n + \frac{2}{n}\sum_{i=0}^{n-1} f(i) & si\ n > 0 \end{cases}$$

Problema 2: Cálculo de funciones

Implementa un algoritmo Basado en Programación dinámica que calcule la siguiente función:

$$f(n,m) = \begin{cases} 0 & si\ m = 0 \\ n & si\ m = 1 \\ f\left(n, \left\lfloor \frac{m}{2} \right\rfloor\right) + f\left(n, \left\lceil \frac{m}{2} \right\rceil\right) & si\ m > 1 \end{cases}$$

Problema 3: Cálculo de funciones

Implementa un algoritmo basado en PD que, dados dos valores enteros m y n, y un vector V que como mínimo tenga m+1 elementos, calcule el valor de Q(m, n) según la siguiente función:

$$Q(m,n) = \begin{cases} 1 & si\ m = 0 \\ 1 & si\ n = 0 \\ Q(m-1,n) & si\ n < v_m \\ \sum_{i=1}^{v_m - 1} Q(m, n-i) & e.o.c. \end{cases}$$

Problema 4: Cambio de Monedas

Vivimos en un país donde el sistema monetario está formado por una serie de monedas, cada una de ellas con un valor facial.

$$M=\{m_1, m_2, \ldots, m_n\}$$

Diseñar un algoritmo que permita devolver una cierta cantidad de dinero C utilizando el menor número de monedas posible.

Guía: Para resolver este problema podemos considerar su similitud con el problema de la mochila. Así, si $x_1, x_2, \ldots, x_{n-1}, x_n$ representa las monedas en un cambio óptimo para una cantidad C, entonces $x_1, x_2, \ldots, x_{n-1}$ debe ser también un cambio óptimo para la cantidad C-Valor_Facial (x_n) y de forma obvia x_n es un cambio óptimo para la cantidad Valor_Facial (x_n).

Definir una función: *monedas (n, C)* que represente el mínimo de monedas necesarias para pagar la cantidad C considerando los tipos de monedas del 1 al n.

A partir se plantea la recurrencia en el caso general, cuando se consideran los tipos de monedas del 1 al i y nos queda pagar una cantidad j: *monedas(i,j)*. (Utilizar un enfoque hacia atrás). Una vez hecho esto habrá que determinar cuáles son los casos base (soluciones triviales) y hallar sus respectivos valores.

Problema 5: El problema del Botellón

Dos amigos, *Agonioso* y *Listillo*, salen de botellón. La nueva moda es poner una fila con n vasos (n es par). Cada vaso i, entre 1 y n contiene una cantidad de líquido c_i distinta. Los amigos beben por turnos. Cada uno, en su turno debe elegir el vaso de uno de los extremos y beberse su contenido. El vaso se retira y el turno pasa al otro amigo. La persona que comienza bebiendo se determina a priori por un procedimiento cualquiera. El objetivo de ambos amigos es beber la mayor cantidad posible de líquido.

La estrategia de *Agonioso* consiste en pensar poco y coger el vaso de los extremos que esté más lleno. En cambio *Listillo* prefiere pensárselo un poco más.

a) Demostrar, con un contraejemplo que la estrategia de *Agonioso* no es óptima, incluso cuando le toca escoger primero.
b) *Listillo* tiene unos amigos que cursan la asignatura de Diseño de Algoritmos y les pide que le diseñen un algoritmo, basado en Programación Dinámica, para que le ayude a conseguir su objetivo, suponiendo que es él quien empieza a escoger.

Guía: Definir una función *botellón(i,j)* que represente la cantidad máxima que bebe *Listillo* con los vasos desde el i hasta el j cuando le toca empezar a beber. La solución final será *botellón(1,n)*

Problema 6: Otra versión más justa del botellón

Esta vez Agonioso y Listillo van a la tienda y compran n vasos ya preparados con su bebida favorita. Cada uno de los vasos *i* tiene una cantidad de bebida c_i para todo *i entre 1 y n*.

Para no beber uno más cantidad que el otro (el líquido de los vasos no se pueden repartir) quieren saber si se pueden elegir los vasos para distribuir la cantidad de líquido a medias y que ambos beban lo mismo.

Diseñar un algoritmo basado en Programación Dinámica que les ayude a averiguar la respuesta.

NOTAS:

- Considerar C= $\sum_{i=1}^{n} Ci$. Por tanto, el problema es equivalente a comprobar si es posible, sumando las cantidades de algunos vasos, obtener C/2. Suponemos que C es par, porque si no, no hay solución.

- La solución se puede representar mediante la siguiente función:
  se_puede(i,j)= verdad*ero si es posible sumar la cantidad "j" eligiendo algunos de los "i" primeros vasos y falso en caso contrario.*

- La solución final es *se_puede(n,C/2)*

- Utilizar un enfoque hacia atrás comparando la cantidad del vaso i (c_i) con la cantidad j que queremos conseguir.

