Fabrizio Luccio. Informatica per le Biotecnologie

5 Problemi polinomiali e esponenziali

Prendiamo come esempio il problema dell'ordinamento trattato nella dispensa 4, per fare un discorso generale che riguarda la complessità computazionale dei problemi che ci troveremo a dover risolvere.

Abbiamo studiato due algoritmi di ordinamento, MERGE-SORT e INSERTION-SORT, che richiedono tempi $\Theta(n \log n)$ e $\Theta(n^2)$ nel caso pessimo. Indichiamoli brevemente con A_1 e A_2, e mettiamoli a confronto con un terzo algoritmo FOOLISH-SORT qui inventato, il cui nome indica la complessità che ci aspetta. Non essendo particolarmente utile lo descriveremo solo a parole, anche se scriverne un programma non sarebbe così difficile.

In assenza di qualsiasi idea su come eseguire un ordinamento, immaginiamo di generare una a una tutte le permutazioni dei dati d'ingresso, controllandole di volta a volta fino a individuare quella ordinata. Da quanto visto finora FOOLISH-SORT, denominato A_3, richiede tempo $n!$ per generare tutte le permutazioni (nel caso pessimo quella ordinata è l'ultima), e tempo n per verificare l'ordinamento in ciascuna di esse: in complesso tempo $\Theta(n^{3/2}(n/e)^n)$ ricordando l'approssimazione di Stirling del fattoriale (dispensa 4).

A_1 e A_2 sono algoritmi *polinomiali* perché la loro complessità è limitata superiormente da un polinomio nella *dimensione* n dei dati d'ingresso (il tempo di A_1 è limitato superiormente da $n^{1+\epsilon}$ per qualsiasi $\epsilon > 0$ poiché $n^\epsilon > \log n$). A_3 è un algoritmo *esponenziale* perché nell'espressione del tempo n figura (anche) all'esponente. Per renderci conto di quanto ciò possa significare in pratica, poniamo che i tre algoritmi richiedano esattamente tempi espressi in secondi da $T_1 = \frac{1}{100} n \log_2 n$, $T_2 = \frac{1}{100} n^2$, $T_3 = \frac{1}{100} n^{3/2}(n/e)^n$, e indichiamo nella seguente tabella i tempi richiesti dai tre per vari valori di n.

n	4	8	16	32	64	128
T_1	0,08	0,24	0,64	1,60	3,84	8,96
T_2	0,16	0,64	2,56	10,24	40,96	163,84
T_3	0,38	$> 10^3$	$> 10^{12}$	$> 10^{37}$	$> 10^{91}$	$> 10^{215}$

È chiaro dalla tabella che al crescere di n l'algoritmo A_1 si comporta assai meglio di A_2, ma il secondo è comunque accettabile, mentre A_3 richiede tempi inammissibili già per valori di n molto piccoli. La crescita di T_1 e T_2 si può anche studiare facilmente notando che al raddoppiare di n si ha:

$$T_1(2n) = \frac{1}{100} 2n \log_2(2n) = \frac{2}{100} n(\log_2 2 + \log_2 n) = 2T_1(n) + \frac{2}{100} n,$$

$$T_2(2n) = \frac{1}{100}(2n)^2 = 4T_2(n).$$

Per quanto riguarda T_3, la sua impressionante crescita non è gran che influenzata da fattori moltiplicativi, ma dipende dalla presenza del fattore esponenziale

$(n/e)^n$. Qualunque funzione esponenziale presenta infatti una crescita praticamente irragionevole: un ipotetico algoritmo di ordinamento A_4 con tempo $T_4 = \frac{1}{100}2^n$, richiederebbe più di 10^{39} secondi, cioè più di 10^{29} millenni, per ordinare 128 elementi.

Vediamo ora come l'impiego di un calcolatore C' più veloce del calcolatore C su cui sono rilevati i valori della tabella precedente, possa influenzare i valori di n per cui il calcolo risulta ragionevole. Per comprendere l'andamento del fenomeno, immaginiamo che la velocità di C' sia k volte maggiore di quella di C e vediamo come cresce la dimensione dei dati trattabili a pari tempo di elaborazione: calcoliamo cioè il nuovo valore n' ottenuto con gli algoritmi A_1, A_2, A_4 (per semplicità non consideriamo A_3) impiegando C' per un tempo $\bar{t}$, rispetto al valore n ottenuto impiegando C per lo stesso tempo. Come vedremo l'incremento da n a n' dipende dalla complessità dell'algoritmo impiegato: migliore è l'algoritmo, maggiore è il vantaggio ottenuto con un calcolatore più veloce.

