UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Corso di Introduzione agli algoritmi
Prof.ssa Tiziana Calamoneri

Il costo di un algoritmo

Costo di un algoritmo (1)

- E' ragionevole pensare che il costo computazionale, inteso come funzione che rappresenta il tempo di esecuzione di un algoritmo, sia una funzione monotona non decrescente della dimensione dell'input.
- Quindi, prima di passare al calcolo del costo, bisogna definire quale sia la **dimensione dell'input**.
- Trovare questo parametro è, di solito, abbastanza semplice: in un algoritmo di ordinamento esso sarà il numero di dati, in un algoritmo che lavora su una matrice sarà il numero di righe e di colonne, in un algoritmo che opera su alberi sarà il numero di nodi, ecc..
- Tuttavia, vi sono casi in cui l'individuazione del parametro non è banale

Costo di un algoritmo (2)

- La notazione asintotica viene sfruttata pesantemente per il calcolo del costo computazionale degli algoritmi, quindi - in base alla definizione stessa – tale costo computazionale potrà essere ritenuto valido ***solo*** asintoticamente.
- In effetti, esistono degli algoritmi che per dimensioni dell'input relativamente piccole hanno un certo comportamento, mentre per dimensioni maggiori un altro.

Costo di un algoritmo (3)

Un dato algoritmo potrebbe avere tempi di esecuzione (e quindi costo computazionale) diversi a seconda dell'input

casi migliore e peggiore (input particolarmente vantaggioso e, rispettivamente, svantaggioso, ai fini del costo computazionale dell'algoritmo)

- Per avere un'idea di quale sia il tempo di esecuzione atteso di un algoritmo, a prescindere dall'input ->caso peggiore
- Vogliamo essere il più precisi possibile e quindi, nel contesto del caso peggiore, cerchiamo di calcolare il costo in termini di notazione asintotica Θ.
- Laddove questo non sia possibile, essa dovrà essere approssimata per difetto (tramite la notazione Ω) e per eccesso (tramite la notazione O).

Costo di un algoritmo (4)

Regole generali per valutare la complessità computazionale di un algoritmo:

- Il costo dell'algoritmo è pari alla somma delle complessità delle istruzioni che lo compongono
- le **istruzioni elementari** (operazioni aritmetiche, lettura del valore di una variabile, assegnazione di un valore a una variabile, valutazione di una condizione logica su un numero costante di operandi, stampa del valore di una variabile, ecc.) hanno costo costante
  - questa costante è diversa da istruzione a istruzione ➔ $\Theta(1)$;
- l'istruzione **if (condizione) then istruzione1 else istruzione2** ha costo pari al costo di verifica della condizione (di solito costante) più il massimo dei costi di istruzione1 e istruzione2
- …

Costo di un algoritmo (5)

Regole generali per valutare la complessità computazionale di un algoritmo (segue):

- le **istruzioni iterative** (o iterazioni: cicli for, while e repeat) hanno un costo pari alla somma dei costi massimi di ciascuna delle iterazioni (compreso il costo di verifica della condizione). Se tutte le iterazioni hanno lo stesso costo, allora il costo dell'iterazione è pari al al prodotto del costo massimo di una singola iterazione per il numero di iterazioni; in entrambi i casi è comunque necessario stimare il numero delle iterazioni.
  Oss. la condizione viene valutata una volta in più rispetto al numero delle iterazioni, poiché l'ultima valutazione, che dà esito negativo, è quella che fa terminare l'iterazione

Costo di un algoritmo (6)

Esempio. Calcolo del massimo di un vettore disordinato contenente *n* valori.

```
Funzione Trova_Max (A: vettore)
1       max ← A[1]                          Θ(1)
2       for i = 2 to n do                   (n – 1) iterazioni più Θ(1)
3          if A[i] > max                          Θ(1)
4                then max ← A[i]            Θ(1)
5   stampa max                              Θ(1)
```

$T(n) = \Theta(1) + [(n-1)\,\Theta(1) + \Theta(1)] + \Theta(1) = \Theta(n)$

Costo di un algoritmo (7)

Esempio. Calcolo della somma dei primi *n* interi.

```
Funzione Calcola_Somma(n: intero)
1       somma ← 0                       Θ(1)
2       for i = 1 to n do               n  iterazioni più Θ(1)
3          aggiungi i a somma                 Θ(1)
4   stampa somma                        Θ(1)
```

$T(n) = \Theta(1) + [n\,\Theta(1) + \Theta(1)] + \Theta(1) = \Theta(n)$

Esempio. Calcolo della somma dei primi *n* interi in modo più efficiente.

Funzione Calcola_Somma(n: intero)

1	somma ← n (n-1)/2	$\Theta(1)$
2	stampa somma	$\Theta(1)$

$T(n) = \Theta(1)$

N.B. Perseguire l'efficienza!

Costo di un algoritmo (9)

Esempio. Valutazione del polinomio $\sum_{i=0}^{n} a_i x^i$ nel punto $x=c$.

