Appunti senza pretese di P2+Lab: **Molto, poco, tantissimo, abbastanza, di piú**

Alessandro Panconesi
DSI, La Sapienza
via Salaria 113, piano terzo
00198 Roma

Per poter effettuare l'analisi degli algoritmi é necessario sviluppare una buona intuizione degli ordini di grandezza di alcune funzioni. In queste dispense cercheremo di iniziare a fare ció per le funzioni che piú sovente si incontrano nell'analisi degli algoritmi.

1 Esponenziale

Iniziamo con la funzione esponenziale la quale cresce molto rapidamente

Fatto 1 *Se ripiego un foglio di giornale 50 volte ottengo una pila di carta cosí alta da arrivare da Roma sino a New York.*

Dimostrazione: Un foglio di giornale é spesso (a occhio) $0.1mm = 10^{-4}m$. Piegando un foglio una volta lo spessore raddoppia, piegando ancora esso quadruplica, e cosí via. Piegando n volte lo spessore diventa

$$2^n.$$

Cinquanta piegamenti corrispondono ad $n = 50$ per cui, ricordando che $2^{10} = 1024 > 10^3$,

$$\begin{aligned}
\text{spessore-pila-di-carta} &= 2^{50} \times \text{spessore-foglio-di-giornale} \\
&= 2^{50}\ 10^{-4}\ m \\
&= (2^{10})^5\ 10^{-4}\ m \\
&\geq (10^3)^5 10^{-4}\ m \\
&= 10^{15}\ 10^{-4}\ m \\
&= 10^{11}\ m \\
&= 10^8\ km \\
&= \textit{100 milioni di kilometri!}
\end{aligned}$$

Piú che abbondante quindi per andare da Roma a NYC. $\triangledown$

Corollario 1 *Se piego un'altra volta ottengo una pila di carta che arriva alla luna e oltre.*

Messaggio analogo ci viene da una storiella di presunta provenienza orientale. L'imperatore del Rajasthan, il magnifico *Jahanpanah Shehenshah Alamgir Jahangir Mughal-e-Azam*[1] Akbar é afflitto da una misteriosa forma di *spleen*. Un asceta locale riesce a curarlo e come

[1] Questi sono tutti titoli onorifici

ricompensa Akbar gli promette di poter esaudire un qualunque suo desiderio. Per nulla impressionato l'asceta decide di dare una lezione di umiltá all'imperatore e chiede che venga portata una scacchiera indiana, di 14×14 caselle. *"Sublime Akbar, desidero che tu metta un chicco di riso sulla prima casella, due chicchi sulla seconda, quattro sulla terza, e cosí via sempre raddoppiando".*

Fatto 2 *L'intera produzione mondiale di riso degli ultimi 100 anni non é sufficiente a soddisfare la richiesta dell'asceta.*

Dimostrazione: Un chicco di riso pesa (a occhio) almeno un centesimo di grammo, cioé $10^{-5} kg$. Il numero totale di chicchi richiesti dall'asceta é

$$1 + 2 + 4 + 8 + 16 + \ldots + 2^{195} = \sum_{i=0}^{195} 2^{-i} = 2^{196} - 1 \approx 2^{196}$$

per cui, ricordando che $2^{10} = 1024 > 10^3$, il numero di kili di riso ammonterebbero a

$$2^{196} 10^{-5} kg > (10^3)^{19.6} 10^{-5} kg = 10^{58.8} 10^{-5} kg > 10^{53} kg$$

La massa dell'universo é stimata essere $10^{55} kg$. L'universo contiene senz'altro almeno cento galassie per cui... $\bigtriangledown$

2 Logaritmo

Prima di procedere é bene che il lettore si rinfreschi la definizione del logaritmo di una funzione. Com'é noto

$$a^b = n \Leftrightarrow \log_a n = b.$$

Dato che il logaritmo é l'inversa della funzione esponenziale esso *cresce molto lentamente.*

2.1 Liberismo überalles

É noto come in Italia la pressione fiscale sia enorme. Un noto imprenditore sta valutando se emigrare all'estero. La sua scelta ricade su due paesi:

- In Logaritmia le tasse sono il logaritmo in base 10 dello stipendio
- In Percentualia esse sono lo 0.1 %, ovvero un millesimo dello stipendio (In Italia mi sa che, contributi compresi, si aggirano intorno al 50 %)

Lo stipendio mensile del noto imprenditore ammonta a 10 miliardi di lire; dove gli conviene andare?

