Paolino Di Felice

Quaderno di Fondamenti di informatica II

www.aracne-editrice.it
info@aracne-editrice.it

via Raffaele Garofalo, 133 A/B
00173 Roma
redazione: (06) 72672222 – telefax 72672233
amministrazione: (06) 93781065

ISBN 88-7999-825-0

I edizione: novembre 2004

Indice del volume

Presentazione

Parte I: Requisiti dei programmi

1. correttezza
2. efficienza temporale
   - modello dei costi
   - t(n), O(g(n))
   - analisi dei programmi iterativi
     - metodo del conteggio delle frequenze
     - regola della somma e del prodotto
   - analisi dei programmi ricorsivi
     - le equazioni di ricorrenza
3. efficienza spaziale
   - s(n), O_s(g(n))

Parte II: Tecniche di rappresentazione dei dati e strutture dati fondamentali

1. Liste
2. Pile
3. Code
4. Matrici sparse
5. Insiemi
6. Grafi
7. Alberi

Parte III: Algoritmi fondamentali

1. Operazioni sulle liste
2. Operazioni sulle matrici sparse
3. Operazioni sugli insiemi
4. Operazioni sui grafi
5. Operazioni sugli alberi binari e alberi binari di ricerca

MOTIVAZIONI ED OBIETTIVI DI QUESTE NOTE

Lo studio di buoni libri rimane un momento formativo unico ed insostituibile che ogni docente si sente in dovere di raccomandare ai suoi allievi. Purtroppo, con l'avvento del Nuovo Ordinamento degli studi (la cosiddetta laurea "3+2"), si assiste ad un fenomeno di larghissime proporzioni in cui i discenti studiano esclusivamente su "magri" appunti, talvolta persino non scritti di loro pugno. Le motivazioni da essi addotte sono molteplici, su tutte la seguente risulta essere prevalente: la *nuova* laurea ha reso molto stringenti le fasi di studio a carico degli allievi delle facoltà di Ingegneria esposti, in genere, allo stesso numero di esami del Vecchio Ordinamento, ma spalmati su di un triennio e non già sugli originali 5 anni.

La motivazione a scrivere queste note è la speranza personale di venire incontro alle esigenze degli studenti offrendo loro del materiale di studio che si colloca a metà strada tra i voluminosi testi di informatica disponibili sul mercato e i "magri" appunti di cui sopra. L'*Aracne editrice*, con la collona delle *microtirature*, ha offerto all'Autore un ulteriore motivazione mancante nei precedenti anni accademici: *il costo basso*. Mi sento in dovere di ringraziarla a nome dei miei allievi.

Questo volume raccoglie molta parte del materiale dell'insegnamento denominato *Fondamenti di Informatica II* (FI2) nel Manifesto degli Studi degli allievi del Corso di Laurea triennale in *Ingegneria Informatica ed Automatica*. L'obiettivo di FI2 è fornire un'ampia rassegna critica di *strutture dati* e *algoritmi* fondamentali nella soluzione automatica di problemi, unitamente ai metodi di *analisi* delle prestazioni *dei programmi* e loro sviluppo e *messa a punto*. Il linguaggio usato nella scrittura dei programmi è il C++ introdotto a Fondamenti d'Informatica I.

Il materiale qui raccolto è il punto di arrivo di tanti anni di insegnamento di FI2 e di altri analoghi previsti nel vecchio ordinamento tenuti dall'Autore presso la Facoltà di Ingegneria dell'Università degli Studi dell'Aquila, sin dal 1983.

Strutturalmente, il materiale è suddiviso in tre *parti* di seguito descritte in sufficiente dettaglio.

PARTE I - Requisiti dei programmi

Obiettivi:

Discussione dei principali requisiti che ogni buon programma deve possedere, quindi ci si sofferma su uno di essi: l'*efficienza spazio-temporale*, fornendo metodi generali per la sua valutazione tanto per i programmi iterativi che ricorsivi.

Contenuti:
Il metodo dei dati di prova, i livelli di correttezza. Analisi dei programmi: efficienza spaziale e temporale dei programmi. *L'efficienza temporale*: il modello dei costi per i costrutti presenti nei linguaggi ad alto livello (il caso del C++), la funzione costo temporale (t(n)), la dimensione dell'input (n), la configurazione dell'input (caso peggiore, medio e migliore). La complessità computazionale dei programmi: la notazione $O(g(n))$. Un metodo per la determinazione della t(n): *il conteggio delle frequenze*. Il passaggio dalla t(n) alla $O(g(n))$. Un teorema che governa tale passaggio nel caso di funzioni costo di tipo polinomiale. Il metodo per la determinazione diretta della $O(g(n))$ basato sulla *regola della somma e del prodotto*. Determinazione della t(n) nel caso di codici ricorsivi. *L'efficienza spaziale*: la funzione costo spaziale (s(n)) e la notazione $O_s(g(n))$. *Studio di casi*.

