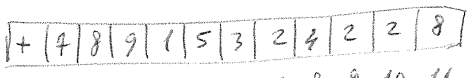
Moltiplicazione veloce di due numeri interi

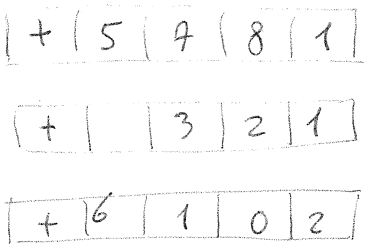
Un numero intero di n cifre, con n arbitrariamente grande, può essere rappresentato mediante un array x di $n+1$ elementi



$x[0]$ segno $\qquad$ $x[i]$ i-esima cifra più significativa, $i = 1, \dots, n$

$x[i] \in \{0, 1, 2, \dots, 9\}$

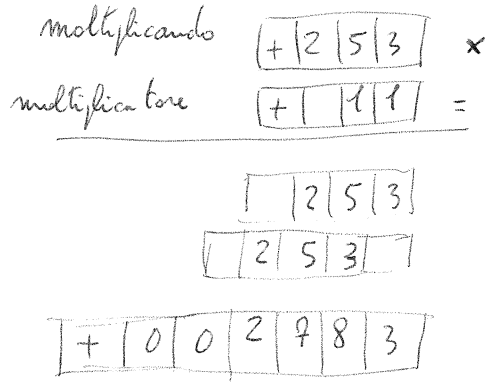
Operazioni. La somma può essere calcolata tramite l'algoritmo familiare, che addiziona le singole cifre e propaga il riporto



Richiede $O(n)$ passi.

Ottimo

Il prodotto, calcolato attraverso l'algoritmo tradizionale, risulta inefficiente.



Ogni cifra del moltiplicatore viene moltiplicata per il moltiplicando eseguendo poi n addizioni di numeri di $O(n)$ cifre.

Costo $O(n^2)$

Il prodotto di due interi di n cifre richiede un array di $2n+1$ celle

Il paradigma divide et impera, unitamente a semplici uguaglianze algebriche, permettono di progettare un algoritmo più efficiente.

Ogni numero W di n cifre si può scrivere come

$$W = W_S \times 10^{n/2} + W_d$$

W_S: $\frac{n}{2}$ cifre più significative

W_d: $n/2$ cifre meno significative

$$\underbrace{8752}_{n=4} = \underline{87} \times 10^2 + \underline{52}$$

Dati due numeri x e y, vale allora l'uguaglianza:

$$x \cdot y = (x_S \cdot 10^{n/2} + x_d)(y_S \cdot 10^{n/2} + y_d)$$

$$= 10^n x_S y_S + 10^{n/2}(x_S y_d + x_d y_S) + x_d y_d$$

Pertanto, possiamo calcolare il prodotto attraverso il seguente algoritmo

Decomposizione: se x e y hanno almeno due cifre, dividili come x_S, x_d, y_S e y_d aventi ciascuno metà cifre.

Ricorsione: calcola ricorsivamente le moltiplicazioni $x_S y_S$, $x_S y_d$, $x_d y_S$ e $x_d y_d$

Ricombinazione: combina i prodotti usando l'uguaglianza mostrata.

La complessità di questo algoritmo è facile da valutare.

Sia $T(n)$ il costo del prodotto di due numeri di n cifre.

Se $n \leq 1$ il costo è costante. Sia esso c_0

Se $n > 1$ eseguiamo 4 moltiplicazioni di numeri di $n/2$ cifre e 3 somme di due numeri di n cifre.

In realtà effettuiamo anche due prodotti per 10^n e per $10^{n/2}$, rispettivamente. Ma sono facili. Basta spostare le cifre a sinistra di k posizioni, se moltiplichiamo per 10^k



$5317 \cdot 10^4$

introducendo degli 0 nella parte destra.

