Algorítmica

Capítulo 4: Programación Dinámica

Ejercicios prácticos

Fecha de entrega: Antes de las 23 horas y 59 minutos del día 15 de mayo

Objetivo de las prácticas

En este caso lo que se persigue con estos ejercicios no es tanto que aprendan a diseñar algoritmos basados en este enfoque, como:

1. Reforzar el conocimiento sobre los algoritmos de PD mediante la comprensión y repetición de estos ejercicios (casi) completamente resueltos.
2. Que comprueben, cuando sea necesaria, la verificación del P.O.B.
3. Que comprendan los algoritmos, los implementen y los apliquen sobre algún caso sencillo.

Entréguenme solo dos ficheros: Uno (MEM Nombre Equipo) con la memoria de su trabajo (soluciones, caso práctico …) y otro (Presen Nombre Equipo) con las presentaciones que harán en clase.

1. Producto máximo con suma fija

- Dado M, un número natural no negativo, ¿cuál es el mayor valor que podemos conseguir multiplicando n números naturales que sumen M?
- Es decir, deseamos

$$\begin{aligned} maximizar \quad & x_1 \times \ldots \times x_n \\ sujeto\ a \quad & x_1 + x_2 + \ldots + x_n = M \end{aligned}$$

- Para aplicar PD consideramos la siguiente función

  MAX(k,C) = Máximo valor de $x_1 \times x_2 \times \ldots \times x_k$
  sujeto a $x_1 + x_2 + \ldots + x_k = C$, siendo $x_1, x_2, \ldots, x_k$ números naturales

- de la que necesitamos el valor MAX(n, M).
- Definir esa función de forma recurrente para aplicar convenientemente la técnica.
- Escribir el algoritmo que calcule los valores de esa función siguiendo la recurrencia definída en el punto anterior.
- Analizar la eficiencia del algoritmo

1. Producto máximo con suma fija

- La función MAX(k,C) = Máximo valor de $x_1 \times x_2 \times \ldots \times x_k$ sujeto a $x_1 + x_2 + \ldots + x_k = C$, siendo $x_1, x_2, \ldots, x_k$ números naturales, puede definirse mediante las siguientes ecuaciones,

  $MAX(k, 0) = 0 \ \forall k \geq 0$

  $MAX(1, C) = C$ si $C > 0$

  $MAX(k, C) = \text{Max}\ \{x \times MAX(k-1, C-x) \mid 1 \leq x \leq C\}$, si $k>1 \wedge C>0$

- Un algoritmo que calcule los valores de una tabla T (0..n, 0..M), de manera que T(k, C) = MAX(k, C).

```
func MAX(n, M) return natural
  for k in 0..n loop T(k,0) ← 0
  for C in 1..M loop T(1,C) ← C
  for k in 2..n loop
    for C in 1..M loop
      T(k,C) ← 0
      for x in 1..C loop
        T(k,C) ← máximo{T(k,C), x × T(k-1,C-x)}
  return T(n,M)
```

- Cuya eficiencia es O(nM),

2. La conexión de "El Pedregal"

- En un archipiélago, con multitud de pequeñas islas cercanas, hay puentes que unen ciertos pares de islas entre sí. Para cada puente (que puede ser de dirección única), además de saber la isla de origen y la isla de destino, se conoce su anchura (número entero mayor que 0).
- La anchura de un camino, formado por una sucesión de puentes, es la anchura mínima de las anchuras de todos los puentes que lo forman.
- Para cada par de islas se desea saber cuál es el camino de anchura máxima que las une (siempre que exista alguno).
- Diseñe un algoritmo con la técnica de Programación Dinámica que resuelva este problema. Analice su algoritmo.

