Programación dinámica: Maximum Subarray Problem

Teoría de Algoritmos I (75.29 / 95.06)

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

Maximum Subarray Problem

Sea

Una lista de “n” elementos ordenados

Cada elemento e_i tiene un valor numérico v_i

Queremos

Calcular un subconjunto contiguo de elementos S

tal que la suma de los valores sea la máxima posible

Ejemplo

Dados los elementos

-3	5	-3	4	2	1	-10	2	2	-2	1	5
1	2	3	4	5	6	7	8	9	10	11	12

El subvector de suma máxima es

-3	5	-3	4	2	1	-10	2	2	-2	1	5
1	2	3	4	5	6	7	8	9	10	11	12

Suma máxima: 9

Solución por fuerza bruta

El elemento 1

Puede ser el elemento inicial de n soluciones

-3	5	-3	4	2	1	-10	2	2	-2	1	5
1	2	3	4	5	6	7	8	9	10	11	12

El elemento 2

Puede ser el elemento inicial de n-1 soluciones

5	-3	4	2	1	-10	2	2	-2	1	5
2	3	4	5	6	7	8	9	10	11	12

…

El elemento n

Puede ser elemento inicial de 1 solución

5
12

Total de subvectores posibles

$n + n-1 + n-2 + \ldots + 1 = n^*(n-1)/2 = (n^2 - n)/2$

Tiempo total de cálculo

Por cada subvector posible, hacer la suma: $O(n^3)$

Podemos hacerlo mejor?

Podemos pensarlo como subproblemas:

MAX(i): El máximo subvector que termina en el elemento i

Para el elemento i=3 tenemos que probar:

(e3)=-3 y (e2,e3)=2 y (e1,e2,e3)=-1

Y quedarnos con el máximo. (e2,e3)

Podemos calcular los n subproblemas

-3	5	-3	4	2	1	-10	2	2	-2	1	5
1	2	3	4	5	6	7	8	9	10	11	12

Y quedarnos con el máximo total

… Lo podemos hacer en $O(n_2)$

… podemos hacerlo aun mejor? SI! Existe una solución usando división y conquista O(nlogn)

… y se puede aun mejor

Algoritmo Kadane

Propuesto por Joseph "Jay" Born Kadane

En 1977 (y resuelto en pocos minutos)

Resuelve el problema de Maximum Subarray Problem

O(n)

La historia de la invención puede leerse en

“programming pearls” por Jon Bentley

https://dl.acm.org/doi/pdf/10.1145/358234.381162

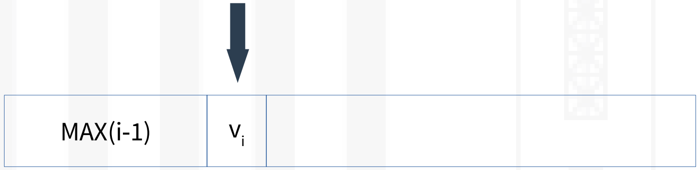
Relación entre subproblemas

El máximo subvector que termina en el elemento i

Esta relacionado con el máximo subvector que termina en el elemento i-1

Si el máximo subvector que termina en i-1

MAX(i-1) > 0 → MAX(i) = MAX(i-1) + vi



MAX(i-1) < 0 → MAX(i) = vi

Recurrencia

Definimos:

$$\begin{cases} \text{MAX}(1) = v[1] \\ \text{MAX}(i) = max\{\text{MAX}(i-1), 0\} + v[1] \end{cases}$$

Queremos obtener:

MAX(n)

Pseudocódigo

Complejidad

Temporal: O(n)

Espacial: O(1)

```
…
MaximoGlobal = v[1]
MaximoLocal = v[1]
IdxFinMaximo = 1

Desde i=2 a n   // elementos
    MaximoLocal = max(MaximoLocal,0) + v[i]

    if MaximoLocal > MaximoGlobal
        MaximoGlobal = MaximoLocal
        IdxFinMaximo = i


Retornar MaximoGlobal
```

Comparación de tiempos de ejecución

TABLE I. Summary of the Algorithms

Algorithm		1	2	3	4
Lines of C Code		8	7	14	7
Run time in microseconds		$3.4N^3$	$13N^2$	$46N \log N$	$33N$
Time to solve problem of size	10^2	3.4 secs	130 msecs	30 msecs	3.3 msecs
	10^3	.94 hrs	13 secs	.45 secs	33 msecs
	10^4	39 days	22 mins	6.1 secs	.33 secs
	10^5	108 yrs	1.5 days	1.3 min	3.3 secs
	10^6	108 mill	5 mos	15 min	33 secs
Max problem solved in one	sec	67	280	2000	30,000
	min	260	2200	82,000	2,000,000
	hr	1000	17,000	3,500,000	120,000,000
	day	3000	81,000	73,000,000	2,800,000,000
If N multiplies by 10, time multiplies by		1000	100	10+	10
If time multiplies by 10, N multiplies by		2.15	3.16	10−	10

Algoritmos: 1) Fuerza bruta. 2) mejoras de fuerza bruta. Tiempo cuadrático. 3) Utilizando división y conquista. 4) Con programación dinámica

Presentación realizada en Abril de 2020