Pertanto, il costo della decomposizione e della ricombinazione (somme e prodotti per 10^k) è complessivamente $O(n)$. Maggiorando tale costo con $c \cdot n$, per una opportuna costante c, risulta

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ \leq T(n/2) + c \cdot n & \text{altrimenti} \end{cases}$$

Applicando il Master Theorem si ha:

$$\alpha = 4, \quad \beta = 2 \quad \text{e} \quad f(n) = n.$$

Poiché

$$\alpha f(n/\beta) = 4\, n/2 = 2 \cdot n = 2 f(n) \qquad \forall n \geq n_0 = 1$$

rientriamo nel caso 3. $(\gamma = 2 > 1, \ \alpha = 4 > 1)$

e quindi $T(n) = O(n^{\log_\beta \alpha}) = O(n^{\log_2 4}) = O(n^2)$.

In conclusione, nessun vantaggio rispetto all'algoritmo delle scuole elementari!

Tuttavia, usando un'ulteriore uguaglianza algebrica, possiamo raffinare l'algoritmo provando che:

Teorema 3.4 La moltiplicazione di due numeri interi di n cifre decimali può essere eseguita in tempo $O(n^{\log_2 3})$.

Dimostrazione. Abbiamo visto che

$$x \cdot y = 10^n \cdot x_s y_s + 10^{n/2}(x_s y_d + x_d y_s) + x_d y_d$$

Osserviamo ancora che vale l'uguaglianza

$$x_s y_s + x_d y_d - (x_s - x_d)\cdot(y_s - y_d)$$

Infatti, sviluppando i termini,

$$\underline{x_s y_s} + x_d y_d - \underline{x_s y_s} + x_s y_d + x_d y_s - \underline{x_d y_d} .$$

Il vantaggio che ha calcolare $x_s \cdot y_d + x_d y_s$ in questo modo è che, invece di calcolare 2 nuovi prodotti e una somma, sfrutto quelli che conosco, $x_s y_s$ e $x_d y_d$, e calcolo due sottrazioni (somma di un numero e l'opposto dell'altro)

$$x + (-y) = x - y$$

e un prodotto.

La complessità dell'algoritmo diventa pertanto

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ 3\,T(n/2) + c \cdot n & \text{altrimenti} \end{cases}$$

Applicando il Master Theorem, $a = 3$, $b = 2$ e $f(n) = n$.

Poiché $a f(n/b) = 3\,\frac{n}{2} = \frac{3}{2} \cdot n = \frac{3}{2} f(n) \quad \forall n \geq n_0 = 1$

rientriamo nel caso 3. ancora e $T(n) = O(n^{\log_b a}) = O(n^{\log_2 3}) = O(n^{1,5})$

Il codice (non banale) è:

MoltiplicazioneVeloce (x, y, n) → x, y: interi di n cifre; n: potenza di 2

se non lo è aggiungiamo zeri a sinistra

```
IF (n == 1) {
    PRODOTTO[1] = (x[1] x y[1]) /10;
    PRODOTTO[2] = (x[1] x y[1]) %10;
} ELSE {
    xs[0] = xd[0] = ys[0] = yd[0] = 1;          /* segno +
    for (i = 1; i ≤ n/2; i = i+1) {              /* calcolo
        xs[i] = x[i];   ys[i] = y[i];                xs, xd, ys, yd
        xd[i] = x[i + n/2];   yd[i] = y[i + n/2];
    }
    P1 = MoltiplicazioneVeloce (xs, ys, n/2);    /*  xs·ys
    for (i = 0; i ≤ n; i = i+1)
        PRODOTTO[i] = P1[i];                     /*  xs ys · 10^n
    P2 = MoltiplicazioneVeloce (xd, yd, n/2);    /*  xd · yd
    xd[0] = yd[0] = -1;                          /* segno -
    P3 = MoltiplicazioneVeloce (Somma(xs, xd), Somma(ys, yd), n/2);
    P3[0] = -P3[0];                              /*  P3 = -(xs - xd)(ys - yd)
    add = Somma (P1, P2, P3);                    /* add = xs ys + xd yd -
    parziale[0] = add[0];                            (xs - xd)(ys - yd)
    for (i = 1; i ≤ 3·n/2; i = i+1)                  = (xs yd + xd ys)
        parziale[i] = add[i + n/2];              /*  (xs·yd + xd ys)·10^(n/2)
    PRODOTTO = Somma (PRODOTTO, parziale, P2);   /*  x·y
}
PRODOTTO[0] = X[0] x Y[0];                       /* segno +/- di x·y
RETURN PRODOTTO;                                 → numero di 2n cifre
```