PARTE II – Tecniche di rappresentazione fondamentali dei dati e strutture dati
Obiettivi:
a) introduce le tecniche di rappresentazione, nella memoria centrale del calcolatore, delle principali strutture discrete che intervengono nella modellazione dei dati coinvolti nella soluzione automatica di problemi rilevanti; b) ne mostra la realizzazione (le *strutture dati* -- SD) in C++; c) analisi critica dell'occupazione di memoria causata dalle diverse soluzioni proposte.

Contenuti:

Liste: rappresentazione tramite array (r. sequenziale), r. tramite puntatore in avanti (r. collegata), r. tramite puntatore in avanti e indietro (r. collegata simmetrica).

Pile: rappresentazione tramite array (r. sequenziale), r. tramite puntatori (r. collegata).

Code: rappresentazione tramite array (r. sequenziale), r. tramite puntatori (r. collegata).

Matrici sparse: rappresentazione compatta a 3 informazioni (r.1) e r. tramite vettore di accesso (r.2) (r. sequenziali), r. compatta a 3 informazioni (r.3) e r. tramite vettore di accesso mediante l'uso di puntatori (r.4) (r. collegate), r. compatta tramite doppio vettore di accesso mediante l'uso di puntatori (r.5).

Insiemi: rappresentazione tramite array (r. sequenziale), r. tramite puntatori (r. collegata).

Grafi: rappresentazione tramite matrice delle adiacenze, r. compatta tramite vettore di accesso dinamico della matrice delle adiacenze (alias, liste dei successori).

Alberi: rappresentazione tramite puntatori (r. collegata) di alberi binari ed ennari.

PARTE III – Algoritmi fondamentali e loro codifica

Obiettivi:

a) passa in rassegna i principali algoritmi noti per la soluzione di problemi classici dell'informatica quali: la visita di grafi ed alberi, l'ordinamento di un insieme di valori, la ricerca di un elemento in un insieme di valori e/o in un albero binario di ricerca, la fusione di sequenze ordinate, l'unione di insiemi, ecc; b) ne discute la soluzione C++ ed i relativi costi computazionali.

Contenuti:

Operazioni sulle liste (creazione, aggiunta di un atomo, estrazione del primo atomo, test di appartenenza di un atomo ad una lista, stampa atomi, stampa bidirezionale atomi, concatenazione di due liste, fusione di due liste, conteggio atomi con medesimo valore su due liste distinte, inversione atomi lista).

Operazioni sulle pile (creazione, stampa atomi, conta atomi, aggiunta/ eliminazione atomo). Operazioni sulle code (creazione, stampa atomi, conta atomi, aggiunta/ eliminazione atomo, concatenazione di due code).

Operazioni sulle matrici sparse (leggi/stampa matrice, cerca massimo/ minimo valore, aggiorna valore, permuta righe, somma).

Operazioni sugli insiemi (creazione, stampa elementi, unione/differenza/ intersezione di due insiemi, ricerca esaustiva di un valore, ricerca binaria iterativa/ricorsiva di un valore, ordinamento dei valori di un insieme: algoritmo a bolle, per selezione e merge-sort).

Operazioni sui grafi (creazione e stampa di un grafo, aggiunta/eliminazione di un nodo/ramo, visita in profondità di un grafo, grado di ingresso/uscita di un nodo).

Operazioni sugli alberi binari e alberi binari di ricerca (visita in preordine, postordine e simmetrica, ricerca di un elemento, stampa in notazione parentetica dei nodi, cancella albero).

L'Autore Estate 2004

PARTE I

REQUISITI DEI PROGRAMMI

L'obiettivo di ogni progettista di programmi (brevemente P nel seguito delle presenti note) è scriverne di "buoni". Molto semplicemente si può affermare che P è buono se possiede *buoni* requisiti. Tra i numerosi requisiti meritano una menzione esplicita i seguenti: la *leggibilità*, la *correttezza*, la *robustezza* e l'*efficienza*. In questa *Prima Parte*, dopo aver accennato alla leggibilità, ci si sofferma in dettaglio crescente sugli altri tre. Si desidera sin da ora sottolineare che nell'economia del Corso di *Fondamenti di Informatica 2*, l'efficienza ha un ruolo di assoluto rilievo.

