COMPLESSITÀ COMPUTAZIONALE DEGLI ALGORITMI

Fondamenti di Informatica
a.a.2006/07
Prof. V.L. Plantamura
Dott.ssa A. Angelini

Studio degli algoritmi

Dato un problema P, le problematiche riguardano:

- **Sintesi o progetto**. Costruzione di un algoritmo A che risolve P;
- **Analisi**. Dato un algoritmo A:
  - Dimostrare che A risolve P (*correttezza*);
  - Valutare la quantità di risorse utilizzate da A (*complessità*);
- **Classificazione**. Data una quantità T di risorse, individuare la classe dei problemi risolubili da algoritmi che utilizzano al più tale quantità.

Confronto di algoritmi

- Valutare la quantità di risorse utilizzate da un algoritmo per la risoluzione di un dato problema.
- Uno stesso problema può essere risolto in modi diversi, cioè con algoritmi diversi;
- Algoritmi diversi possono avere diversa complessità (sforzo di applicazione, costo) e quindi diversa efficienza;
- Le differenze possono essere di poco conto nella manipolazione di piccole quantità di dati ma crescono proporzionalmente alla loro quantità.

Obiettivo

- Studio di un modello di calcolo per la misura dell'efficienza di algoritmi.

Efficienza

- In fase di progetto ci sono due obiettivi spesso in antitesi:
  - Facilità di comprensione, codifica e test
  - Efficienza in termini di spazio e tempo
- Fattori di influenza per una scelta di efficienza:
  - *Utilizzo del programma* → costi di esecuzione maggiori dei costi di progettazione e implementazione
  - *Fattori di sicurezza* quali controllo di processi pericolosi e sistemi in tempo reale → efficienza prioritaria
- Risorse principali: spazio e tempo. Il **tempo** è da privilegiare in quanto risorsa non riutilizzabile.

Efficienza

- Prima definizione intuitiva di efficienza: un programma è più efficiente di un altro se la sua esecuzione richiede meno tempo di calcolo.

Esempio

- Elenchiamo il tempo di esecuzione in ***secondi*** di 4 programmi che implementano rispettivamente due algoritmi A_1 e A_2 su due diverse architetture.

	computer 1		computer 2	
Dim. Input	A_1	A_2	A_1	A_2
50	0,005	0,07	0,05	0,25
100	0,003	0,13	0,18	0,55
200	0,13	0,27	0,73	1,18
300	0,32	0,42	1,65	1,85
400	0,55	0,57	2,93	**2,57**
500	0,87	**0,72**	4,60	**3,28**
1000	3,57	**1,50**	18,32	**7,03**

Si può notare che: la ***superiorità del secondo algoritmo*** non è evidente per input di dimensione piccola

CI sono diversi fattori che influenzano il tempo di esecuzione!

Fattori di influenza

- La bontà del compilatore;
- La struttura del sistema di elaborazione;
- La potenza di calcolo;
- I dati in ingresso (dimensione e bontà);
- La complessità temporale dell'algoritmo sottostante.

Fattori di influenza

- La potenza di calcolo, la bontà del compilatore e la struttura del sistema di elaborazione non rendono la misura oggettiva;
- Non è possibile esprimere la complessità temporale intrinseca di un algoritmo utilizzando unità di misura quali i secondi.

Complessità Temporale

- Obiettivo: studio di un modello per la misura dell'efficienza di un programma che sia *indipendente* dal sistema di elaborazione sul quale lo si intende eseguire;
- Per ottenere una misura oggettiva della complessità temporale, cerchiamo un modello di calcolo che tenga conto:
  - Dell'algoritmo;
  - Dei dati in ingresso.

Complessità Temporale

- Considerato un algoritmo A, vogliamo caratterizzare la sua complessità computazionale (temporale);
- Sia T il tempo di esecuzione richiesto da A su un input x. Calcolare $T_A(x)$ può essere molto complicato a causa del variare di x sull'insieme di tutti gli input.

Dimensione dell'input

- Sia allora |x| la dimensione di una istanza, che raggruppa tutti gli input che hanno la stessa dimensione;
- "Dimensione dell'input"= Cardinalità dei dati in ingresso;
- Osservazione: occorre definire in anticipo come si misura la dimensione della particolare istanza del problema. Non esiste un criterio universale.

Esempi

- Nel caso di algoritmi di ordinamento una dimensione possibile è data dal numero di elementi da ordinare ($|(50,4,1,9,8)|=5$);
- Nel caso di algoritmi di addizione e moltiplicazione una misura possibile è data dal numero di cifre decimali necessario ad esprimere la lunghezza degli operandi.