2. La conexión de "El Pedregal"

- Denominaremos A(i, j) la anchura del puente que va de la isla i a la isla j. Si no hay puente en esa dirección entonces A(i, j) = 0. Obsérvese que conviene A(i, i) = ∞ puesto que para ir de una isla a ella misma no debe existir ninguna restricción.
- Definimos, de manera recursiva, la siguiente función:

  Max Anchura(i, j, k) = máxima anchura de los caminos que van de la isla i a la isla j, pudiendo pasar por las islas {1,…,k}

  Max Anchura(i, j, 0) = A(i, j)
  Max Anchura(i, j, k) = Max{Max Anchura(i, j, k – 1),
  Min{Max Anchura(i, k, k – 1),
  Max Anchura(k, j, k – 1)}} si k>0

- Así podemos diseñar un algoritmo como el siguiente, que calcula los valores de una tabla M(1..n, 1..n), de manera que al terminar:

  M(i, j) = Max Anchura(i, j, n) ∀i, j

2. La conexión de "El Pedregal": Algoritmo

```
proc MÁXIMA_ANCHURA(A, M)
  M(1..n, 1..n) ← A(1..n, 1..n)
  for k in 1..n loop
    for i in 1..n loop
      for j in 1..n loop
        if M(i,k) < M(k,j) then aux ← M(i,k)
        else aux ← M(k,j)
        if M(i,j) < aux then M(i,j) ← aux
```

- El algoritmo es fácil ver que es $O(n^3)$

3. Regalos por la fama

- Mª Gabriela y Eduardo Fernando han recibido un montón de regalos por su estupendo trabajo en una serie de televisión de reconocida fama. Cada regalo viene en una caja destinada a ambos. Como no tienen suficiente tiempo para desempaquetar y mirar que es cada cosa, han decidido utilizar el siguiente criterio para repartirse los regalos:
- Cada uno debe quedarse con la misma cantidad de peso
- Para ello cuentan con los pesos de cada una de las cajas P_1,..., P_n (números enteros positivos).
- Al cabo de un buen rato, todavía no han conseguido hacer el reparto según ese criterio.
- Diseñar un algoritmo, utilizando la técnica de Programación Dinámica, que resuelva el problema de esta pareja y determine una forma de reparto de los regalos, si es que es posible.
- Analizar el algoritmo.

3. Regalos por la fama

- Sea $D = P_1 + \ldots + P_n$ la suma de los pesos. Asumimos que D es par, puesto que en otro caso no es posible ese reparto paritario. El problema consiste en determinar si existe un subconjunto de las cajas tal que la suma de sus pesos sea D .
- Definimos la función booleana siguiente para aplicar PD,

  $S(m, k)$ = True si existe un subconjunto de $\{P_1, \ldots, P_k\}$ que sume m. En caso contrario
  $S(m, k)$ = False

  $S(0, k)$ = True $\forall k \geq 0$
  $S(m, 0)$ = False si $m > 0$
  $S(m, k) = S(m, k-1) \vee S(m - P_k, k-1)$ si $m \geq P_k$ y $k > 0$
  $S(m, k) = S(m, k-1)$ si $m < P_k$ y $k > 0$

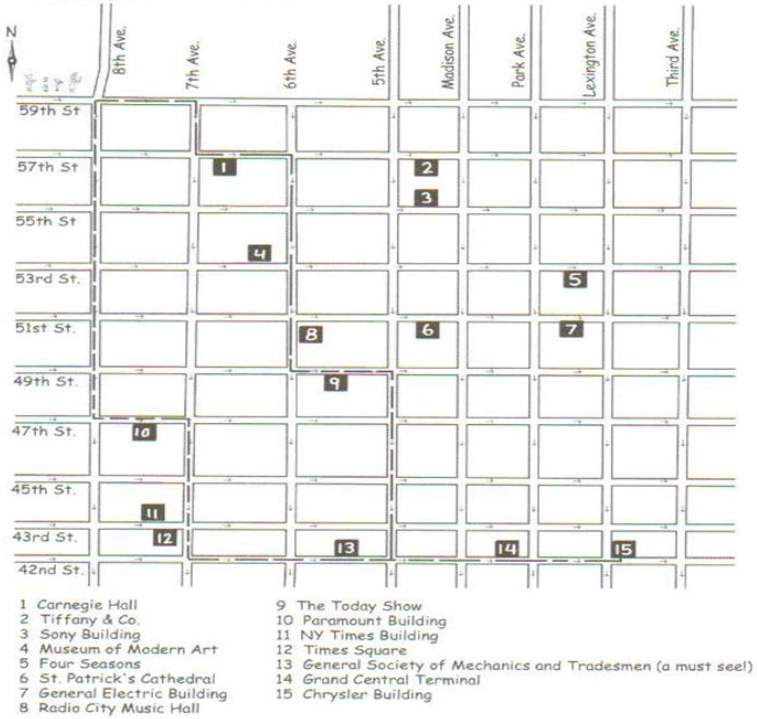
- Necesitamos el valor $S(D/2, n)$. El siguiente algoritmo calcula ese valor con ayuda de una tabla con el mismo nombre S. Además, el algoritmo incluye unas sentencias que dejan las marcas pertinentes para poder construir después la colección de regalos de uno de ellos; el otro tomaría el resto de los regalos.

3. Regalos por la fama: Algoritmo

```
func HAY_REPARTO(P(1..n), S(0..D/2, 0..n)) return (Boolean × Conjunto)
  for k in 0..n loop S(0,k) ← True
  for m in 1..D/2 loop S(m,0) ← False
  for k in 1..n loop
    for m in 1..D/2 loop
      if m < P(k) then
        S(m,k) ← S(m,k-1)
        marca(m,k) ← False
      elsif S(m-P(k), k-1) then
        S(m,k) ← S(m-P(k), k-1)
        marca(m,k) ← True
      else
        S(m,k) ← S(m, k-1)
        marca(m,k) ← False
  end loop
  {Ahora calculamos la colección de regalos}
  if S(D/2, n) then
    i ← D/2
    j ← n
    C ← ∅
    while j > 0 loop
      if marca(i,j) then
          C ← C ∪ {j}
          i ← i - P(j)
          j ← j - 1
        else
          j ← j - 1
    else
      return (False, ∅)
```

El algoritmo es O(nD)

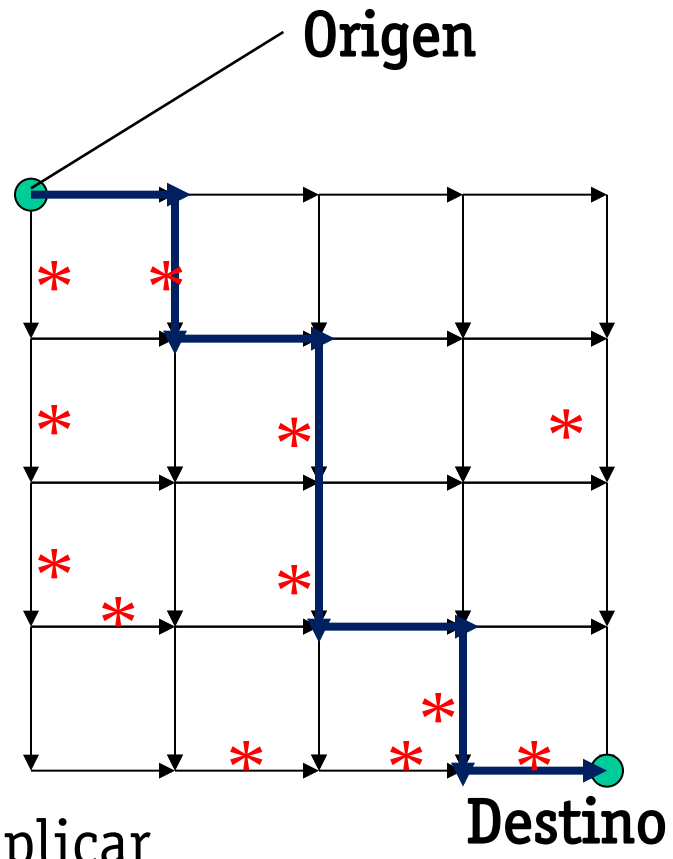
4. El Problema del Turista en Manhattan*



* An introduction to bioinformatics algorithms. N. C. Jones and P. Pevzner, 2004

4. El Problema del Turista en Manhattan

Se trata de encontrar un camino (desde un origen a un destino) tal que caminando solo hacia el sur y hacia el este, nos permita visitar el mayor número lugares turísticos (*) en el mapa de Manhattan.



- El problema, como el del Play-Off, no verifica el POB, pero igual que aquel es n-etápico y se le puede aplicar el enfoque matricial que sugiere la PD.
- Desarrollar un algoritmo que resuelva el problema. Calcular su eficiencia. Aplicarlo a un caso práctico