Ejercicio 1: Producto máximo con suma fija.

Para resolver un problema con Programación Dinámica, necesitamos que éste tenga naturaleza n-etápica, que verifique el Principio de Optimalidad de Bellman. Además, se debe poder plantear una ecuación de recurrencia que represente la forma de ir logrando etapa por etapa la solución optimal, hasta encontrar la solución final.

Descripción

Dado *C*, un número no negativo, ¿cuál es el mayor valor que podemos obtener multiplicando *n* números naturales que sumen *C*?

Más técnicamente, buscamos:

Maximizar $x_1 \times x_2 \times \ldots \times x_k$

Sujeto a $x_1 + x_2 + \ldots + x_k = C.$

Vamos a desarrollar las características necesarias para resolver el problema por Programación Dinámica:

Naturaleza n-etápica

Sabemos que es un problema n-etápico, ya que tenemos:

- $MAX(k, C)$ como el máximo producto alcanzable.
- x como el valor de la primera subdivisión, es decir, del número que, al sumarse con las $(n-1)$ partes restantes, resulte C
- $(C - x)$ como el valor de la suma de las $(n-1)$ partes restantes

Verificación del POB

Vamos a verificar el POB mediante reducción al absurdo.

Sea $x_1, x_2, x_3, \ldots x_k$ enteros que suman C y cuyo producto es máximo (solución óptima).

Se ha obtenido x_1, entero óptimo, por lo que $x_2, x_3, \ldots x_k$ debe ser un subconjunto óptimo. De no ser óptimo:

- Sea $y_2, y_3, \ldots y_k$ una solución óptima, y por lo tanto, mejor que $x_2, x_3, \ldots x_k$
- Entonces $y_2 \times y_3 \times \ldots \times y_k$ > $x_2 \times x_3 \times \ldots \times x_k$
- Por lo tanto, $x_1 \times y_2 \times y_3 \times \ldots \times y_k$ > $x_1 \times x_2 \times x_3 \times \ldots \times x_k$

Esto es una contradicción, puesto que hemos partido de la hipótesis de que $x_1, x_2, x_3, \ldots x_k$ es solución óptima.

Por lo tanto, el POB se puede verificar y aplicar a este problema.

Solución

La ecuación que describe el problema es:

$$MAX(k,C) = Máximo_{1 \leq x \leq C}\{xMAX(k-1, C-x)\}, \text{ siendo } k \geq 1 \text{y } C > 0$$

Vamos a demostrar que esto es cierto (y calcular la x máxima entre 1 y C) mediante inducción.

Primero, sabemos que:

- $MAX(k,0) = 0 \quad \forall k \geq 0$
- $MAX(1,C) = C \quad si\ C > 0$ (caso más pequeño, k=1)

Vamos a resolver, como caso base, para k=2:

$$MAX(2,C) = Máximo_{1\leq x\leq C}\{xMAX(1,C-x)\}$$

Como sabemos que $MAX(1,C) = C$:

$$MAX(2,C) = Máximo_{1\leq x\leq C}\{x(C-x)\}$$

Vamos a sacar ese máximo derivando la función $f(x) = x(C-x)$ e igualándola a 0.

$$f'(x) = C - 2x$$
$$C - 2x = 0, \text{ por lo que } x = \frac{C}{2}$$

Sustituyendo, $MAX(2,C) = \frac{C}{2}MAX(1,\frac{C}{2})$, es decir,

$$MAX(2,C) = \frac{C}{2}\cdot\frac{C}{2} =$$

$$(\frac{C}{2})^2, \text{ con soluciones } \{\frac{C}{2},\frac{C}{2}\}$$

Vamos a probar con k=3:

$$MAX(3,C) = Máximo_{1<x<C}\{xMAX(2,C-x)\}$$

Como sabemos que $MAX(2,C) = (\frac{C}{2})^2$,

$$MAX(2,C-x) = (\frac{C-x}{2})^2.$$

Por lo que $MAX(3,C) = Máximo_{1\leq x\leq C}\{x(\frac{C-x}{2})^2\}$

Vamos a sacar ese máximo derivando la función $f(x) = x(\frac{C-x}{2})^2$ e igualándola a 0.

