Algoritmi
e
Strutture Dati

v0.3.0

Luca De Franceschi Università degli studi di Padova

Diario delle modifiche

Autore	**Versione**	**Data**	**Descrizione**
Luca De Franceschi	0.4.0	11/06/2014	Inserito capitolo analisi complessità con metodo dell'integrale e metodo dell'esperto
Luca De Franceschi	0.3.0	11/06/2014	Inserita spiegazione metodo di sostituzione
Luca De Franceschi	0.2.0	11/06/2014	Inserita teoria su programmazione dinamica
Luca De Franceschi	0.1.0	11/06/2014	Creata struttura del documento

Indice

1 Stima della complessità di un algoritmo

Identifichiamo dei casi base, studiando la complessità degli algoritmi noti.

1. Le operazioni elementari, messe al di fuori dei cicli, e che riguardano l'uso di variabili hanno complessità costante c_i, aprossimabile a 0 nello studio della complessità asintotica;
2. Da *insertion-sort* si vede che un *for i = 2 to n* ha complessità pari a n. Constatiamo dunque che un ciclo for che va dall'indice 1 all'indice n avrà complessità $c_i(n+1)$;
3. Tutte le operazioni elementari che compaiono all'interno di un ciclo for di complessità $c_i(n+1)$ hanno complessità $c_i n$;
4. Per i cicli annidati in altri cicli a complessità è data da $c_i \sum_{j=x}^{n}(c_j)$, dove gli estremi della sommatoria sono gli estremi del del ciclo esterno.

Per valutare la complessità si scrive l'equazione $T(n)$ sommando tutte le complessità. Per studiare le sommatorie si utilizza il **metodo dell'integrale**.

1.1 Metodo dell'integrale

Se $f(x)$ è una funzione **non decrescente**:

$$\int_a^{b+1} f(x)\mathrm{d}x \leq \sum_{i=a}^{b} f(i) \leq \int_{a-1}^{b+1} f(x)\mathrm{d}x$$

Se $f(x)$ è una funzione **non decrescente**:

$$\int_a^{b+1} f(x)\mathrm{d}x \leq \sum_{i=a}^{b} f(i) \leq \int_{a-1}^{b} f(x)\mathrm{d}x$$

Inoltre riportiamo di seguito le comuni sommatorie:

- **Serie aritmetica**: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$;
- **Serie geometrica**: $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$ $\quad q \neq 1$

1.2 Andamento asintotico

Una volta ottenuta una funzione che rappresenta la complessità dell'algoritmo ci può interessare prendere in esame l'andamento asintotico della medesima. Per farlo introduciamo le seguenti notazioni:

- **"O" grande**: date due funzioni $f(n)$ e $g(n)$ si dice che $f(n)$ è "O" grande di $g(n)$ se esiste un $c > 0$ e un h_0 tali che:

  $$f(n) \leq cg(n)$$

  per $n \geq h_0$ (**limite asintotico superiore**). In pratica l'ordine di crescita di $f(n)$ è non superiore a quello di $g(n)$;

- **"Ω" grande**: date $f(n)$ e $g(n)$ si dice che $f(n)$ è "Ω" grande di $g(n)$ se esiste una costante $c > 0$ e un h_0 tale che:

  $$f(n) \geq cg(n)$$

  per $n \geq h_0$ (**limite asintotico inferiore**). In pratica l'ordine di crescita di $f(n)$ è non inferiore a quello di $g(n)$;

- **"Θ" grande**: date $f(n)$ e $g(n)$ si dice che $f(n)$ è "Θ" grande di $g(n)$ se ci sono costanti positive c_1, c_2 e un h_0 tali che:

  $$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

  per $n \geq h_0$ (**limite asintotico stretto**). In pratica diciamo che se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$ allora è vero anche che $f(n) = \Theta(g(n))$

Di una funzione non ci interessa la sua forma ma il suo comportamento asintotico. Spesso è possibile determinare dei limiti asintotici calcolando il limite di un rapporto:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)}$$

In base al risultato di questo limite ho tre casi:

1. Ottengo un valore costante $k > 0$: in questo caso $f(n)$ è dello stesso ordine di $g(n)$, e dunque:

   $$\forall \epsilon > 0, \exists h_0 | h \geq h_0 : k - \epsilon \leq f(n)/g(n) \leq k + \epsilon$$

   ponendo:

   $$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

   Dunque concludo dicendo che $f(n) = \Theta(g(n))$;

2. Il limite tende a ∞: $f(n) = \Omega(g(n))$;

3. Il limite tende a 0: $f(n) = O(g(n))$.

1.3 Metodo dell'esperto

Per risolvere le ricorrenze il primo metodo da utilizzare è il **metodo dell'esperto**. Se la ricorrenza è espressa nella forma:

$$T(n) = aT(n/b) + f(n)$$

e se $a \geq 1$ e $b < 1$ allora:

1. Tolgo eventuali arrotondamenti;

2. Calcolo $\log_b a$ e calcolo il limite: $\lim_{n \to \infty} \frac{f(n)}{n^{\log_b a}}$;

A questo punto, in base al valore del limite ho 3 possibili casi:

