Programación dinámica: Presentación

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski
vpodberezski@fi.uba.ar

Programación dinámica

Metodología de resolución

de problemas de optimización (minimización o maximización)

Nombrada por Richard Bellman

En 1950 mientras trabajaba para la RAND Corporation (una historia interesante!)

Divide el problema en subproblemas

con una jerarquía entre ellos (de menor a mayor tamaño).

Cada subproblema

Puede ser utilizado ser reutilizado en diferentes subproblemas mayores

Propiedades

Un problema debe contener las siguientes propiedades

para poder resolverse de forma optima mediante un algoritmo programación dinámica

- **Subestructura óptima**
- **Subproblemas superpuestos**

Subestructura óptima

Un problema

Contiene una subestructura óptima

Si la solución optima global del mismo

Contiene en su interior las soluciones optimas de sus subproblemas

Subproblemas superpuestos

Un problema

Contiene una subproblemas superpuestos

Si en la resolución de sus subproblemas

Vuelven a aparecer subproblemas previamente calculados

Relación de recurrencia

Se puede resolver el problema recursivamente

Donde por cada problema se abrirán un conjunto de subproblemas

Utilizaremos una Ecuación de recurencia para representarlo

Cada término de la secuencia es definido como una función de términos anteriores

$$T_n = F(T_{n-1}, T_{n-2}, \dots)$$

Existe uno o varios términos base o iniciales desde los cuales se calculan los siguientes.

Memorización

Técnica que consiste en

- En almacenar los resultados de los subproblemas previamente calculados

Para evitar repetir su resolución

- Cuando vuelva a requerirse

De esa forma reducen la cantidad total de subproblemas a calcular

- Consiguiendo reducir significativamente la complejidad temporal de la solución

Ejemplo: Corte de soga

Sea

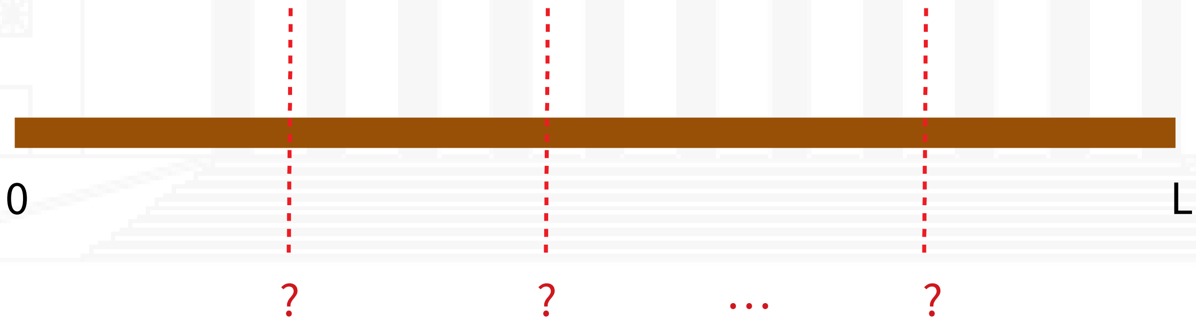
Una soga de longitud L divisible

Una tabla de precios por longitud de la soga

Long.	Gan.
1	p_1
2	p_2
3	p_3
4	p_4
5	p_5

Queremos

Saber que cortes realizar para maximizar la ganancia



Análisis del problema

Cada corte realizado de longitud l_i en la soga de longitud L

- Nos brindara una ganancia de p_i
- Dejara una nueva soga de longitud $L – l_i$ (un subproblema)

Debemos pensar como cortar la soga

- Podemos elegir con un corte inicial y luego continuar cortando con algún criterio

Existe una elección greedy válida?

