

Note per la Lezione 2

Ugo Vaccaro

Dati un intero $n \geq 1$ ed una generica sequenza $a = a[0]a[1] \cdots a[n-1]$ di numeri (che possono essere sia positivi o negativi), per ogni coppia di indici $0 \leq i \leq j \leq n-1$, denotiamo con $V(i, j)$ la quantità

$$V(i, j) = \sum_{k=i}^j a[k],$$

ovvero $V(i, j)$ è la somma degli elementi della sequenza a dall' i -esimo allo j -esimo, consecutivamente. Denotiamo con $V(a)$ la quantità

$$V(a) = \max_{0 \leq i \leq j \leq n-1} V(i, j),$$

in altre parole $V(a)$ è il *massimo valore* che possiamo ottenere sommando sottosequenze di elementi *consecutivi* della sequenza a . Varrà quindi

$$\forall i, j \quad V(a) \geq V(i, j).$$

Siamo interessati a risolvere il seguente problema algoritmico.

Input: sequenza di numeri $a = a[0]a[1] \cdots a[n-1]$

Output: $V(a) = \max_{0 \leq i \leq j \leq n-1} V(i, j) = \max_{0 \leq i \leq j \leq n-1} \sum_{k=i}^j a[k]$.

Prima possibile soluzione. Calcoliamo la somma di *tutte* le possibili sottosequenze di elementi consecutivi di a , ovvero calcoliamo tutti i possibili valori $V(i, j)$, per ogni coppia di indici (i, j) per cui $0 \leq i \leq j \leq n-1$, e diamo in output la somma massima. Il seguente algoritmo fa proprio questo.

```

1. MaxConsecutivaSomma1(a)
2. SommaMassima= a[0]
3. FOR(i = 0; i < n; i = i + 1){
4.     FOR(j = i; j < n; j = j + 1){
5.         SommaCorrente= 0
6.         FOR(k = i; k < j + 1; k = k + 1){
7.             SommaCorrente=SommaCorrente+a[k]
8.         }
9.         IF(SommaCorrente>SommaMassima){
10.            SommaMassima=SommaCorrente
        }
    }
}
11. RETURN SommaMassima

```

Il ciclo FOR delle istruzioni 6. e 7. calcola $V(i, j) = \sum_{k=i}^j a[k]$. Si vede immediatamente che le istruzioni all'interno del FOR($k = i; k < j + 1; k = k + 1$) possono essere eseguite al più n volte, così come il codice all'interno del ciclo FOR($j = i; j < n; j = j + 1$) ed anche del ciclo FOR($i = 0; i < n; i = i + 1$), per cui l'algoritmo `MaxConsecutivaSomma1(a)` eseguirà al più $c \times n \times n \times n = cn^3$ operazioni elementari,¹ per qualche costante c . In altri termini, la complessità dell'algoritmo `MaxConsecutivaSomma1(a)` su di un input composto da una sequenza di n numeri è $O(n^3)$.

Seconda possibile soluzione. Calcoliamo sempre la somma di tutte le possibili sottosequenze di elementi consecutivi di a , usando questa volta la seguente osservazione. Vale che:

$$V(i, j) = \sum_{k=i}^j a[k] = \sum_{k=i}^{j-1} a[k] + a[j] = V(i, j - 1) + a[j]. \quad (1)$$

Daremo in output sempre il massimo di tutti i valori calcolati, ovvero $\max_{i,j} V(i, j)$. L'osservazione (1) di sopra ci permette di risparmiare tempo, in quanto con *una sola* somma ci possiamo ora calcolare $V(i, j)$ come $V(i, j) = V(i, j - 1) + a[j]$, invece di eseguire il FOR($k = i; k < j + 1; k = k + 1$), come facevamo nel precedente algoritmo. L'algoritmo seguente fa uso dell'osservazione sopra fatta.

```

MaxConsecutivaSomma2(a)
SommaMassima= a[0]
FOR(i = 0; i < n; i = i + 1){
  SommaCorrente= 0
  FOR(j = i; j < n; j = j + 1){
    SommaCorrente=SommaCorrente +a[j]
    IF(SommaCorrente>SommaMassima){
      SommaMassima=SommaCorrente
    }
  }
}
RETURN SommaMassima

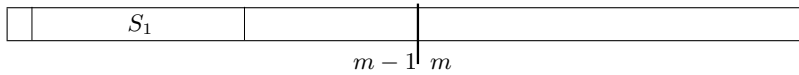
```

Si vede immediatamente che le istruzioni all'interno del FOR($j = i; j < n; j = j + 1$) possono essere eseguite al più n volte, così come le istruzioni all'interno del FOR($i = 0; i < n; i = i + 1$), per cui l'algoritmo `MaxConsecutivaSomma2(a)` eseguirà al più dn^2 operazioni elementari, per qualche costante d . Ovvero, la complessità di `MaxConsecutivaSomma2(a)` è $O(n^2)$.