Poiché C' è k volte più veloce di C, impiegare C' per $\bar{t}$ secondi equivale in prima approssimazione a impiegare C per $k\bar{t}$ secondi. Per A_1 possiamo quindi scrivere:

$$\frac{1}{100}n\log_2 n = \bar{t}, \quad \frac{1}{100}n'\log_2 n' = k\bar{t},$$

da cui si ottiene immediatamente $n'\log_2 n' = kn\log_2 n$. Questa relazione non è facilmente esplicitabile, ma la lentezza di crescita della funzione logaritmo indica che i valori di n' sono quasi k volte i valori di n. Per esempio utilizzando un calcolatore 10 volte più veloce, cioè $k = 10$, per un tempo $\bar{t} = 8,96$ secondi, con A_1 si possono ordinare $n = 128$ dati su C (vedi tabella), mentre per C' avremo $n'\log_2 n' = 10 \cdot 128 \cdot \log_2 128 = 3840$, da cui $n' = 911$ con un fattore di incremento prossimo a 10 (con un algoritmo lineare avremmo ottenuto esattamente $n' = 10\,n$).

Impiegando A_2 possiamo scrivere:

$$\frac{1}{100}n^2 = \bar{t}, \quad \frac{1}{100}n'^2 = k\bar{t},$$

da cui $n' = \sqrt{k}n$. Per $k = 10$ abbiamo $n' = 3,16\,n$ con un incremento di oltre 3 volte, inferiore a quello ottenuto con A_1 che ha complessità minore di A_2, ma sempre consistente. Impiegando un algoritmo polinomiale di complessità n^c, con $c > 2$, avremmo $n' = k^{1/c}n$, con un vantaggio tanto minore quanto più è alto il valore di c, cioè quanto meno efficiente è l'algoritmo.

Da queste osservazioni si deduce che progettare algoritmi di bassa complessità è un vantaggio immediato e futuro. E in questo senso gli algoritmi esponenziali forniscono un comportamento limite. Utilizzando per esempio l'algoritmo A_4 abbiamo:

$$\frac{1}{100}2^n = \bar{t}, \quad \frac{1}{100}2^{n'} = k\bar{t},$$

da cui $2^{n'} = k2^n$; ed estraendo il logaritmo a base 2 da entrambi i membri abbiamo: $n' = n + \log_2 k$. Dunque per algoritmi esponenziali l'incremento di velocità del calcolatore comporta un beneficio quasi inesistente: non solo l'aumento da n a

n' è additivo anziché moltiplicativo, ma il termine aggiunto nella somma è estremamente basso perché ridotto dalla funzione logaritmo. Per esempio motiplicando per $k = 1024$ la velocità del calcolatore si potrebbero ordinare con A_4, a pari tempo, $n' = n + log_2 1024 = n + 10$ dati. **Gli algoritmi esponenziali sono dunque irrimediabilmente inutilizzabili**.

Chiediamoci a questo punto se esistano problemi intrinsecamente esponenziali, tali cioè da richiedere, per essere risolti, tempo sicuramente esponenziale nella dimensione dei dati. Appartengono banalmente a questa famiglia tutti i problemi che richiedono risultati di lunghezza esponenziale: si può per esempio richiedere, dato un insieme di n elementi, che se ne costruiscano le n! permutazioni. Ma problemi così formulati sono sostanzialmente non interessanti perché si chiede di produrre un risultato che non può poi essere realisticamente esaminato.

Il primo problema non banale intrinsecamente esponenziale fu individuato nel 1972 e da allora altri ne sono seguiti, afferenti in genere alla teoria dei linguaggi formali e alla logica matematica. Fortunatamente per questi problemi non sono in genere importanti in pratiche applicazioni, mentre si incontrano comunemente problemi che richiedono tempo esponenziale nel senso che non siamo capaci di risolverli meglio. Dirigiamo dunque su questi la nostra attenzione.

Per comprendere la natura del fenomeno partiamo dai *problemi decisionali* che chiedono di stabilire se esiste una soluzione con una determinata proprietà, salvo eventualmente produrre tale soluzione in una fase successiva. La risposta è dunque binaria, e ha la forma di affermazione o negazione. Un problema decisionale è detto polinomiale se esiste un algoritmo che stabilisce se esiste una soluzione con la proprietà richiesta in tempo polinomiale nella dimensione dell'input (per la precisione, nel numero di bit necessari a descrivere la particolare istanza del problema da risolvere, o in un parametro proporzionale a quel numero poiché i calcoli saranno eseguiti in ordine di grandezza). Si pone la definizione:

Definizione. $\mathcal{P}$ è l'insieme dei problemi decisionali risolubili in tempo polinomiale.

