Appunti di Algoritmi e Strutture Dati

Marco Natali

30 gennaio 2018

Capitolo 1

Introduzione agli Algoritmi

Il termine Algoritmo proviene da Mohammed ibn-Musa al-Khwarizmi, matematico uzbeco del IX secolo a.c. da cui proviene la moderna Algebra.
Algoritmo:sequenza di passi che portano alla risoluzione di un problema

1.1 Pseudocodice

Lo pseudocodice è un linguaggio, ispirato ai linguaggi di programmazione, utilizzato per rappresentare e presentare gli algoritmi in maniera compatta e chiara; ogni libro e programmatore definisce la propria specifica di Pseudocodice ma comunque quasi tutti si ispirano alla sintassi del Pascal,C oppure di Java;i costrutti definiti nel mio pseudocodice sono i seguenti:

Istruzione	Significato
$x = 5$	assegna ad x il valore 5
integer nome	definisce una variabile nome di tipo intero
real nome	definisce una variabile nome di tipo reale
boolean nome	definisce una variabile nome di tipo booleano
if condizione	definisce la struttura if
For estrInf **To** estrSup	definisce la struttura For da estrInf a estrSup
For estrSup **Downto** estrInf	definisce la struttura For da estrSup a estrInf
While condizione	Definisce l'istruzione While
Return valore	Ritorna valore da una procedura

1.2 Analisi di Algoritmi

In questo paragrafo verranno scritti degli algoritmi e verrà effettuata l'Analisi dell'Algoritmo, che può avvenire in due modalità:

- **Tempo di Esecuzione**:è il numero di operazioni primitive che vengono eseguite da parte di un algoritmo;l'esecuzione di un'istruzione si assume che richiede un tempo costante per evitare di rendere la valutazione dipendente dall'hardware e dalla bravura del programmatore.

- **Spazio di Esecuzione**:è il numero di spazio in bit occupato in memoria dall'algoritmo ma questa analisi non viene quasi mai eseguita in quanto oramai è superfluo.

Dato un Array di interi A[0....length-1] si definisce $\min(A) = a \Leftrightarrow \forall b \in A : a \leq b$ e il pseudocodice della procedura MIN è il seguente:

Algorithm 1 MIN(ITEM[] A,**integer** n)

		Costo	Volte
1	$min = A[0]$	c_1	1
2	**for** j = 1 **to** *A.length-1*	c_2	n
3	**if** $A[j] \leq min$	c_3	$n-1$
4	$min = A[j]$	c_4	$n-1$
5	**return** min	c_5	1

Per valutare un Algoritmo bisogna verificare due proprietà:**CORRETTEZZA** e **EFFICENZA**.
Per provare la correttezza di un algoritmo bisogna effettuare una dimostrazione matematica attraverso l'*invariante di ciclo*, ossia una proprietà che mostra la correttezza dell'algoritmo.
L'invariante deve essere vera in due casi specifici:

1. **passo base**:l'affermazione è vera all'inizio della prima iterazione del ciclo
2. **passo induttivo**:supposta vera all'inizio dell'iterazione deve essere vera anche all'inizio dell'iterazione successiva

Es: Verificare la correttezza di min(A) **Invariante di ciclo**:all'inizio di ogni iterazione la variabile *min* contiene il minimo parziale degli elementi A[0....j-1]

Dimostrazione. **Passo base**:per j = 1 l'array A è composto da un solo elemento
Ipotesi induttiva:*min* contiene il minimo degli elementi A[0...j-1] all'inizio dell'iterazione
Passo induttivo:all'esecuzione di un iterazione si possono verificare due casi:
Caso $A[j] < min$

la variabile *min* viene aggiornata con il valore A[j] e ciò verifica la proprietà
Caso $A[j] >= min$

la variabile *min* contiene già il minimo parziale degli elementi A[0...j] □

Per provare l'efficienza di un algoritmo bisogna calcolare e dimostrare il numero di confronti necessari,in funzione di n, per risolvere un problema computazionale come viene mostrato nell'esempio.
Es:Calcolo tempo di esecuzione algoritmo min(ITEM[] A,integer n)

Algorithm 2 MIN(ITEM[] A,**integer** n)

		Costo	Volte

1	$min = A[0]$	c_1		1
2	**for** j = 1 **to** *A.length-1* c_2		n	
3	**if** $A[j] \leq min$	c_3		$n-1$
4	$min = A[j]$	c_4		$n-1$
5	**return** min	c_5		1

Il costo dell'algoritmo MIN nel caso peggiore è $T(n) = (c_2 + c_3 + c_4)n + (c_1 - c_3 - c_4 + c_5)$.
L'algoritmo MIN nel caso peggiore è una funzione lineare.