1.3.1 Caso 2

Se il limite è **finito** e diverso da zero:

$$f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} \log n)$$

1.3.2 Caso 1

Se il limite è **uguale a zero** devo trovare un valore $\epsilon > 0$ per il quale risulta finito il limite:

$$\lim_{n \to \infty} \frac{f(n)}{n^{\log_b a - \epsilon}} = k$$

Se lo trovo allora posso affermare che:

$$f(n) = O(n^{\log_b a - \epsilon}) \Rightarrow T(n) = \Theta(n^{\log_b a})$$

1.3.3 Caso 3

Se il limite è $\int$ allora devo trovare un $\epsilon > 0$ per il quale risulti:

$$\lim_{n \to \infty} \frac{f(n)}{n^{\log_b a + \epsilon}} \neq 0$$

Se lo trovo allora devo studiare l'equazione:

$$af(n/b) \leq k(f(n))$$

se trovo un $k < 1$ allora posso concludere che:

$$f(n) = \Omega(n^{\log_b a + \epsilon}) \Rightarrow T(n) = \Theta(f(n))$$

1.4 Metodo di sostituzione

Se non riesco ad applicare il metodo dell'esperto allora devo utilizzare il **metodo di sostituzione**.

Per capire il metodo di sostituzione proviamo a risolvere il seguente esercizio:

La ricorrenza $T(n) = 4T(n/2) + n^2 \log n$ si può risolvere con il metodo dell'esperto? Giustificare la risposta. Se la risposta è negativa usare il metodo di sostituzione per dimostrare che $T(n) = O(n^2 \log^2 n)$.

Anzitutto vediamo i dati a disposizione:

$$a = 4, b = 2$$
$$f(n) = n^2 \log n$$
$$g(n) = n^{\log_b a} = n^{\log_2 4} = n^2$$

Calcoliamo ora il limite:

$$\lim_{n \to +\infty} \frac{n^2 \log n}{n^2} = \infty$$

Da cui deduco che:

$$f(n) = \Omega(n^2)$$

Potrei dunque essere nel caso 3. Devo trovare un

$$\epsilon > 0$$

tale che:

$$\lim_{n \to +\infty} \frac{n^2 \log n}{n^{2+\epsilon}} \neq 0$$

Ma mi accorgo subito che la cosa è impossibile, in quanto il denominatore, incrementando l'esponente, crescerà molto più velocemente rispetto al numeratore, per cui avrò sempre un valore tendente allo zero. Da questa considerazione deduco che la ricorrenza **non è risolvibile con il metodo dell'esperto**.

Procedo dunque con la sostituzione. Proviamo $T(n) = O(n^2 \log^2 n)$. Assumiamo che per un'opportuna costante $C > 1$ e $\forall x < n$ sia verificata la disuguaglianza $T(x) \leq C(x^2 \log^2 x)$ e dimostriamo che vale anche per n:

$$\begin{aligned} T(n) = 4T(n/2) + n^2 \log n &\leq 4C(n/2)^2 \log^2(n/2) + n^2 \log n \\ &= Cn^2(\log n - 1)^2 + n^2 \log n \\ &= Cn^2(\log^2 n - 2\log n + 1) + n^2 \log n \\ &= Cn^2 \log^2 n - 2Cn^2 \log n + Cn^2 \log n + Cn^2 + n^2 \log n \\ &= Cn^2 \log^2 n - (C-1)n^2 \log n - Cn^2(\log n - 1) \end{aligned}$$

Ora applico una **maggiorazione**:

$$\leq Cn^2 \log^2 n$$

Dunque ho dimostrato che: $T(n) = O(n^2 \log^2 n)$

2 Programmazione dinamica

In maniera del tutto generale la programmazione dinamica può essere descritta nel seguente modo:

1. Identifichiamo dei **sottoproblemi** del problema originario e utilizziamo una *tabella* per memorizzare i risultati intermedi;
2. Inizialmente vanno definiti i **valori iniziali** di alcuni elementi della tabella, corrispondenti a sottoproblemi più semplici;
3. Al generico passo, avanziamo in modo opportuno sulla tabella calcolando il valore della soluzione di un sottoproblema in base alla soluzione di sottoproblemi precedentemente risolti (corrispondenti ad elementi della tabella precedentemente calcolati);
4. Alla fine restituiamo la soluzione del problema originario, che è stato memorizzato in un particolare elemento della tabella.

La programmazione dinamica è usata normalmente per **problemi di ottimizzazione**, il termine "programmazione" si riferisce al metodo tabulare, non alla scrittura di codice.

La programmazione è applicabile con vantaggi se:

- Gode della proprietà di **sottostruttura ottima**: una soluzione si può costruire a partire da soluzioni ottime di sottoproblemi;
- Il numero di sottoproblemi distinti è molto minore del numero di soluzioni possibili tra cui cercare quella ottima, altrimenti c'è la **ripetizione di sottoproblemi**, ovvero se il numero di sottoproblemi distinti è molto minore del numero di soluzioni possibili tra cui cercare quella ottima, allora uno stesso sottoproblema deve comparire molte volte come sottoproblema di altri sottoproblemi.

Ordine di calcolo delle soluzioni dei sottoproblemi

Bottom-up: le soluzioni dei sottoproblemi del problema in esame sono già state calcolate. È il metodo migliore se per il calcolo della soluzione globale servono le soluzioni di tutti i sottoproblemi.

Top-down: è una procedura ricorsiva che dall'alto scende verso il basso. È la soluzione migliore se per il calcolo della soluzione globale servono soltanto alcune delle soluzioni dei sottoproblemi.