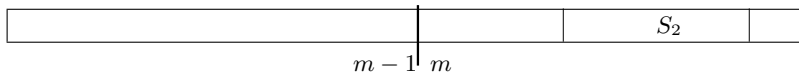
Terza possibile soluzione. Applichiamo la tecnica Divide-et-Impera

Idea: Data la sequenza input di n numeri $a = a[0]a[1] \dots a[n - 1]$, poniamo $m = \lfloor n/2 \rfloor$. La massima somma consecutiva (MSC) della sequenza $a = a[0]a[1] \dots a[n - 1]$ deve necessariamente essere una delle seguenti:
 S_1 = la MSC della sottosequenza $a[0]a[1] \dots a[m - 1]$

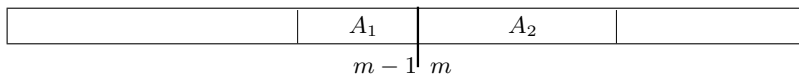
¹Per il momento, per noi un'operazione fondamentale è una qualsiasi operazione nell'insieme {somma tra numeri, moltiplicazione tra numeri (se non troppo grandi...), assegnazione di valori ad una variabile, incremento o decremento di valori ad una variabile, controllo di valori di una variabile, ed operazioni simili...}. Di conseguenza, un'operazione elementare è una operazione che richiede un tempo *costante* per la sua esecuzione, ovvero il suo tempo di esecuzione *non* dipende dalla grandezza dei numeri coinvolti.



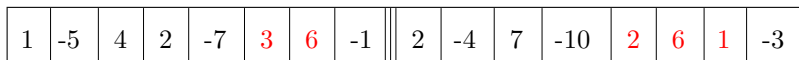
oppure S_2 : la MSC della sottosequenza $a[m]a[m+1] \cdots a[n-1]$



oppure é a "cavallo" di $a[m-1]$, ovvero la massima somma consecutiva di a è della forma $A = A_1 \cup A_2$, con



Vediamo un esempio. Sia a rappresentata nella figura di sotto riportata:



In questo caso $S_1 = [3, 6]$ con valore $3 + 6 = 9$, e $S_2 = [2, 6, 1]$ con valore $2 + 6 + 1 = 9$.

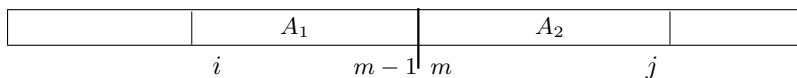


Ma abbiamo anche $A_1 = [3, 6, -1]$, $A_2 = [2, -4, 7]$, con $A = A_1 \cup A_2 = [3, 6, -1, 2, -4, 7]$, di valore totale $3 + 6 - 1 + 2 - 4 + 7 = 13$.

Dobbiamo quindi trovare S_1 , S_2 e A ...

- il valore S_1 lo si trova determinando la MSC di $a[0]a[1] \cdots a[m-1]$, ovvero chiamando l'algoritmo ricorsivamente sulla parte sinistra $a[0]a[1] \cdots a[m-1]$ della sequenza $a[0]a[1] \cdots a[n-1]$
- il valore S_2 lo si trova determinando la MSC di $a[m]a[m+1] \cdots a[n-1]$, ovvero chiamando l'algoritmo ricorsivamente sulla parte destra $a[m]a[m+1] \cdots a[n-1]$ della sequenza $a[0]a[1] \cdots a[n-1]$
- Come trovare il valore A lo vediamo in un secondo ...
- La MSC dell'intera sequenza $a[0]a[1] \cdots a[n-1]$ sarà quindi quel valore tra S_1 , S_2 , ed A , che ha valore *massimo*.

Come trovare A : la fase di Conquer



- A_1 é la massima somma contigua della forma $a[i] \dots a[m-1]$ per qualche valore di $i \in \{0, \dots, m-1\}$ (m è fissato e solo i può variare):

ci sono solo $m \leq n$ tali sequenze, tante quanti sono i corrispondenti valori di $i, 0 \leq i \leq m - 1$. Pertanto la sequenza contigua A_1 di valore massimo può essere trovata usando al più $O(m) = O(n)$ operazioni, con un semplice ciclo FOR, con l'indice i del FOR che varia tra 0 e $m - 1$.

- Analogamente, A_2 è la massima somma contigua della forma $a[m] \dots a[j]$. per qualche valore di $j \in \{m, \dots, n - 1\}$ (m è fissato e solo j può variare):

ci sono solo $n - m \leq n$ tali sequenze, tante quanti sono i corrispondenti valori di $j, m \leq j \leq n - 1$. Pertanto la sequenza contigua A_2 di valore massimo può essere trovata in $O(n - m) = O(n)$ operazioni, sempre con un semplice ciclo FOR, con l'indice j del FOR che varia tra m ed $n - 1$.