1.3 InsertionSort

L'algoritmo INSERTION-SORT risolve il problema dell'ordinamento definito come: **Input**:una sequenza di n numeri $(a_1, a_2, \ldots, a_n)$
Output:una permutazione $(a'_1, a'_2, \ldots, a'_n)$ tale che $a'_1, a'_2, \ldots, a'_n$

INSERTION-SORT(A)

```
1  for j = 2 to A.length
2      key = A[j]
3      i = j − 1
4      while i > 0 and A[i] > idkey
5          A[i + 1] = A[i]
6          i = i − 1
7      A[i + 1] = key
```

L'algoritmo INSERTION-SORT è un algoritmo efficiente per ordinare un ristretto numero di elementi.
L'algoritmo INSERTION-SORT prende una carta alla volta ed effettua l'ordinamento cosicchè ogni volta che si ha un sequenza di oggetti la si ha ordinata ed aggiungendo un nuovo elemento si effettua ancora l'ordinamento.
Questo algoritmo INSERTION-SORT è la soluzione più naturale ed utilizzata dalle persone umane per effettuare l'ordinamento di una sequenza, ad esempio di Numeri e di Carte. Per poter affermare che l'algoritmo è corretto e risolve il problema dell'ordinamento bisogna dimostrare l'invariante del ciclo **For**.
Invariante di Ciclo: All'inizio di ogni iterazione del ciclo **for**, linee $1-8$, il sottoarray $A[1\,..\,j-1]$ è ordinato.
L'analisi del tempo di esecuzione dell'algoritmo INSERTION-SORT è il seguente:

INSERTION-SORT(A)

costo	numero di volte		
	1	**for** $j = 2$ **to** *A.length*	c_1
	2	$key = A[j]$	c_2
	3	$i = j - 1$	c_3
	4	**while** $i > 0$ and $A[i] > idkey$	c_4
	5	$A[i + 1] = A[i]$	c_5
	6	$i = i - 1$	c_6
	7	$A[i + 1] = key$	c_7

1	c_1	n	n
2	c_2	$n-1$	$n-1$
3	c_3	$n-1$	$n-1$
4	c_4	$\sum_{j=2}^{n} t_j$	$\sum_{j=2}^{n} t_j$
5	c_5	$\sum_{j=2}^{n} (t_j - 1)$	$\sum_{j=2}^{n} (t_j - 1)$
6	c_6	$\sum_{j=2}^{n} (t_j - 1)$	$\sum_{j=2}^{n} (t_j - 1)$
7	c_7	$n-1$	$n-1$

Il tempo di esecuzione dell'algoritmo è la somma dei tempi di esecuzione per ogni istruzione eseguita quindi il tempo di esecuzione di INSERTION-SORT è:

$$T(n) = c_1n+c_2(n-1)+c_3(n-1)+c_4\sum_{j=2}^{n} t_j+c_5\sum_{j=2}^{n}(t_j-1)+c_6\sum_{j=2}^{n}(t_j-1)+c_7(n-1)$$

In caso l'algoritmo sia già ordinato, caso migliore, si avrebbe sempre $A[i] < key$ quindi t_j è sempre 1, per cui il tempo di esecuzione sarebbe:

$$T(n) = c_1n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1) \tag{1.1}$$
$$= (c_1 + c_2 + c_3 + c_4 + c_5)n - (c_2 + c_3 + c_4 + c_7) \tag{1.2}$$
$$\tag{1.3}$$

In caso si abbia una sequenza decrescente, caso peggiore, nel ciclo While bisogna confrontare ogni elemento $A[j]$ con il sottoarray $A[1\,..\,j-1]$ per cui $t_j = j$ per $j = 2, 3, \ldots, n$ e poiche si ha

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1\sum_{j=2}^{n}(j-1) = \frac{n(n-1)}{2} \tag{1.4}$$

il tempo dell'algoritmo INSERTION-SORT nel caso peggiore è il seguente:

$$T(n) = c_1n + c_2(n-1) + c_3(n-1) + c_4(\frac{n(n+1)}{2} - 1) + c_5(\frac{n(n-1)}{2}) + c_6(\frac{n(n-1)}{2}) + c_7(n-1) \tag{1.5}$$
$$= c_1n + c_2(n-1) + c_3(n-1) + c_4(\frac{n^2+n-2}{2}) + c_5(\frac{n^2-n}{2}) + c_6(\frac{n^2-n}{2}) + c_7(n-1) \tag{1.6}$$
$$= (c_4/2 + c_5/2 + c_6/2)n^2 + (c_1 + c_2 + c_3 + c_4/2 - c_5/2 - c_6/2 + c_7)n - (c_2 + c_3 + c_4 + c_7) \tag{1.7}$$

Il tempo dell'algoritmo può essere scritto come $an^2 + bn + c$ che è una funzione quadratica che viene indicata, nel caso peggiore come $\omega(n^2)$.