Fatto 3 *Logaritmia forever!*

Dimostrazione: In Percentualia le tasse da pagare ammontano a Lit. $10^{10} 10^{-3} = 10$ milioni. In Logaritmia invece

$$\log_{10} 10^{10} = 10 \text{ Lire!}$$

$\bigtriangledown$

Esercizio 1 *In Logaritmia vengono applicati due tassi, uno pari al logaritmo in base 2, l'altro pari al logaritmo in base 10. Quale conviene di piú?*

Esercizio 2 *In Logaritmia ci sono due opzioni per pagare le tasse: (a) una volta al mese, ogni mese, oppure (b) una volta sola a fine anno con un'aliquota pari al logaritmo del totale guadagnato in un anno. Quale conviene di piú?*

3 Radice

Consideriamo il seguente problema. Vogliamo scrivere un programma per determinare se un numero é primo oppure no, ovvero un *test di primalitá*. L'algoritmo banale che deriva direttamente dalla definizione di numero primo é ovviamente il seguente (l'istruzione modulo, `n mod i`, restituisce il resto della divisione intera tra n ed i):

```
function isPrime (n: integer): boolean;
var i:  integer;
var prime:  boolean;
begin
prime := true;
for i := 2 to n-1 do
  if (n mod i) = 0 then prime := false;
if prime then writln("Prime!") else writeln("Composite!");
end
```

Notare che il numero di operazioni necessarie a questo programma é proporzionale ad n.

Esercizio 3 *Riscrivete il programam usando un ciclo* `while` *che termini non appena si é determinato se il numero é primo oppure no. In quale caso questa implementazione impiegherá il tempo massimo?*

Vediamo di valutare il tempo di calcolo dell'algoritmo per interi abbastanza grandi. Supponiamo di eseguirlo su un calcolatore in grado di compiere circa un miliardo di operazioni al secondo (un miliardo $= 10^9$); una stima ottimistica in quanto un processore all'avanguardia come il Pentium é, anche in questo caso molto ottimisticamente, in grado di effettuare 10^8 operazioni al secondo. Se diamo in input al programma un numero di 11 cifre tipo

$$21.032.987.731 \approx 21 \times 10^9$$

il tempo di esecuzione sarádell'ordine dei secondi. Quando l'input é un numero di 15 cifre tipo

$$m = 176.897.547.951.007 \approx 1.76 \times 10^{15}$$

il numero di operazioni sará dell'ordine di

$$10^{15} = 10^9 \times 10^6 = 10^6 \ \text{ secondi}$$

vale a dire un milione di secondi. Orbene, in un giorno ci sono

$$24 \times 60 \times 60 = 86.400 \leq 90.000 \ \text{ secondi,}$$

per cui il tempo di esecuzione é

$$\frac{10^6}{9 \times 10^4} > 11 \ \text{ giorni!}$$

Se l'input fosse un numero di 20 cifre, con un calcolo analogo si arriverebbe alla conclusione che il tempo di esecuzione sarebbe superiore a

$$(11 \text{ giorni}) \times 10^5 > 10^6 \text{ giorni} \approx 3000 \text{ anni!}$$

(1000 giorni sono poco meno di 3 anni). Quindi, l'algoritmo proposto é del tutto inutilizzabile giá per numeri di 20 cifre. Si consideri che per le applicazioni crittografiche sono comuni numeri di centinaia di cifre binarie.

