Esempio di progettazione di algoritmi

Problema computazionale: ricerca del segmento di somma massima

Input: sequenza di n interi $a = [a_1, a_2, \dots, a_n]$

Output: segmento $a[i, j]$ tale che $\sum_{k=i}^{j} a[k]$ sia massima tra tutti i possibili segmenti

Segmento = qualunque sotto-sequenza di elementi consecutivi $a_i, a_{i+1}, \dots, a_j$

A parità di somma massima, si sceglie il "più corto".

Osservazioni preliminari:

- se $a = [a_1, a_2, \dots, a_n]$ contiene SOLO elementi positivi, naturalmente il segmento cercato è a stesso!
- se $a = [a_1, a_2, \dots, a_n]$ contiene SOLO elementi negativi, il segmento è dato dall'elemento il cui valore assoluto è minimo.

Supponiamo pertanto che a contenga sia elementi positivi che negativi.

È facile osservare che

- una soluzione, in questo caso, deve avere gli estremi positivi, altrimenti potremmo avere una soluzione migliore TOGLIENDO gli estremi negativi.

Prima soluzione:

- Generiamo direttamente tutti i segmenti calcolandone la somma e restituiamo quello di somma massima

```
SommaMassima (a)
1   max = 0;
2   FOR (i=0; i<n; i=i+1) {
3       FOR (j=i; j<n; j=j+1) {
4           somma = 0;
5           FOR (K=i; K<=j; K=K+1) {
6               somma = somma + a[K];      /* calcola la somma a[i,j]
7           IF (somma > max) max = somma;
        }
    }
    RETURN MAX;
```

Cosa fa l'algoritmo?

- un segmento è univocamente determinato dalla coppia (i, j) e dai suoi estremi $a[i]$ e $a[j]$
- generiamo tutte le coppie $0 \le i \le j \le n-1$ e calcoliamo le somme dei relativi segmenti

Complessità computazionale?

- Il ciclo più esterno viene eseguito n volte. Il primo ciclo interno al più n volte ad ogni iterazione di quello esterno. Il ciclo più interno viene eseguito ogni volta $j-1+1$ volte, cioè al più n volte (i.e., $i=0$ e $j=n-1$)

Pertanto il costo è $O(n^3)$.

D'altra parte, la riga 6 (somma = somma + a[k];) viene eseguita esattamente

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j-i+1)$$ che può essere riscritta, osservando che

Ponendo $k = j - i + 1$

$j = i \Rightarrow k = j - i + 1 = 1$

$j = n-1 \Rightarrow k = j - i + 1 = n - i$

come $$\sum_{i=0}^{n-1} \sum_{k=1}^{n-i} k .$$

Ricordiamo che $$\boxed{\sum_{k=1}^{n} k = \frac{n \cdot (n+1)}{2}}$$ (FORMULA DI GAUSS) per il calcolo della somma dei primi n numeri interi positivi

possiamo dire che:

$$\sum_{i=0}^{n-1} \sum_{k=1}^{n-i} k = \sum_{i=0}^{n-1} \frac{(n-i)(n-i+1)}{2} \geq \sum_{i=0}^{n-1} \frac{(n-i)^2}{2}$$

$$\geq \frac{1}{2} \cdot \sum_{i=0}^{n/2} (n-i)^2 \geq \frac{1}{2} \cdot \sum_{i=0}^{n/2} \left(\frac{n}{2}\right)^2$$

$$\geq \frac{1}{2} \cdot \frac{n}{2} \cdot \frac{n^2}{4} \geq \frac{n^3}{16} = \Omega(n^3)$$

Pertanto, la complessità computazionale del nostro primo algoritmo è $\Theta(n^3)$.

Si può fare di meglio...

Idea: una volta calcolata la somma a $[i, j-1]$ non ripartiamo dall'inizio per calcolare a $[i, j]$ ma calcoliamo

somma a $[i, j]$ = somma a $[i, j-1]$ + a $[j]$;

```
SommaMassima (a)
1   max = 0;
2   FOR (i=0; i<n; i=i+1){
3       somma = 0;
4       FOR (j = i; j<n; j=j+1) {
5           somma = somma + a[j];
6           IF (somma > max) max = somma;
        }
    }
    RETURN max;
```

Complessità computazionale?

Il ciclo più esterno viene eseguito n volte. Il ciclo interno, per ogni iterazione del ciclo esterno, al più n volte.
Pertanto il costo è $O(n^2)$.

Con conti simili ai precedenti si può mostrare che le righe 5 e 6 vengono eseguite $\Omega(n^2)$ volte.

Quindi la complessità computazionale di questo secondo algoritmo è $\Theta(n^2)$.

Ma se ragioniamo ancora e analizziamo la "struttura" del problema, possiamo far meglio.