Consideriamo per esempio un insieme S di n interi positivi e un intero k, e chiediamoci se esistono due elementi di S la cui somma è k. Si tratta di un problema decisionale: lo indicheremo come $P_{\Sigma 2}$ e prenderemo n come dimensione dell'input. Considerando per esempio l'insieme S memorizzato nel seguente vettore A, per $k = 49$ la risposta è affermativa poiché esistono i due elementi $A[1]$, $A[5]$ la cui somma è 49; per $k = 42$ la risposta è negativa:

	0	1	2	3	4	5	6	7
A	22	31	7	47	30	18	15	40

Il problema $P_{\Sigma 2}$ può essere risolto banalmente con il seguente algoritmo iterativo SIGMA2(A, k) che prova la somma di tutte le possibili coppie di elementi. L'analisi di complessità è immediata perché il programma consiste unicamente in due cicli **for** uno nell'altro, per un totale di $O(n^2)$ operazioni.

SIGMA2(A, k)
  for $(i = 0; i \leq n-2; i++)$
    for $(j = i+1; j \leq n-1; j++)$
      if $(A[i] + A[j] == k)$ **return** una coppia esiste;
  return nessuna coppia esiste;

Esercizio 1. Ideare un algoritmo per risolvere $P_{\Sigma 2}$ con $O(n \log n)$ operazioni ordinando inizialmente il vettore A.

$P_{\Sigma 2}$ corrisponde per esempio alla domanda: dati n file di dimensioni note, esistono due di essi che riempiono esattamente un disco di dimensione k? In caso affermativo interessa ovviamente individuare i due file, ma gli algoritmi per deciderne l'esistenza li determinano implicitamente: la difficoltà di questa risposta non è maggiore, in ordine di grandezza, di quella della decisione sulla loro esistenza.

Per quanto riguarda il limite inferiore al tempo di calcolo di $P_{\Sigma 2}$ applichiamo anzitutto il metodo dell'albero di decisione. Il numero di soluzioni del problema è $S(n) = n(n-1)/2+1$, ove $n(n-1)/2$ è il numero di coppie di elementi ciascuna delle quali potrebbe essere una soluzione, e 1 rappresenta il caso in cui nessuna coppia abbia somma k. Considerando i confronti eseguiti dal programma otteniamo un albero di profondità $\geq \log_2 S(n) = \log_2(n^2/2 - n/2 + 1)$, quindi di ordine $\Omega(\log n)$ poiché $\log n^2 = 2 \log n$. Molto più forte è il limite $\Omega(n)$ che deriva dalla semplice osservazione che tutti gli elementi di A devono essere esaminati almeno una volta. Si noti invece che **non tutte le coppie** di elementi devono essere necessariamente esaminate: se per esempio abbiamo esaminato le coppie $(A[i],A[j])$ e $(A[i],A[r])$ stabilendo che $A[i] + A[j] > k$ e $A[i] < A[r]$, non dobbiamo esaminare la coppia $(A[r],A[j])$ la cui somma è sicuramente $> k$.

Dunque siamo ancora in presenza di un gap tra il limite inferiore $\Omega(n)$ e il limite superiore $O(n \log n)$ dell'esercizio 1.

Generalizziamo ora $P_{\Sigma 2}$ nel nuovo problema di decisione P_Σ (noto in gergo come "somma-di-sottoinsieme"): dato un insieme S di n interi positivi e un intero k, esiste un sottoinsieme di S di dimensioni arbitrarie la cui somma degli elementi è k? Nell'esempio numerico precedente, per $k = 92$ la risposta affermativa poiché $A[2] + A[4] + A[6] + A[7] = 92$.