1.4 Ricerca di Valori

Input:una sequenza di valori A[0 . . . length-1] e un valore *key*
Output:*index*,indice della sequenza A in cui A[index] = key, altrimenti null

Capitolo 2

Crescita delle Funzioni

Nel valutare l'algoritmo MIN si ottiene una funzione $T(n) = an + b$ che è una funzione lineare.
Durante la valutazione di un algoritmo difficilmente si riesce a quantificare con esattezza le costanti coinvolte per cui si analizza il comportamento della funzione al tendere di n all'infinito.
Al tal fine si utilizzano le notazioni $O,\ \Omega,\ \Theta$ definite come segue:

- $f(n) \in O(g(n)) \Leftrightarrow\ \exists c \geq 0\ \exists m \geq 0 : f(n) \leq cg(n)\ \forall n \geq m$
- $f(n) \in \Omega(g(n)) \Leftrightarrow \exists c, m \geq 0 : f(n) \geq cg(n) \forall n \geq m$
- $f(n) \in \Theta(g(n)) \Leftrightarrow \exists c_1, c_2, m \geq 0 : c_1 g(n) \leq f(n) \leq cg(n)\ \forall n \geq m$

Per maggiore chiarezza si utilizza un'abuso di linguaggio scrivendo $f(n) = O(g(n))$ al posto di $f(n) \in O(g(n))$
Per poter definire se una funzione appartiene a una notazione bisogna mostrarlo attraverso una dimostrazione formale con l'utilizzo dell'induzione matematica.

Esempio: la funzione $f(n) = 4n^2 + 4n - 1 \in O(n^2)$ va dimostrata attraverso induzione
Caso Base: $n = 1$
Per $n = 1$ si ha $4 + 4 - 1 \leq c * 1$ che è vera per $0 \leq c \leq 7$

Ipotesi Induttiva:

2.1 Proprietà Notazioni O

La notazione O possiede le seguenti proprietà:

1. Riflessività: $\forall c\ cf(n) = O(f(n))$(lo stesso vale per $\Theta ed \Omega$)
2. Transività:$f(n) = O(g(n)) \wedge g(n) = O(h(n)) \Leftarrow f(n) = O(h(n))$
3. Simmetrià trasposta:$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega f(n)$
4. Somma: $f(n) + g(n) = O(\max\{f(n), g(n)\})$
5. Prodotto: $f(n) = O(h(n)) \wedge g(n) = O(q(n)) \Leftrightarrow f(n)g(n) = O(h(n)q(n))$

2.2 Proprietà Notazione Ω

La notazione Ω possiede le seguenti proprietà:

1. Riflessività: $\forall c\ cf(n) = \Omega(f(n))$
2. Transività: $f(n) = \Omega(g(n)) \wedge g(n) = \Omega(h(n)) \Leftarrow f(n) = \Omega(h(n))$
3. Simmetrià trasposta: $f(n) = \Omega g(n) \Leftrightarrow g(n) = O(f(n))$
4. Somma: $f(n) + g(n) = \Omega(\max\{f(n), g(n))\})$
5. Prodotto: $f(n) = \Omega(h(n)) \wedge g(n) = \Omega(q(n)) \Leftrightarrow f(n)g(n) = \Omega(h(n)q(n))$

2.3 Proprietà Notazione Θ

La notazione Θ possiede le seguenti proprietà:

1. Riflessività: $\forall c\ cf(n) = \Theta(f(n))$
2. Transività: $f(n) = \Theta(g(n)) \wedge g(n) = \Theta(h(n)) \Leftarrow f(n) = \Theta(h(n))$
3. Simmetria: $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$
4. Somma: $f(n) + g(n) = \Theta(\max\{f(n), g(n))\})$
5. Prodotto: $f(n) = \Theta(h(n)) \wedge g(n) = \Theta(q(n)) \Leftrightarrow f(n)g(n) = \Theta(h(n)q(n))$

Θ implementa una relazione di equivalenza sulle funzioni in quanto sono rispettate la Riflessività,Transività e la Simmetria.

Capitolo 3

Divide et Impera

Come già visto nel precedente capitolo, *divide et Impera* è una tecnica di sviluppo di algoritmi ricorsivi, in cui si divide il problema in sottoproblemi più semplici da risolvere.
Per risolvere l'equazione di ricorrenza, funzione che descrive il tempo di esecuzione in funzione del tempo di esecuzione dei sottoproblemi, vi sono 3 metodi:

Metodo di Sostituzione :ipotiziamo un tempo di esecuzione e utilizziamo l'induzione matematica per dimostrare la correttezza dell'ipotesi

Metodo dell'albero di Ricorsione :converte l'equazione in un albero i cui nodi rappresentano i costi ai vari livelli della ricorsione

Metodo dell'Esperto :fornisce i limiti dell'equazioni di ricorrenza che rispettano determinate condizioni(Analizzato nei prossimi paragrafi)

3.1 Il metodo di Sostituzione

Il metodo di sostituzione è un tecnica di risoluzione delle equazioni di ricorrenza degli algoritmi divide et impera, che richiede due passi:

1. Ipotizzare la forma della risoluzione.
2. Usare l'induzione matematica per trovare le costanti e dimostrare che la soluzione proposta funziona ed è corretta.

3.2 Albero di Ricorsione

3.3 Teorema dell'Esperto