- No!
- Debemos evaluar todos los corte posibles y elegir el máximo

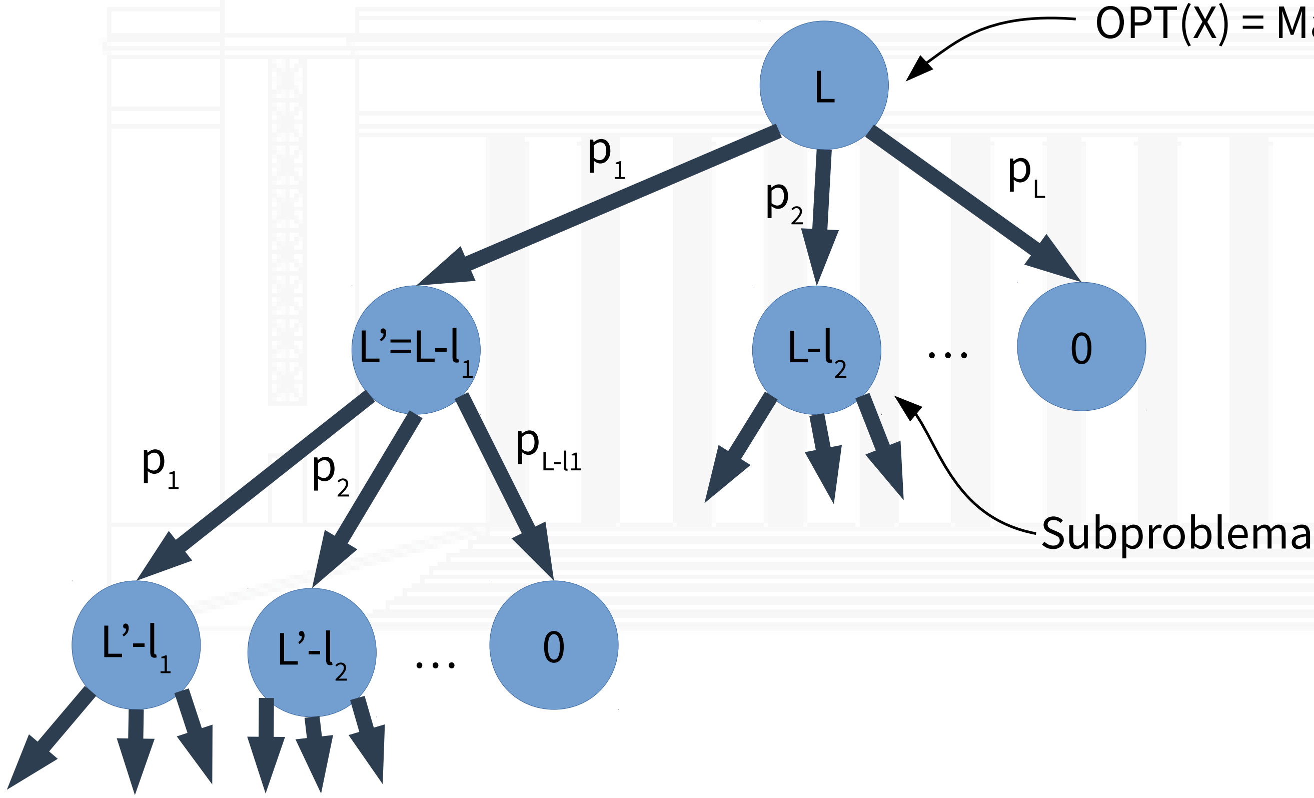
Árbol de decisión

Podemos representar en un árbol

Las diferentes elecciones a realizar

Podemos calcular la solución de un subproblema como:

$OPT(X) = Max_{i=1...n} \{pi + OPT(X\text{-}li)\}$



Cada subproblema x nos dirá cuanto es lo máximo que se puede ganar con su longitud OPT(x)

Solución recursiva – relación de recurrencia

Expresamos cada subproblema de la forma

$$OPT(X) = Max_{i=1}^{X}\{p_i + OPT(X-i)\}, \qquad \text{si } X>0$$

El caso base

$$OPT(X) = 0, \qquad \text{si } X \leq 0$$

¿Cuantos subproblemas tengo que resolver?

Si revistamos completamente el árbol de decisión vemos que existen un numero exponencial de nodos (subproblemas).