Funzione Calcola_Polinomio(n: intero, A: vettore, c reale)

```
                                                  /* A contiene i coefficienti
1  somma ← A[0]                                   Θ(1)
2  for i = 1 to n do                              n iteraz. più Θ(1)
3     potenza ← 1                                 Θ(1)
4     for j = 1 to i do                           i iterazioni più Θ(1)
5        potenza ← c*potenza                      Θ(1)
6        somma ← somma + A[i]*potenza             Θ(1)
7  stampa somma                                   Θ(1)
```

$$T(n) = \Theta(1) + \left[\sum_{i=1}^{n} (\Theta(1) + \Theta(i)) + \Theta(1) \right] + \Theta(1) = \Theta(1) + \Theta(n) + \Theta(n^2) = \Theta(n^2)$$

Costo di un algoritmo (10)

Esempio. Valutazione del polinomio $\sum_{i=0}^{n} a_i x^i$ nel punto $x=c$ in modo più efficiente.

Funzione Calcola_Polinomio(n: intero, A: vettore, c reale)

1	somma ← A[0]	$\Theta(1)$
2	potenza ← 1	$\Theta(1)$
3	for i = 1 to n do	n iteraz. più $\Theta(1)$
4	potenza ← c*potenza	$\Theta(1)$
6	somma ← somma + A[i]*potenza	$\Theta(1)$
7	stampa somma	$\Theta(1)$

$T(n)=\Theta(1)+\Theta(1)+[(n-1)\ \Theta(1)+\Theta(1)]+\Theta(1)=\Theta(n)$

Costo di un algoritmo (11)

Studiamo ora come variano i tempi di esecuzione di un algoritmo in funzione del suo costo computazionale.

- Ipotizziamo di disporre di un sistema di calcolo in grado di effettuare una operazione elementare in un nanosecondo (10^9 operazioni al secondo), e supponiamo che la dimensione del problema sia $n = 10^6$:
- tempo computaz. $O(n)$
  tempo di esecuzione: 1 millesimo di secondo;
- tempo computaz. $O(n \log n)$
  tempo di esecuzione: 20 millesimi di secondo;
- Tempo computaz. $O(n^2)$
  tempo di esecuzione: 1000 secondi = 16 minuti e 40 secondi.

N.B. abbiamo ignorato le costanti

Costo di un algoritmo (12)

E che succede se il costo cresce esponenzialmente, ad esempio quando è $O(2^n)$?

Se anche $n = 100$ con il precedente sistema di calcolo, il tempo computazionale è di $1{,}26*10^{21}$ secondi, cioè circa $3*10^{13}$ anni!

Costo di un algoritmo (13)

Si potrebbe pensare che l'avanzamento tecnologico possa prima o poi rendere abbordabile un tale problema, ma purtroppo non è così:

- supponiamo di avere un calcolatore estremamente potente che riesce a risolvere un problema di dimensione $n = 1000$, avente complessità $O(2^n)$, in un determinato tempo T
- quale dimensione $n' = n + x$ del problema riusciremmo a risolvere nello stesso tempo utilizzando un calcolatore mille volte più veloce?
- $T = \frac{2^{1000}\ operazioni}{10^k\ operazioni\ al\ secondo} = \frac{2^{1000+x}\ operazioni}{10^{k+3}\ operazioni\ al\ secondo}$

$$\frac{2^{1000+x}}{2^{1000}} = 2^x = \frac{10^{k+3}}{10^k} = 10^3 = 1000$$

da cui $x = log\ 1000 \approx 10$

Costo di un algoritmo (14)

Dunque, con un calcolatore mille volte più veloce riusciremmo solo a risolvere, nello stesso tempo, un problema di dimensione 1010 anziché di dimensione 1000!!!

In effetti esiste un'importantissima branca della teoria della complessità che si occupa proprio di caratterizzare i cosiddetti problemi ***intrattabili***, ossia quei problemi il cui costo computazionale è tale per cui essi non sono né saranno mai risolubili per dimensioni realistiche dell'input.

Esercizi (1)

Calcolare il costo del seguente algoritmo scritto in pseudocodice, distinguendo tra caso migliore e caso peggiore se necessario:

```
Funzione Insertion_Sort(A[1..n])
1  for j = 2 to n do
2        x ← A[j]
3        i ← j - 1
4        while ((i > 0)and(A[i] > x))
5                A[i+1] ← A[i]
6                i ← i - 1
7        A[i+1] ← x
```

Esercizi (2)

Calcolare il costo del seguente algoritmo scritto in pseudocodice, distinguendo tra caso migliore e caso peggiore se necessario:

```
Funzione Selection_Sort(A[1..n])
1  for i = 1 to n – 1 do
2        m ← i
3        for j = i + 1 to n do
4              if (A[j] < A[m])
5                    m ← j
6        Scambia A[m] e A[i]
```

Esercizi (3)

Calcolare il costo del seguente algoritmo scritto in pseudocodice, distinguendo tra caso migliore e caso peggiore se necessario:

```
Funzione Bubble_Sort(A[1..n])
1  for i = 1 to n do
2    for j = n downto i + 1 do
3          if (A[j] < A[j - 1])
4                  Scambia A[j] e A[j - 1]
```