Per P_Σ, e per molti altri problemi, non si conosce un algoritmo polinomiale e si deve ricorrere a un'enumerazione esponenziale di scelte per risolverli. Nel caso presente un metodo tipico è quello di associare ad A un *vettore di appartenenza* V contenente interi 0, 1 con il significato: $A[i]$ fa parte di una scelta di elementi se e solo se $V[i] = 1$. Genereremo quindi tutti i possibili vettori V di n elementi e verificheremo se per uno di essi è verificata l'uguaglianza

$$\sum_{i=0}^{n-1} A[i] \cdot V[i] = k \tag{1}$$

nel qual caso la risposta è affermativa (si noti infatti che solo gli elementi per i quali $V[i] = 1$ contribuiscono alla somma). Nell'esempio avremo la soluzione:

	0	1	2	3	4	5	6	7
A	22	31	7	47	30	18	15	40
V	0	0	1	0	1	0	1	1

Una risposta affermativa, come nell'esempio riportato, permette anche di determinare quali siano gli elementi da scegliere nel sottoinsieme attraverso il particolare vettore V che verifica l'uguaglianza (1). Una risposta negativa - il sottoinsieme cercato non esiste - richiede che si generino tutti i 2^n vettori V e se ne provi la non validità, che permette di concludere che il sottoinsieme non esiste. Un algoritmo ricorsivo per generare i vettori V lunghi n contenenti 0, 1 e verificare la relazione (1) per ciascuno di essi è per esempio SIGMA(V, j) riportato sotto. Il vettore A e l'intero k sono dati globali noti al programma. Un intero t, anch'esso globale, noto al programma e da esso modificato nel corso del calcolo, indica quanti vettori di appartenenza sono stati generati fino al momento considerato. Il calcolo viene innescato dalla chiamata SIGMA$(V, 0)$ dopo aver inizializzato t a zero e genera gli elementi di V a partire da $V[0]$. Si noti la posizione delle frasi **return** che interrompono il programma se è stato determinato un sottoinsieme la cui somma è k, o se sono stati esaminati tutti i sottoinsiemi senza trovare soluzione. Naturalmente il programma potrà essere eseguito solo per piccoli valori di n a causa della dimensione esponenziale dell'uscita. Generare tutti i vettori V con un programma iterativo sarebbe notevolmente più difficile.

$t = 0$; // *t conta il numero di volte in cui il contenuto di V è stato completato*
SIGMA(V, j)
  for $(v = 0; v \leq 1; v++)$
    { $V[j] = v$;
      if $(j == n - 1)$ // *il vettore V è stato completato*
        { **if** $(\sum_{i=0}^{n-1} A[i] \cdot V[i] == k)$ **return** "un sottoinsieme esiste";
          if $(t == 2^n - 1)$ **return** "nessun sottoinsieme esiste" **else** $t = t + 1$;
        }
      else SIGMA$(V, j + 1)$;
    }

Esercizio 2 (**Difficile**, ma utile per comprendere il funzionamento di un algoritmo ricorsivo). Simulare a mano i passi dell'algoritmo SIGMA per un esempio a scelta di pochi elementi.

La trattazione matematica dei problemi esponenziali è molto tecnica e piuttosto difficile. Ci limiteremo a riportarne alcuni concetti fondamentali. Per un gran numero di essi, che si presentano come fondamentali in molti campi di applicazione (compresi alcuni aspetti computazionali della biologia) un'informazione addizionale K, detta *certificato*, permette di verificare in tempo polinomiale se i dati

D dell'istanza considerata hanno la proprietà desiderata. Tecnicamente ciò significa che esiste un algoritmo di verifica $VER(K, D)$ che controlla in tempo polinomiale l'esistenza di una soluzione per D avente la proprietà richiesta. Nel problema della somma di sottoinsieme il vettore V relativo a una soluzione costituisce un certificato perché, una volta noto, si può verificare la soluzione del problema attraverso la relazione (1) in tempo polinomiale (il calcolo della (1) richiede infatti tempo $\Theta(n)$).

Si pone allora la definizione:

Definizione. $\mathcal{NP}$ è l'insieme dei problemi decisionali verificabili in tempo polinomiale.

P_Σ appartiene dunque a $\mathcal{NP}$.

Mentre la definizione della classe $\mathcal{P}$ è semplice e intuitiva, quella della classe $\mathcal{NP}$ è obiettivamente artificiale e richiede alcune riflessioni perché se ne comprenda la portata. Anzitutto l'*esistenza* di un certificato K non ha alcun effetto sull'algoritmo di soluzione perché K non è noto a priori. Il suo scopo è quello di individuare la classe dei problemi $\mathcal{NP}$: vi sono moltissimi problemi esponenziali che non sono nemmeno certificabili, ma in genere tra essi non vi sono i problemi praticamente più importanti che sono invece in $\mathcal{NP}$.