L'analisi è "sottile". Anche se ci sono $O(n^2)$ segmenti $a[i,j]$ possibili, si può mostrare che:

- solo $O(n)$ di essi possono essere di somma massima
- sono disgiunti

Infatti, esaminando un segmento $a[i,j]$ di somma massima, deve accadere che sia di lunghezza minima e deve soddisfare due proprietà:

a) Ogni "prefisso" di $a[i,j]$ ha somma positiva, cioè risulta $\text{somma}(a[i,k]) > 0$ per ogni $i \le k < j$.

Se così non fosse, risulterebbe:

$$\text{somma}(a[i,j]) = \underbrace{\text{somma}(a[i,k])}_{\text{prefisso a somma} \le 0} + \text{somma}(a[k,j]) \le \text{somma}(a[k+1,j])$$

e sarebbe una contraddizione in quanto $\text{somma}(a[k+1,j])$ avrebbe valore MAGGIORE e sarebbe PIÙ CORTO!

b) Il segmento $a[i,j]$ NON PUÒ essere esteso a sinistra, cioè $\text{somma}(a[k,i-1]) \le 0$ per ogni $0 \le k \le i-1$.

Se così non fosse, esisterebbe un k tale che

$$\text{somma}(a[k,j]) = \underbrace{\text{somma}(a[k,i-1])}_{\text{estensione a sinistra a somma} > 0} + \text{somma}(a[i,j]) > \text{somma}(a[i,j])$$

e sarebbe una contraddizione in quanto somma(a[k,j]) > somma(a[i,j]).

Proviamo allora a scrivere un codice che rispetta a) e b).

```
SommaMassima (a)

max = 0;
somma = 0;
  FOR (j = 0;  j < n;  j = j + 1) {
    IF (somma > 0) {
        somma = somma + a[j];          a)
    } ELSE {
        somma = a[j];                  b)
    }
  IF (somma > max) max = somma;
}
Return max;
```

Complessità computazionale?

Il ciclo FOR viene eseguito n volte. Pertanto il costo è $\Theta(n)$.

Complessità di spazio delle tre soluzioni?

Tutte e tre utilizzano semplici variabili di "appoggio" (e.g. max, somma ...).

Pertanto la complessità in spazio è $\Theta(1)$.

Circa l'ultimo algoritmo, si noti che:

- Ogni algoritmo che risolve il problema dato DEVE leggere tutti gli n elementi dell'array $a=[a_1, \dots, a_n]$ perché, se così non fosse, ci sarebbe ALMENO un elemento non considerato. Pertanto, ogni algoritmo risolutore ha complessità in tempo $\Omega(n)$. Ciò significa anche dire che la complessità in tempo del PROBLEMA è $\Omega(n)$

- L'algoritmo finale ha complessità in tempo $\Theta(n)$. Pertanto è un algoritmo OTTIMO per il problema.

- Il problema dato ha (pertanto) COMPLESSITÀ in tempo $\Theta(n)$.

- Relativamente allo spazio, ogni algoritmo risolutore deve usare un numero costante di variabili di appoggio. Ciò significa dire che la complessità in spazio del problema è $\Omega(1)$.

- L'algoritmo finale ha complessità in spazio $\Theta(1)$. Pertanto è un algoritmo OTTIMO per il problema.

- Il problema dato ha pertanto COMPLESSITÀ in spazio $\Theta(1)$.

Prova della formula di Gauss

$$\sum_{k=1}^{n} k = \frac{n \cdot (n+1)}{2}$$

Storiella del maestro a scuola ...

Sia $S_n = \sum_{k=1}^{n} k$. Risulta:

$$S_n = 1 + 2 + 3 + \dots + n$$

ma anche

$$S_n = n + n-1 + n-2 + \dots + 1$$

$$2 S_n = (n+1) + (n+1) + \dots + (n+1) = n \cdot (n+1)$$

Pertanto $$S_n = \frac{n \cdot (n+1)}{2} ;$$

Consigli per la lettura (R. Courant e H. Robbins, CHE COS'È LA MATEMATICA?, Universale Bollati Boringhieri). Al cap. 1 trovate questa e altre formule di interesse

Complessità in tempo algoritmo 2.

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} 2 = 2 \sum_{i=0}^{n-1} (n-1-i) = \left(\begin{array}{l} \text{Cambio di variabile} \\ \text{Ponendo } j = n-1-i \text{ si ha} \\ i=0 \Rightarrow j = n-1 \\ i=n-1 \Rightarrow j = 0 \end{array} \right)$$

somma IF( )

$$= 2 \sum_{j=0}^{n-1} j = 2 \cdot \frac{(n-1) \cdot n}{2} = \Omega(n^2)$$