Riassumendo

$A = A_1 \cup A_2$ può essere trovato in $O(n)$ operazioni

L'Algoritmo Completo Divide-et-Impera sarà quindi il seguente:

```

MaxConsecutivaSomma3(a, i, j)
1. IF i == j RETURN a[i] ELSE
2.   S1 = MaxConsecutivaSomma3(a, i, [(i + j)/2])
3.   S2 = MaxConsecutivaSomma3(a, [(i + j)/2] + 1, j)
4.   A1 = 0, B1 = a[[(i + j)/2]]
5. FOR(s = [(i + j)/2]; s > i - 1; s = s - 1){
6.     A1 = A1 + a[s]
7.     IF(A1 > B1) {
8.       B1 = A1
9.     }
10.  }
11. A2 = 0, B2 = a[[(i + j)/2] + 1]
12. FOR(t = [(i + j)/2] + 1; t < j + 1; t = t + 1){
13.   A2 = A2 + a[t]
14.   IF(A2 > B2) {
15.     B2 = A2
16.   }
17. }
18. S3 = B1 + B2
19. RETURN MAX(S1, S2, S3)

```

Detto $T(n)$ il numero di operazioni di $\text{MaxConsecutivaSomma3}(a, 0, n - 1)$, abbiamo:

l'istruzione 1. richiede tempo $O(1)$.

Le istruzioni 2. e 3. richiedono tempo $T(n/2)$, ciascuna, in quanto in ciascuna di esse eseguiamo il nostro algoritmo MSC su di una sequenza lunga la metà di quella di partenza.

Le istruzioni 4-15 richiedono in totale tempo $O(n)$.

In totale

$$\boxed{T(n) = 2T(n/2) + O(n)} \implies \boxed{T(n) = O(n \log n)}$$

Quarta possibile soluzione. Per un generico indice $0 \leq i < n$, denotiamo con $M(i)$ la seguente quantità:

$M(i)$ = il valore della massima somma consecutiva che ha come *ultimo* termine uguale a $a[i]$.

Ovviamente vale che

$$V(a) = \max\{M(0), M(1), \dots, M(n-1)\},$$

visto che la somma consecutiva di valore massimo da qualche parte deve pur terminare.

Inoltre, vale che

$$M(i) = \max\{M(i-1) + a[i], a[i]\}. \quad (2)$$

Se $M(i) = a[i]$, non c'è niente da provare. Infatti, se $M(i) = x + a[i]$, allora la somma x ha come ultimo termine il valore $a[i-1]$, e *necessariamente* vale che $x = M(i-1)$. Se non fosse così, ovvero se $x < y = M(i-1)$, allora esisterebbe un'altra somma che termina con $a[i]$, di valore $y + a[i] > x + a[i] = M(i)$, contro l'ipotesi che $M(i)$ è il valore della *massima* somma che ha come ultimo termine $a[i]$.

Il seguente algoritmo calcola $V(a)$, per un'arbitraria sequenza $a = a[0]a[1] \dots a[n-1]$ di n numeri, utilizzando la importante identità (2).

```
MaxConsecutivaSomma4(a)
SommaMassima= a[0]
SommaMassimaFAQ= a[0]
FOR(i = 1; i < n; i = i + 1){
  IF(SommaMassimaFAQ+a[i] > a[i]){
    SommaMassimaFAQ=SommaMassimaFAQ+a[i]
  } ELSE SommaMassimaFAQ= a[i]
}
IF(SommaMassimaFAQ>SommaMassima) {
  SommaMassima=SommaMassimaFAQ
}
}
return SommaMassima
```

Dopo ogni esecuzione del ciclo $\text{FOR}(i = 1; i < n; i = i + 1)$, la variabile `SommaMassimaFAQ` contiene il valore $M(i)$. Visto che l'algoritmo consta di *un solo* ciclo `FOR`, esso eseguirà al più *en* operazioni elementari, per qualche costante e . In altri termini, la complessità di `MaxConsecutivaSomma4(a)` su input a di dimensione n è $O(n)$.

A mò di esempio numerico, supponiamo di eseguire i quattro algoritmo sopra elencati su di un calcolatore che esegue *un miliardo* di operazioni al secondo, per input di dimensione $10^3, 10^4, 10^5, 10^6, 10^8$. Supponendo per semplicità che le costanti all'interno della notazione O siano tutte pari ad 1, avremmo i risultati seguenti:

	n	$n \log n$	n^2	n^3
10^3	0.000001s	0.00000996578s	0.001s	1s
10^4	0.00001s	0.00013287712s	0.1s	≈ 16.6 m
10^5	0.0001s	0.00166096404s	10s	11.5giorni
10^6	0.001s	0.01993156856s	≈ 17 m	≈ 31 anni
10^8	0.1s	2.65754247591s	≈ 6 mesi	troppo...

La morale della lezione consiste in:

- Uno stesso problema algoritmico può essere risolto attraverso varie e differenti tecniche, le quali produrranno vari e differenti algoritmi per la risoluzione dello stesso problema;
- É importante avere algoritmi efficienti.