Esercizio 4 *Dimostrare che per numeri dell'ordine di 2^{150}, comunemente usati nelle applicazioni crittografiche, la funzione* `isPrime` *impiegherebbe, su un processore Pentium (10^8 operazioni al secondo) un tempo superiore a l'etá dell'universo, stimata a 5×10^{17} anni.*

Ma anche per numeri di 15 cifre l'algoritmo é praticamente inutile. Ora é possibile modificare quel semplice algoritmo per ottenerne uno che per numeri di 20 e piú cifre sia molto piú efficiente. Un miglioramento che viene subito in mente é di limitarsi a controllare la divisibilitá solamente per i numeri dispari (se n non é divisibile per 2 non lo sará anche per tutti i numeri pari). Questo dimezza all'incirca il tempo di calcolo ma é chiaro che é ancora troppo poco; per numeri di 15 cifre il tempo di esecuzione scenderebbe a circa 5 giorni mentre per quelli di 20 rimarrebbe pur sempre di 1500 anni!

Anche tenendo conto dei multipli di 3, i multipli di 5, ecc. non si otterebbero grandi miglioramenti. Ci vuole qualcosa di piú.. radicale! Se si analizza meglio il concetto di numero primo e quello complementare di numero composto si puó arrivare a scoprire qualcosa di interessante. Se n é composto questo significa che $n = m \times k$ per due interi m e k. Consideriamo il piú piccolo dei due e supponiamo che sia m. Cosí $m \leq k$, ma allora $m \times m \leq m \times k = n$, quindi m é minore od uguale alla radice quadrata di n. Quindi arriviamo ad una conclusione semplice ma utile:

Fatto 4 *n é composto se e solo se n ha un divisore $k \leq \sqrt{n}$.*

Abbiamo cosí trovato un modo di ridurre le operazioni svolte dal nostro algoritmo da ordine di n a ordine di $\sqrt{n}$:

```
function isPrime (n: integer): boolean;
var i:  integer;
var prime:  boolean;
begin
prime := true;
for i := 2 to sqrt(n) do
  if (n mod i) = 0 then prime := false;
if prime then writln("Prime!") else writeln("Composite!");
end
```

Questa semplice, ma non completamente banale, modifica ha delle ripercussioni notevoli sull'efficienza dell'algoritmo. Ad esempio, su interi di 15 cifre il nuovo algoritmo eseguirá all'incirca

$$\sqrt{10^{15}} < 10^8 \text{operazioni}$$

che con il nostro microprocessore da 10^9 operazioni al secondo significa circa un decimo di secondo. Vale a dire siamo passati da un tempo dell'ordine dei giorni ad uno dell'ordine dei millesimi di secondo! Per interi di 20 cifre il numero di operazioni é all'incirca

$$\sqrt{10^{20}} = 10^{10},$$

che in termini di tempo equivale a circa 10 secondi; da migliaia d'anni siamo quindi passati a pochi secondi.

Ma purtroppo anche l'algoritmo migliorato mostra la corda abbastanza presto. Per interi di 40 cifre ritorniamo a tempi di esecuzione dell'ordine delle migliaia d'anni. Per risolvere il problema in modo definitivo occorrono metodi ben piú sofisticati. É comunque chiaro che

- Investire in "matematica" puó comportare risparmi di gran lung asuperiori a quelli ottenibili investendo in hardware piuú costoso e sofisticato;

- un tempo di risposta tipo radice quadrata é di molto preferibile ad un tempo di risposta *lineare* (cioé proporzionale ad n).

4 Crescita Lineare vs. Crescita Logaritmica

Nel paragrafo precedente abbiamo visto come la scelta dell'algoritmo puó avere ripercussioni importantissime sulla qualitá della soluzione. Adesso vedremo come lo stesso valga per la scelta delle strutture dati.

Una casa editrice sta per pubblicare una nuova enciclopedia in **20 volumi**. Ciascun volume contiene **1000 pagine** mentre ciascuna pagina in media contiene **1000 parole**. Prima della pubblicazione si vuole procedere ad un esame di correttezza ortografica del testo (spell checking) tramite computer. Sia il testo dell'enciclopedia che il vocabolario sono memorizzati su computer. Vengono proposti due metodi.