Il certificato K deve avere lunghezza polinomiale nella dimensione di D altrimenti l'algoritmo VER non potrebbe esaminarlo in tempo polinomiale. Inoltre la definizione di $\mathcal{NP}$ implica che esista un certificato per i dati che hanno la proprietà richiesta: si certifica cioè l'esistenza della proprietà ma non la sua inesistenza. Di nuovo tutto ciò sembra assai artificiale: se un insieme S ammette una certa somma di sottoinsieme il vettore V certifica che tale somma esiste, ma se S non possiede tale somma non è richiesto (e non è comunque sicuro) che ciò sia certificabile in tempo polinomiale. Queste apparenti asimmetrie si spiegano esaminando il modo in cui un problema in $\mathcal{NP}$ può essere risolto, al di là delle definizioni formali.

In sostanza per risolvere un problema in $\mathcal{NP}$ è necessario ricorrere a un metodo enumerativo che esegua un numero esponenziale di prove per verificare se almeno una di queste corrisponde a una soluzione accettabile. Un certificato per il problema è in genere una informazione sulla sequenza di successive scelte che conducono a una soluzione, se essa esiste. Noto il certificato si possono seguire queste scelte raggiungendo direttamente la soluzione in n passi, e verificare quindi in tempo polinomiale che essa esiste. Se però una soluzione non esiste, l'algoritmo deve tentare tutte le scelte possibili prima di arrestarsi con risposta negativa, e a questo non corrisponde un certificato di lunghezza polinomiale.

Possiamo ora osservare che per ogni problema in $\mathcal{P}$ si può verificare l'esistenza di una soluzione in tempo polinomiale applicando certificato qualsiasi che viene ignorato e risolvendo direttamente il problema. Da ciò segue il risultato fondamentale:

$$\mathcal{P} \subseteq \mathcal{NP}.$$

Non è però mai stato dimostrato se vale il contenimento in senso stretto tra le due classi, cioè se sia effettivamente $\mathcal{P} \neq \mathcal{NP}$. Tale quesito è tra i più importanti

problemi irrisolti nella matematica,[1] anche se molti risultati collaterali e oltre quarant'anni di intensa ricerca inducono a credere che $\mathcal{P}$ sia strettamente un sottoinsieme di $\mathcal{NP}$; ovvero esistono problemi in $\mathcal{NP}$, come quello della somma di sottoinsiemi, che con certezza non sono risolubili in tempo polinomiale. In ogni caso finché non sarà dimostrato il contrario dovremo considerare intrattabili i problemi in $\mathcal{NP} - \mathcal{P}$ per cui non conosciamo un algoritmo polinomiale di soluzione.

Come dobbiamo dunque comportarci dinanzi a un problema esponenziale? Scartata l'ipotesi di cercare la soluzione esatta per valori di n che non siano estremamente piccoli, dovremo accontentarci di soluzioni approssimate (per esempio di una somma di elementi di un sottoinsieme di S che sia "circa" uguale a k), ove l'approssimazione richiesta deve essere precisamente definita in relazione al problema considerato. L'importante campo di studi in questo settore ha raggiunto risultati notevoli, ma non possiamo trattarne qui.

Esempi di problemi in $\mathcal{NP}$ per cui non si conosce alcun algoritmo polinomiale di soluzione

1. Biologia molecolare: dato un insieme arbitrario di sequenze di DNA stabilire se esiste una sottosequenza comune di lunghezza assegnata.

2. Manifattura: stabilire se un insieme di pezzi di lamiera possono essere ottenuti tagliandoli da un foglio assegnato disponendoli opportunamente su di esso.

3. Scheduling: dato un insieme di procedure da eseguire su un insieme di macchine, stabilire se tali procedure possono essere distribuite tra le macchine in modo che tutte queste terminino di operare entro un tempo prefissato.

4. Algebra: stabilire se un'equazione algebrica di secondo grado in due variabili x, y, a coefficienti interi, ammette una soluzione in cui x e y hanno valori interi (se l'equazione è di primo grado il problema è in $\mathcal{P}$).

Tutti questi problemi si sanno risolvere solo provando un numero di possibili soluzioni che è esponenziale rispetto ai dati d'ingresso. Si può però facilmente vedere che verificare la correttezza di una soluzione richiede tempo polinomiale.

[1] Vi è un premio internazionale di un milione di dollari per chi riuscirà a dimostrare che $\mathcal{P} \neq \mathcal{NP}$. Il premio non sarebbe però assegnato a chi dovesse dimostrare che $\mathcal{P} = \mathcal{NP}$, cioè che tutti i problemi in $\mathcal{NP}$ si possono in realtà risolvere in tempo polinomiale. Il motivo di questa asimmetria del premio è troppo complicato per spiegarlo qui.