Problema 7: Los puentes que unen un grupo de islas

Un archipiélago está formado por varias islas. Existen una serie de puentes de dirección única que unen ciertos pares de islas entre sí. Para cada puente se conoce su anchura, que siempre es mayor que 0.

La *anchura de un camino* formado por una sucesión de puentes es la anchura mínima entre las anchuras de todos los puentes que lo forman.

Diseñar un algoritmo basado en Programación Dinámica para saber, si existe, cuál es el camino de anchura máxima entre todo par de islas.

Guía: Para resolverlo pensad en una estrategia parecida a la del Algoritmo de Floyd.

Problema 8: Otra variante del problema de la mochila

Diseñar un algoritmo basado en Programación Dinámica para resolver el problema de la mochila en la que hay *n* tipos de objetos y más de un objeto de cada tipo.

Recordad que para todo *i=1, 2, …, n*; un objeto de tipo *i* tiene un peso positivo p_i y una ganancia positiva g_i. *La* capacidad máxima de la mochila es M. El objetivo es llenar la mochila de manera que se maximice la ganancia de los objetos incluidos, respetando la restricción de capacidad. Solamente se pueden incluir objetos enteros.

NOTAS:

- La solución se puede representar mediante la siguiente función:
  *ganancia2(1,*i*,j)* = ganancia máxima obtenida al considerar los *i* primeros tipos de objetos para una mochila de capacidad *j*.
- La solución final será: *ganancia2(1,*n*,M)*.
- Hay que tener en cuenta que cuando se coge un objeto, no se agota. Se puede seguir considerando ese tipo de objeto.

Problema 9: Optimización del algoritmos de Coeficientes Binomiales

Optimizar el algoritmo del cálculo de los coeficientes binomiales usando solo las celdas de la matriz que realmente son necesarias, es decir, que se reserve solamente la memoria necesaria para hacer el cálculo.

$$\binom{i}{j} = \begin{cases} 1 & si \quad j = 0 \\ 1 & si \quad i = j \\ \binom{i-1}{j} + \binom{i-1}{j-1} & e.o.c. \end{cases}$$

Problema 10: Las rutas en barca por el río

En un río hay ***n*** embarcaderos. En cada uno de ellos llamémosle ***i***, se puede alquilar un bote para ir a cualquier otro, río abajo, llamémosle ***j***, con un coste determinado.

Así, para ir del embarcadero ***i*** al ***j*** el precio es de T[i][j]. Puede suceder que para ir de ***i*** a ***j*** sea más barato hacer una o más escalas intermedias, es decir, podría suceder que t[i][j]>t[i][k]+t[k][j] siendo i<k<=j.

Sabiendo que el río no se puede remontar, diseña un algoritmo basado en **Programación Dinámica** que calcule el precio más barato para ir desde ***i*** hasta ***j***.

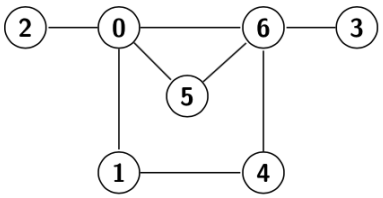
Problema 11: El traductor de textos

Se desea hacer traducciones de textos entre varios idiomas. Se dispone de varios traductores automáticos que permiten la traducción bidireccional entre dos idiomas. No se dispone de diccionarios para cada par de idiomas por lo que es preciso encadenar varias traducciones. Dados N idiomas y M traductores, implementa un algoritmo basado en Programación Dinámica que devuelva la cadena de traducciones de longitud mínima entre dos idiomas dados, si es posible.

Ejemplo: Traducir del español al griego disponiendo de los siguientes diccionarios: español-inglés, inglés-francés, francés-italiano, italiano-griego, francés-griego. La **solución** sería: español-inglés, inglés-francés, francés-griego, realizando 3 traducciones.

Problema 12: Número mínimo de nodos entre cada par de nodos de un grafo

Sea G=(N,A) un grafo no dirigido, siendo n el número de nodos y L la matriz de adyacencia con valores booleanos (1 si están conectados, 0 si no). Diseña un algoritmo basado en la técnica de la Programación Dinámica que calcule, para todo par de nodos i,j ∈ N, el camino más corto entre ellos, siendo la distancia el número de nodos por los que pasa. En el siguiente grafo la distancia mínima entre el nodo 2 y el 4 es de 2 (2-[0-1]-4).



Nota: La solución se puede representar de la siguiente forma:

D[i,j,p]= distancia mínima de i a j cuando se pasa, a lo sumo, por p nodos.