- Ricerca Lineare. Dato che "ormai i computer sono velocissimi", per ogni parola del testo, basta scandire sequenzialmente il vocabolario.

- Ricerca Binaria. Memore degli insegnamenti di TAMC, un laureato dell'Universitá di Bologna (Corso di Laurea di Cesena) propone invece di implementare il seguente algoritmo: data una parola di testo T, la si confornti con la parola a metá del dizionario, D. Possono darsi tre casi: (a) $T = D$, nel qual caso la ricerca termina; (b) in ordine alfabetico $T > D$, nel qual caso la ricerca prosegue nella parte di dizionario che consiste della parole che, in ordine alfabetico, sono piú grandi di D; (c) $T < D$, caso analogo al precedente per il quale la ricerca prosegue nella parte di dizionario che consiste della parole piú piccole di D.

Esercizio 5 *La ricerca binaria termina sempre; perché?*

Confrontiamo i due metodi assumendo abbastanza realisticamente che il costo di un confronto tra due parole sia di **un microsecondo** e che il dizionario contenga **centomila vocaboli**. Per la ricerca lineare é ragionevole supporre che per ogni parola di testo sia necessario scorrere una parte sostanziale del vocabolario. Ad esempio molte parole iniziano con la lettera M, N, S, R o T e per tutte queste si dovrá scorrere piú della metá del dizionario. Assumiamo quindi salomonicamente che ogni parola del testo venga confrontata in media con la metá delle voci del dizionario. La ricerca lineare costerebbe quindi:

$$\begin{aligned}
\text{tempo} &= (\#\ \text{medio confronti}) \times (\#\ \text{costo confronto}) \\
&= (\#\ \text{parole enciclopedia}) \times \frac{1}{2}(\#\ \text{voci dizionario}) \times 10^{-6} sec \\
&= 20 \times 10^3 \times 10^3 \times \frac{10^5}{2} \times 10^{-6} sec \\
&> 11\ \text{giorni.}
\end{aligned}$$

Per calcolare il costo della ricerca binaria bisogna prima determinare il numero di confronti che sono necessari per ogni parola di testo. L'osservazione chiave é che **ogni confronto come minimo elimina metá dei vocaboli**. La domanda diventa quindi: se parto da n ed ogni volta elimino la metá, quante operazioni di "eliminazione" devo fare prima di arrivare ad 1? Come é noto la risposta é

$$\log_2 n$$

Per cui, nel caso peggiore,

$$(\#\text{ confronti per parola di testo}) \leq \log_2 100.000 \leq 16.61$$

Il costo sarebbe quindi

$$\begin{aligned}
\text{tempo} &= (\#\text{ confronti}) \times (\#\text{ costo confronto}) \\
&= (\#\text{ parole enciclopedia}) \times (\#\text{ confronti per parola di testo}) \times 10^{-6} sec \\
&\leq 20 \times 10^3 \times 10^3 \times 16.61 \times 10^{-6} sec \\
&< 6 \text{ minuti.}
\end{aligned}$$

Esercizio 6 *Confrontare al ricerca binaria e quella lineare nel caso di un romanzo in edizione tascabile di 200 pagine, assumendo un vocabolario di 200.000 voci e che il costo di un confronto sia un decimo di microsecondo (10^{-7} secondi).*

Esercizio 7 *La ricerca binaria si basa sul fatto che il dizionario é giá ordinato. Come vedremo in seguito, gli algoritmi per ordinare un insieme di n numeri o parole impiegano un tempo di calcolo proporzionale ad $n \log n$. Supponendo che si disponga di un programma di ordinamento il cui tempo di calcolo sia $100n \log n$, conviene ancora usare la ricerca binaria per la correzione ortografica dell'enciclopedia? E per i romanzi di 200 pagine?*

Ringraziamenti

Ringrazio il Prof. Riccardo Silvestri per l'esempio concernente i numeri primi.