1. **LEGGIBILITÀ**

Per leggibilità di un P s'intende la sua intrinseca proprietà di agevolare la comprensione, in ogni momento, delle scelte progettuali (sia in termini di strutture dati che di algoritmo) che sono alla sua base.

Alcune regole per tenere alta la leggibilità dei programmi

1. Adottare una saggia articolazione di P in sottoprogrammi;
2. commentare adeguatamente ogni "sottoprogramma" (*cosa* fa e *come* lo fa);
3. indentare il codice;
4. usare identificatori auto-esplicativi.

2. **CORRETTEZZA**

Definizione P è corretto se aderisce alle specifiche di progetto.

Le specifiche di progetto di un P riguardano:

- i vincoli sui dati iniziali, ossia i dati leciti per la sua attivazione (*dati di input*),
- i vincoli sul risultato, ossia i risultati attesi al termine dell'esecuzione di P (*dati di output*).

Dunque:

- le specifiche di input caratterizzano l'insieme dei dati d'ingresso, mentre
- le specifiche di output caratterizzano l'insieme dei risultati possibili.

Per verificare la correttezza di un P esistono due percorsi alternativi: un *metodo teorico* e il *metodo* cosiddetto dei *dati di prova*.

METODO TEORICO

Esso consiste nel dimostrare che, a partire da dati di input leciti per P, l'esecuzione di P produce dati di output corretti. Causa la difficoltà e la lentezza intrinseca dell'approccio formale, nella realtà la prova di correttezza non viene effettuata, specie nel caso di P lunghi e complessi, ossia proprio in quelle situazioni ove maggiore è l'esigenza di controllare l'attendibilità del programma.

Un altro punto a sfavore del metodo teorico è il seguente. La definizione di correttezza di un programma non specifica quale debba essere il suo comportamento quando i dati d'ingresso su cui esso viene eseguito non

soddisfano le specifiche di input. Questa indeterminatezza non tiene conto del fatto che nella maggioranza delle applicazioni si richiede che il programma sia in grado di rilevare e trattare adeguatamente eventuali anomalie presenti nei dati d'ingresso. Infatti, se questo non si verifica, può accadere che il programma termini fornendo dei risultati che di fatto non sono significativi. E' a questo punto che interviene la nozione di *robustezza* di P, intesa come la capacità di P di operare in maniera prevedibile anche in quei casi in cui esso viene attivato con dati non conformi alle specifiche di input.

2.1 METODO DEI DATI DI PROVA (/TEST)

L'esito di una prova formale di correttezza (metodo teorico) può essere positivo o negativo. Se positivo, allora P è corretto, e se è negativo? Purtroppo il metodo formale (ed in generale tutti i metodi che poggiano su una dimostrazione) non ammette "mezze tinte", gli unici colori conosciuti sono: bianco (ossia, prova ok e quindi P ok), oppure nero (ossia, prova ko e quindi P ko).

Di seguito si presenta un approccio più pragmatico al problema della correttezza, il quale riunisce in sé le nozioni di correttezza e di robustezza dei programmi. Detto metodo persegue la correttezza del programma eseguendolo. I dati utilizzati nell'attivazione di P sono detti dati di prova (/di test), ed il metodo è detto metodo dei dati di prova (/test).

Il test di un programma consiste nei seguenti passi:

1. si procede alla determinazione di un insieme di dati di input I;
2. si esegue il programma con i dati in I;
3. si verificano i risultati ottenuti:
   3.1. se l'esecuzione del programma non termina in un lasso di tempo ragionevole o se i risultati ottenuti dalla sua esecuzione non sono quelli attesi, allora il programma non è corretto.

Si noti che, generalmente, il test di un programma non permette di stabilirne la correttezza, a meno che non vengano provate tutte le possibili istanze dei dati d'ingresso. Infatti, se eseguiamo il programma con un sottoinsieme dei possibili input non possiamo escludere che esista un insieme di dati, che non è stato provato, che ne evidenzi errori. Sfortunatamente la prova di un programma con tutti i possibili dati d'ingresso è praticamente impossibile nella maggioranza dei casi reali.

Si faccia ad esempio riferimento ad una funzione che calcola il prodotto di due interi compresi tra -100000000 e +100000000: