Appunti di Algoritmi e strutture dati

Riccardo Lo Iacono

Dipartimento di Matematica & Informatica
Università degli studi di Palermo
Sicilia
a.a. 2022-2023

Indice.

– 1 – Introduzione.

Dato un problema, è importante chiedersi come trovare una soluzione allo stesso. Inoltre, supposto che esista una soluzione algoritmica A, è opportuno poter confrontarla con una soluzione B.

– 1.1 – Algoritmi e complessità.

La valutazione di algoritmi può essere effettuata secondo diversi criteri. In generale, di interesse sono la velocità di crescita in termini di *spazio* e *tempo*.

In generale, il tempo necessario ad un algoritmo, espresso come funzione della taglia del problema, è detta *complessità di tempo*. Dicasi *complessità asintotica di tempo* il comportamento limite della complessità di tempo, al crescere della taglia[1] del problema.

– 1.2 – Notazioni asintotiche.

Le diverse complessità di un'algoritmo possono essere studiate secondo tre aspetti: *caso ottimo, caso pessimo, caso medio*.

– 1.2.1 – Caso ottimo: notazione Omega.

La notazione Omega Ω definisce un limite inferiore ad un funzione $f(n)$. In generale, data una certa funzione $g(n)$ si definisce $\Omega(g(n))$ come segue.

$$\Omega(g(n)) = \{f(n) : \exists c \in \mathbb{R}, n_0 \in \mathbb{N}, c, n > 0 \mid 0 \leq c \cdot g(n) \leq f(n), \forall n \geq n_0\}$$

– 1.2.2 – Caso pessimo: notazione O-grande.

La notazione O-grande definisce un limite superiore ad una funzione $f(n)$. In generale, data una certa funzione $g(n)$ si definisce $\mathcal{O}(g(n))$ come segue.

$$\mathcal{O}(g(n)) = \{f(n) : \exists c \in \mathbb{R}, n_0 \in \mathbb{N}, c, n_0 > 0 \mid f(n) \leq c \cdot g(n), \forall n \geq n_0\}$$

– 1.2.3 – Caso medio: notazione Theta.

La notazione Theta Θ definisce dei limiti ad una funzione $f(n)$. In generale, data una certa funzione $g(n)$ si definisce $\Theta(g(n))$ come segue.

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2 \in \mathbb{R}, n_0 \in \mathbb{N} \mid c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n), \forall n \geq n_0\}$$

[1] Indica la misura della quantità di dati in input.

– 2 – Modelli di computazione.

Prima di poter parlare di algoritmi, è necessario stabile il *modello di calcolo* con cui si intende risolvere un problema; si pensi ad un modello di calcolo come un modello che descrive come l'output sia ottenuto in funzione dell'input.

Nella presente sezione si descriveranno due dei modelli più diffusi quali *modello RAM* e *macchina di Turing*.

– 2.1 – Modello RAM.

Il modello RAM[2], modella un computer in cui le istruzioni non modificano se stesse. Procedendo all'analisi strutturale di una RAM questa si compone di due nastri: uno di input, uno di output, ciascuno con la propria testina; un programma e una memoria.

Il nastro di input è rappresentato come una sequenza di celle, ciascuno contente un intero: ogni volta che un'istruzione di lettura è eseguita la testina è spostata di un posto verso destra.

Il nastro di output ha una struttura analoga a quello di input, con la differenza che inizialmente è completamente vouto. Quando un'istruzione di scrittura è eseguita sulla cella puntata dalla testina è impresso un intero, successivamente la testina si sposta di un cella a destra.

La memoria è un insieme di registri $r_0, r_1, \ldots, r_i, \ldots$, ciascuno con la capacità di memorizzare un intero di taglia arbitraria.

Il programma, non risiedente in memoria, per cui si assume non auto-modificante, è una mera sequenza di istruzioni[3], opzionalmente, etichettate.

Nota: Tutte le computazioni avvengo in r_0, l'*accumulatore*.

[2] La sigla RAM sta ad indicare *Random Access Machine*.

[3] La natura delle istruzioni è trascurabile fintanto che queste somiglino a istruzioni macchina reali.

– 2.1.1 – Istruzioni e analisi di un programma RAM.

Le istruzioni fondamentali di un programma RAM sono quelle di *Figura 1.* A queste è possibile aggiungere ulteriori istruzioni, sempre con la condizione che questi siano simili ad istruzioni reali. Ciascun'istruzione si compone di due parti un *codice* e un *indirizzo*.

Codice	Indirizzo
LOAD	operando
STORE	operando
ADD	operando
SUB	operando
MULT	operando
DIV	operando
READ	operando
WRITE	operando
JUMP	label
JZERO	label
JGTZ	label
HALT	

Figura 1: Tabella delle istruzioni RAM

Gli operandi sono di tre tipi:

1. $=i$: indica l'intero stesso.
2. i: un intero non negativo che indica il contenuto dell'i-esimo registro.
3. $*i$: indica un indirizzamento indiretto. Cioè i è il contenuto del registro j. Se $j < 0$ la macchina si arresta.

Considerando un programma P, il suo significato è definibile tramite due quantità: una funzione $c : \mathbb{Z} \to \mathbb{N}$ e un *contatore di posizione* nel programma.
Inizialmente $c(i) = 0, \forall i \geq 0$ e il contatore di posizione è posto alla prima istruzione del programma. Ai fini di comprendere il significato di un'istruzione, si definisca $v(a)$, definita come segue, il valore dell'operando a.

$$v(=i) = i$$
$$v(i) = c(i)$$
$$v(*i) = c(c(i))$$

In generale, si può pensare ad un programma RAM come una "mappatura" dei dati di in input in quelli di output. Sebbene esistano varie interpretazioni di tale mappatura, due delle più importanti sono le interpretazioni di analizzatore di funzioni e di linguaggi.

Procedendo all'analizzare le due rappresentazioni prima citate, si ha quanto segue.

1. Sia P un programma che legge sempre n interi dal nastro di input, e che al più scriva un carattere su quello di output. Se $x_1, x_2, \ldots, x_n$ sono gli interi nelle prime n celle del nastro di input, P scrive y nella prima cella del nastro di output, e fatto ciò si arresta; si dirà che P computa una funzione $f(x_1, x_2, \ldots, x_n) = y$.

2. Sia Σ un certo alfabeto; rappresentando con $1, \ldots, k$ i simboli dell'alfabeto, una RAM accertatrice di linguaggi si comporta come segue. Se $s = a_1 a_2 \ldots a_n$ è una stringa, si pone s sul nastro di input e nella (n + 1)-esima cella si pone un carattere terminale. Si dirà che un programma P accetta s se, questi legge l'intera stringa e il carattere terminale, scrive sul nastro di output 1 e si arresta. Con tale definizione, un linguaggio accettato è l'insieme delle stringhe accettate.

Esempio: Sia $f(n)$ la funzione di seguito definita, scrivere un programma RAM che la calcoli.

$$f(n) = \begin{cases} n^n, & \text{se } n \geq 1 \\ 0, & \text{altrimenti} \end{cases}$$

In *Figura 2* rispettivamente l'implementazione algoritmica e quella in RAM.

```
begin
    read r1
    if r1 ≤ 0 then write 0;
    else
        begin
            r2 ← r1;
            r3 ← r1 − 1;
            while r3 > 0 do
                begin
                    r2 ← r2 * r1;
                    r3 ← r3 − 1;
                end
            write r2;
        end
end
```

(a) Algoritmo per n^n

```
        READ 1
        LOAD 1
        JGTZ pos
        WRITE =0
        JUMP enfif
pos:    LOAD 1
        STORE 2
        LOAD 1
        SUB =1
        STORE 3
while:  LOAD 3
        SUB =1
        JGTZ cont
        JUMP endw
cont:   LOAD 2
        MULT 1
        STORE 2
        LOAD 3
        SUB =1
        STORE 3
        JUMP while
endw:   WRITE 2
endif:  HALT
```

(b) Programma ram per n^n

Figura 2: implementazioni di $f(n)$

– 2.2 – Complessità di un programma RAM.

Il modello RAM sinora descritto è definito come *modello RAM con costo uniforme*: in questo caso ogni istruzione richiede un'unità di tempo e ogni registro un'unità di spazio. Un'ulteriore modello è il *modello RAM con costo logaritmico*, analizzato nel paragrafo seguente.

– 2.2.1 – RAM con costo logaritmico.

Nel modello con costo logaritmico si tiene conto della dimensione finita della memoria, nella fattispecie della dimensione limitata di una WORD. In tal senso, sia $l(i)$ la seguente funzione che stabilisce il numero di bit per rappresentare l'operando

$$l(i) = \begin{cases} \lfloor \log |i| \rfloor + 1, & \text{se } i \neq 0 \\ 1, & \text{se } i = 0 \end{cases}$$

Si ha quindi che il costo per accedere agli operandi è quanto segue

$$\begin{aligned} =i: &\quad l(i) \\ i: &\quad l(i) + l(c(i)) \\ *i: &\quad l(i) + l(c(i)) + l(c(c(i))) \end{aligned}$$

Analizzando, a titolo di esempio, l'istruzione **ADD** *i, da quanto sopra si necessita $l(i) + l(c(i)) + l(c(c(i)))$ per accedere ad *i, a cui aggiungere il costo di accesso all'accumulatore pari a $l(c(0))$.

– 2.3 – La macchina di Turing.

Prima di discutere la macchina di Turing, è necessario parlare di *relazione polinomiale*

Definizione: Due funzioni $f_1(n)$ e $f_2(n)$ sono in relazione polinomiale se esistono $p_1(x)$ e $p_2(x)$ tali da soddisfare la seguente relazione.

$$f_1(n) \leq p_1(f_2(n)) \wedge f_2 \leq p_2(f_1(n)) \qquad ,\forall n$$

Esempio: Siano $f_1(n) = 2n^2$ e $f_2(n) = n^5$: si osserva che queste sono in relazione polinomiale. Infatti se $p_1(x) = 2x$ e $p_2(x) = x^3$ segue: $2n^2 \leq 2n^5$ e $n^5 \leq (2n^2)^3$

Si considera ora un nuovo modello di calcolo, la *macchina di Turing*.