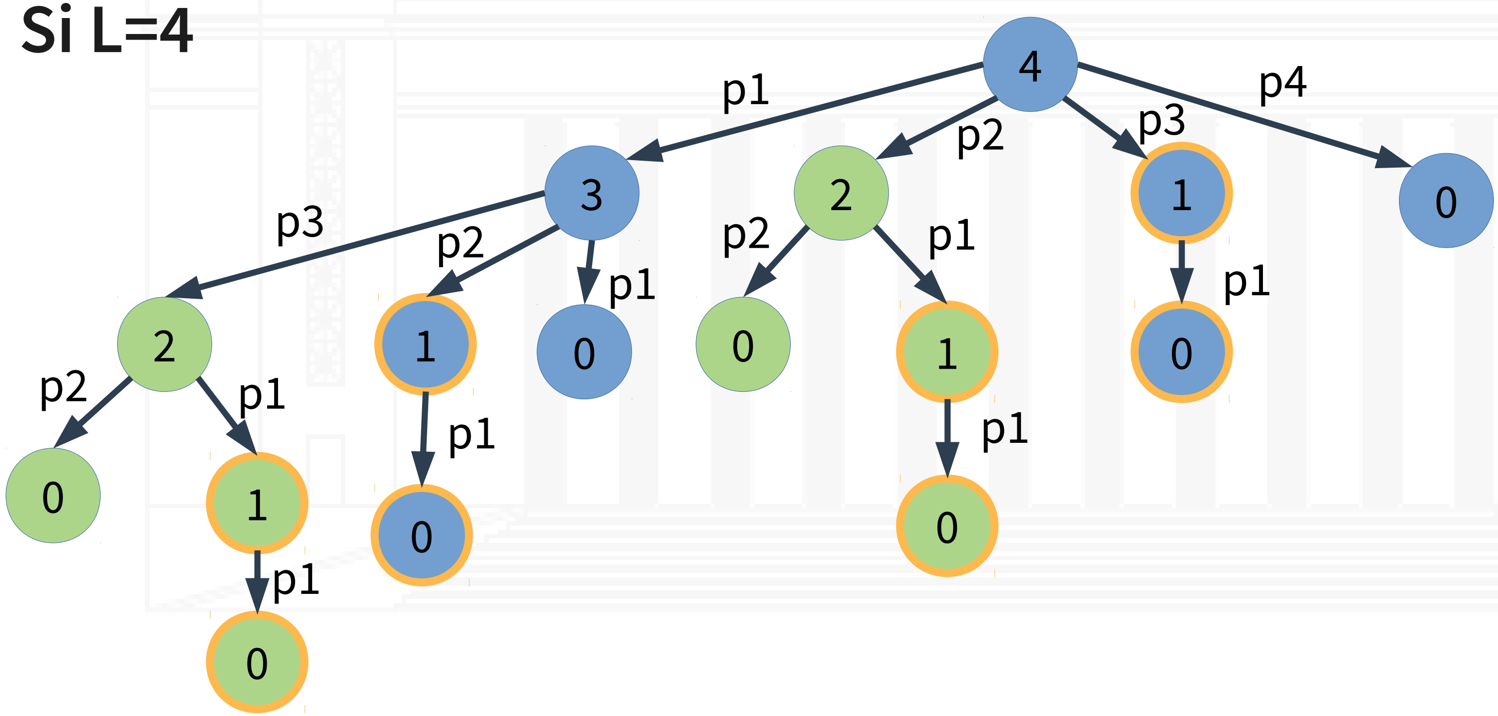
Sin embargo...

Subproblemas superpuestos

Podemos ver que ....

En nuestro árbol de decisión ciertos problemas se repiten.

Si L=4



El Subproblema L=1 se repite 4 veces
El Subproblema L=2 se repite 2 veces

Su resultado no cambia, sin importar por el camino que se llega al mismo

Memorización

OPT	Gan.
1	g_1
2	g_2
3	g_3
4	g_4

Podemos

Calcular la primera vez el resultado y luego utilizarlo.

Almacenaremos en una tabla los suproblemas resueltos

Solo debemos calcular L subproblemas!

(en vez de un numero exponencial!)

La ganancia óptima

estará en la última fila de nuestra tabla de memorización

Solución iterativa

La expresión recursiva

Tiene como inconveniente que dificulta la memorización

(tenemos que recorrer el arbol de raiz a hojas)

Podemos resolver el problema

invirtiendo el orden de la resolución de los subproblemas

De los mas pequeños a los mas grandes,

En nuestro ejemplo:

OPT(0)=0
OPT(1)=p1
OPT(2)= max {p2 + OPT(0) ; p1 + OPT(1)}
...
OPT(L)=...

Pseudocódigo

```
OPT[0] = 0

Desde i=0 a L
     OPT[i] = 0
     Desde j=0 a i

          val = p[j] + OPT(i-j)

          si val>OPT[i]
               OPT[i]=val

Retornar OPT[L]
```

Complejidad espacial

Necesito guardar en la tabla L óptimos → O(L)

Complejidad temporal

Requiero calcular L suproblemas, y para cada uno de ellos evaluar sus subproblemas → $O(L^2)$

Si en hubiesen "c" cortes posibles la complejidad seria O(cL) → O(L)

Reconstrucción de las decisiones

Además de la ganancia

Seria muy útil saber como cortar la soga

Se puede conseguir

Almacenando para cada subproblema cuál corte fue el óptimo

Luego reconstruimos las elecciones partiendo del OPT(L) para atrás

```
OPT[0] = 0
Eleccion[0]=0

Desde i=0 a L
    OPT[i] = 0
    Desde j=0 a i

        val = p[j] + OPT(i-j)

        si val>OPT[i]
            OPT[i]=val
            Eleccion[i]=j

Imprimir OPT[L]

resto=L
Elegido = Eleccion[resto]
Mientras Elegido <> 0
    Imprimir Elegido
    resto = resto - Elegido
    Elegido =Eleccion[resto]
```

Presentación realizada en Octubre de 2020