Definizione intuitiva

- Indipendentemente dal tipo di dati, indichiamo con “n” la dimensione dell’input;
- Dato un algoritmo A, siamo interessati a trovare $\boldsymbol{T_A(n)}$**;**
- *Def*: La complessità temporale di un algoritmo A su una istanza x di dimensione n del problema P è uguale al *numero di passi* $T_A(n)$ necessari per eseguire l’algoritmo.

Modello di calcolo …

- Dovendo prescindere da un particolare compilatore o architettura, il modello che si assumerà per il calcolo di $\boldsymbol{T_A(n)}$ è la RAM: modello computazionale mono-processore in cui tutte le istruzioni sono eseguite una dopo l’altra, senza alcun parallelismo;
- L’accesso ad ogni cella di memoria avviene in tempo costante;

… Modello di calcolo

- In questo modello le istruzioni semplici hanno un costo unitario in termini di tempo impiegato per la loro esecuzione;
- Su questa base si procede al calcolo del *tempo complessivo T impiegato da tutte le istruzioni dell'algoritmo*, tenendo presente che il tempo dovrà dipendere solo dalla dimensione dei dati: n.

Costi delle istruzioni

- Operazione di *costo unitario*, la cui esecuzione non dipende dai valori e dai tipi di dati delle variabili (ossia dalla dimensione dell'input):
  - Lettura: `scanf ()`
  - Scrittura: `printf ()`
  - Assegnamento, operazioni aritmetiche predefinite, return:

```
y=sqrt(x)+3;
return x;
```

Costi delle istruzioni

- Accesso ad un qualunque elemento di un array residente in memoria centrale:

```
A[i] = A[1];
x=A[10000];
```

- Valutazione di una qualsiasi espressione booleana:

```
if ((x>100) || ((j<=n) && (B == true)))
{…
```

Costi delle istruzioni

- Altre operazioni di costo *non* unitario:
  - Istruzione composta:
    somma dei costi delle istruzioni componenti
  - Istruzione ciclo:
    costo totale della condizione + costo totale del corpo

Costi delle istruzioni

  - Istruzione condizionale:
    costo della condizione + il costo del ramo **then** (se la condizione è vera) oppure il costo del ramo **else** (se la condizione è falsa)
  - Attivazione di funzione:
    costo di tutte le istruzioni che la compongono (eventualmente tenendo conto di attivazioni al suo interno)

Esempi

1)
```
i = 1;
while (i <= n)
        i = i+1;
```

Assegnamento esterno	1	+
Num. di test (condizione while)	n+1	+
Assegn. interni (corpo while)	n*1	+
Numero totale di passi base	2*n+2	

Esempi

2)
```
i =1;
while (i <= n) {
   i = i + 1;
   j = j*3 + 42;
}
```

Assegnamento esterno	1	+
Numero di test	n+1	+
Assegnamenti interni	n*2	+
Numero totale di passi base	3*n+2	

Esempi

3)
```
i = 1;
while (i <= 2*n) {
   i = i + 1;
   j = j*3 + 4367;
}
```

Assegnamento esterno	1	+
Numero di test	2*n+1	+
Assegnamenti interni	(2*n)*2	+
Numero totale di passi base	6*n+2	

Esempi

4)
```
i = 1;
while (i <= n) {                    //n+1 test
   for (j = 1; j < n; j ++) {       //un ciclo for per ogni ripetizione while
         printf ("CIAO!");  }       //una scrittura per ogni ripetizione for
   i = i + 1; }                     //un assegnamento per ogni while
```

Assegnamento esterno	1	+
Test while	n+1	+
Numero cicli while	n	*[
For	n	+
Corpo for	(n-1)*1	+
Assegn. Interno while	1	]
Numero totale di passi base	$2*n^2+n+2$	

Esempi

5)
```
i =1;
while (i *i <= n) {
    i = i + 1;
    }
```

Quanti test vengono eseguiti?
Caso n=9:
i=1
i^2=1 <=9 ? SI i:=2
i^2=4 <=9 ? SI i:=3
i^2=9 <=9 ? SI i:=4
i^2=16 <=9 ? NO FINE

Si cicla 3=√9 volte, eseguendo 4= √9 + 1 test.
La complessità di questo blocco è dunque:
1+ √n + 1 + √n = 2+ 2√n passi base (con arrotondamento all'intero superiore)