Definizione: Una macchina di Turing (MT) multi-nastro si compone di k nastri, ciascuno dei quali è diviso in celle, ciascuna delle quali contiene un simbolo di nastro. Le operazioni sono dettate da un programma primitivo:il *controllo finito*.

In una computazione, in accordo col controllo finito e il carattere puntato dalla testina di ciascun nastro, una MT può effettuare almeno un delle seguenti operazioni.

1. Cambiare lo stato del controllo finito.
2. Sovrascrivere uno o più caratteri nelle celle indicate dalle testine.
3. Per ciascun nastro, in maniera indipendente, spostare la testina verso destra, sinistra o lasciarla li.

Formalmente una generica MT viene identificata tramite la settupla[4]

$$(Q, T, I, \delta, b, q_0, q_f)$$

L'attività di una MT può formalmente essere descritta tramite istantanee (ID). Quest'ultime sono k-ple $\alpha_1, \ldots, \alpha_k$, con α_i una stringa del tipo xqy tale che, xy sia la stringa dell'i-esimo nastro esclusi i black iniziali e finali, con q lo stato della MT.

[4] Per il significato di ciascun componente si veda "Appunti di Informatica Teorica".

– 2.4 – Relazione tra MT e RAM.

Principale applicazione delle MT è quello di determinare un *lower bound* al tempo e allo spazio necessari alla risoluzione di un problema.

Considerando la relazione tra MT e RAM, risulta ovvio che una RAM possa simulare una MT a k nastri, semplicemente mantenendo un cella del nastro della MT in un registro. Si supponga una MT con complessità di tempo $T(n) \geq n$; ne segue che una RAM legga gli input in $\mathcal{O}(T(n))$ se a costo uniforme, in $\mathcal{O}(T(n)\ln T(n))$ se con costo logaritmico. In ambo i casi, il tempo di una RAM è limitato superiormente dal tempo della MT. Un risultato inverso, cioè la possibilità che una MT simuli un RAM, vale solo se la RAM è con costo logaritmico. In tal caso vale il teorema a seguire.

Teorema 2.1.
Sia L un linguaggio accettato da una RAM con costo logaritmico, in tempo $T(n)$. Se la RAM non fa uso di istruzioni **MULT** *e/o* **DIV**, *allora la complessità di tempo è al più $\mathcal{O}\left(T^2(n)\right)$.*

Dimostrazione: Sia rappresentato ciascun registro della RAM, non contenente zeri, come in *Figura 3*.

#	#	i_1	#	c_1	#	#	⋯	#	#	i_k	#	c_k	#	#	⋯

Figura 3: MT simulante RAM con costo logaritmico.

Il nastro è dunque una sequenza di coppie (i_j, c_j), scritte in binario, separate da un delimitatore. Il contenuto dell'accumulatore è memorizzato su di un secondo nastro, e un terzo nastro è usato come supporto. In aggiunta a questi vi sono due nastri aggiuntivi: uno per gli input e uno per gli output della RAM. Segue che un passo della RAM è rappresentato da un insieme finito di passi della MT.

Si procede ora col dimostrare che una RAM con costo computazionale k richiede al più $\mathcal{O}\left(k^2\right)$ passi di una MT. Il costo per memorizzare c_j in i_j è $l(c_j) + l(i_j)$, da cui si conclude che la lunghezza di caratteri non blank è $\mathcal{O}(k)$.
La simulazioni di istruzioni diverse da **STORE** sono $\mathcal{O}(k)$, poiché il costo principale è dato dalla ricerca nel nastro. Analogamente il costo di uno **STORE** è dato dal tempo di ricerca nel nastro, in aggiunta al tempo necessario per copiarlo entrambe $\mathcal{O}(k)$. Da cio si deduce che, a meno di **MULT** e/o **DIV**, un'istruzione RAM può essere simulata in $\mathcal{O}(k)$ passi della MT. Infine poiché sotto il criterio logaritmico ciascuna istruzione RAM costa almeno un'unita di tempo, il costo totale speso dalla MT è $\mathcal{O}\left(k^2\right)$

– 3 – Design di algoritmi efficienti.

La costuzione di algoritmi efficiente è generalmente effettuata sfruttando apposite tecniche di programmazione, e di diverse strutture dati.

– 3.1 – Strutture dati.

Nota: In questa sezione saranno discusse solamente strutture elementari quali code, dizionari, pile, grafi e alberi, per le strutture avanzate si rimanda alle

– 3.1.1 – Code.

Una coda è una sequenza S di n elementi. Operazioni consentite sulle code sono le seguenti.

- ISEMPTY(): verifica se S è vuota o meno.
- ENQUEUE(elem e): aggiunge e come ultimo elemento.
- DEQUEUE(): toglie da S il primo elemento e lo restituisce.
- FIRST(): restituisce il primo elemento senza toglierlo