$$f'(x) = \frac{(C-x)^2}{2} - x(C-x)$$

$$\frac{(C-x)^2}{2} - x(C-x) = 0;\ C - x = 2x;\ \text{por lo que } x = \frac{C}{3}$$

Sustituyendo, $MAX(3,C) = \frac{C}{3} \cdot (\frac{C-\frac{C}{3}}{2})^2 = \frac{C}{3} \cdot \frac{2C}{3} = (\frac{C}{3})^3$

Las soluciones son $\{\frac{C}{3}, \frac{C}{3}, \frac{C}{3}\}$, ya que $MAX(3,C) = \frac{C}{3} \cdot MAX(2, C - \frac{C}{3})$, y $MAX(2, C - \frac{C}{3})$ es bi-etápico, con soluciones $\{\frac{C}{3}, \frac{C}{3}\}$

Los dos ejemplos anteriores nos llevan a pensar el máximo valor alcanzable del problema cuando k=i sería $MAX(i,C) = (C/i)^i$, con soluciones $\{\frac{C}{i}, \frac{C}{i}, \frac{C}{i}, ..., \frac{C}{i}\}$

Es decir, planteamos la siguiente ***hipótesis***:

$$MAX(i,C) = Máximo_{1 \leq x \leq C}\{xMAX(i-1, C-x)\} = Máximo_{1 \leq x \leq C}\{x.\frac{(C-x)^{i-1}}{i-1}\}$$

Derivando la función entre corchetes e igualando a 0, comprobamos que el *x* máximo es $\frac{C}{i}$

Para demostrar esto, vamos a usar inducción:

Suponiendo la anterior hipótesis como cierta, vamos a ver si se cumple para k=i+1:

De acuerdo con el POB, tenemos que:

$$MAX(i+1,C) = Máximo_{1 \leq x \leq C}\{xMAX(i, C-x)\} = Máximo_{1 \leq x \leq C}\{x \cdot \frac{(C-x)^i}{i}\}$$

Derivando e igualando a 0 la función entre corchetes, llegamos a que $x = \frac{C}{i+1}$, siendo el máximo valor del producto,

$$MAX(i+1,C) = (\frac{c}{i+1})^{i+1}$$

Vemos que el resultado tiene sentido, por lo que la hipótesis se puede tomar como cierta.

Una vez demostrado por inducción, podemos llegar a la conclusión de que si tomamos como soluciones $\{\frac{C}{n}, \frac{C}{n}, \frac{C}{n}, ..., \frac{C}{n}\}$ alcanzamos el valor máximo, el cual es $(\frac{C}{n})^n$.

Ecuación recurrente:

La ecuación que resolvería este problema, es:

$$MAX(n, C) = \frac{C}{n} \cdot MAX(n-1, C-\frac{C}{n})$$

Un ejemplo de uso: Producto máximo de tres números cuya suma resulte 9.

Anticipadamente, podemos saber el resultado con la fórmula $(\frac{C}{n})^n$: $(\frac{9}{3})^3 = 27$

Usando la ecuación recurrente:

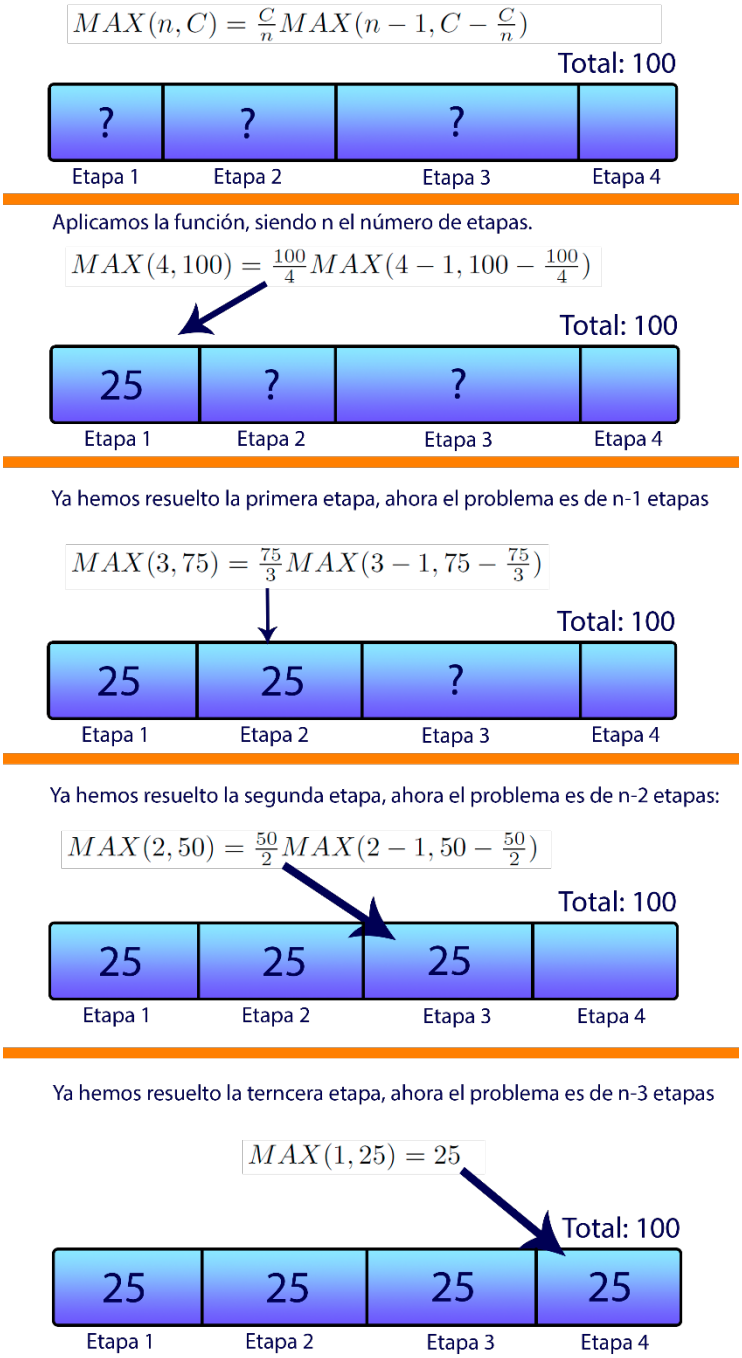
$$MAX(3,9) = \frac{9}{3} \cdot MAX(2,6) = \frac{9}{3} \cdot \frac{6}{2} \cdot MAX(1,3) = \frac{9}{3} \cdot \frac{6}{2} \cdot 3 = 27$$

Como vemos, el resultado es correcto.

Ejemplo visual

Vamos a comprobar visualmente cómo se resuelve el problema mediante PD para poder apreciar la naturaleza de “etapa a etapa”. En este ejemplo, buscaremos 4 números que sumen 100 y cuyo producto sea máximo. Como vemos, en cada recurrencia se resuelve una etapa, es decir, uno de los 4 números que son solución.

Se ve claramente como sigue el POB, en el que en un problema de n etapas, encuentra la solución óptima para una etapa y se vuelve a resolver el problema para las n-1 etapas restantes.



Como vemos, la solución es el producto de los cuatro números resultantes de resolver cada etapa: 25 x 25 x 25 x 25.

Pseudo-código del algoritmo (eficiencia de O(nM)

```
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)
```

Código implementado en C++

```cpp
#include <iostream>
using namespace std;


int MAX( int n, int C ){
  int T[n+1][C+1];
  for( int k=0; k<=n; k++ ){
    T[k][0] = 0;
  }
  for( int k=1; k<=C; k++ ){
    T[1][k] = k;
  }
  for( int k=2; k<=n; k++ ){
    for( int c=1; c<=C; c++ ){
      T[k][c] = 0;
      for( int x=1; x<=c; x++ ){
        if( x*T[k-1][c-x] > T[k][c] ) T[k][c] = x*T[k-1][c-x];
      }
    }
  }

  return T[n][C];
}
```

Ejecución del algoritmo implementado

Ponemos a prueba el código anterior con los siguientes parámetros:

Máximo producto de 3 números que sumen 9.

```
25  int main(){
26    int sol = MAX(3,9);
27    cout << "El producto máximo de 3 números que suman 9 es: " << sol << endl;
28  }
```

El resultado de la ejecución es la siguiente.

```
julioxxx@julio-pc:~/Desktop$ ./alg
El producto máximo de 3 números que suman 9 es: 27
```

Como vemos, el resultado es correcto.