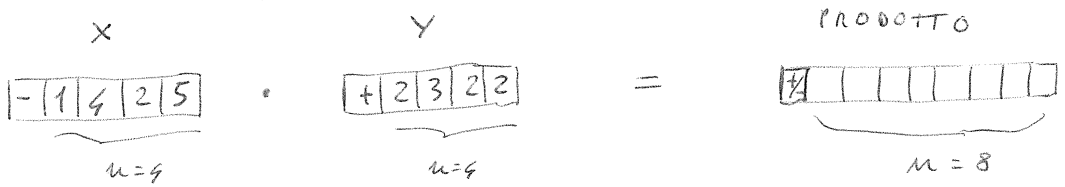
es. $2 \times 7 = 14$

$14/10$ → PRODOTTO[1]

$14 \% 10$ → PRODOTTO[2]

Esempio 3.4

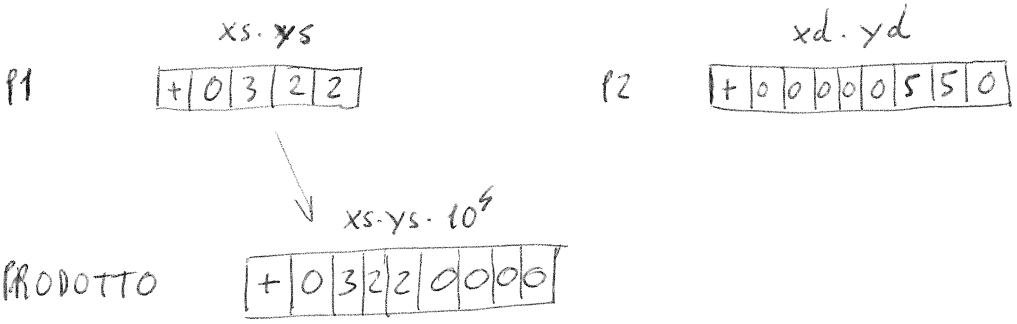
Applichiamo la MoltiplicazioneVeloce agli interi $x = -1425$ e $y = 2322$



L'algoritmo costruisce i valori assoluti delle due metà degli array x e y.



Gli array P1 e P2 contengono:

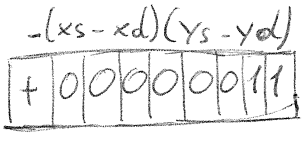


Per il calcolo di P3 si inseriscono prima i segni di xd e yd per calcolare $x_s - x_d$ e $y_s - y_d$ attraverso la funzione Somma.



$$\text{Somma}(+14, -25) = +11 \qquad \text{Somma}(+23, -22) = +01$$
$$xs \quad xd \qquad\qquad ys \quad yd$$

P3 conterrà il prodotto $-(x_s - x_d)(y_s - y_d)$ (con segno cambiato)



A questo punto viene effettuato il calcolo di $x_s y_d + x_d y_s$ mettendo il risultato in add

$x_s y_d + x_d y_s$ ↓ Somma (P1, P2, P3)

add

$x_s y_d + x_d y_s$								
+	0	0	0	0	0	8	8	3

→ parziale

$(x_s y_d + x_d y_s) \cdot 10^2$							
+	0	0	0	8	8	3	0	0

Infine il prodotto viene calcolato come somma dello stesso PRODOTTO, di parziale e di P2. Il segno è dato dai segni di x e y.

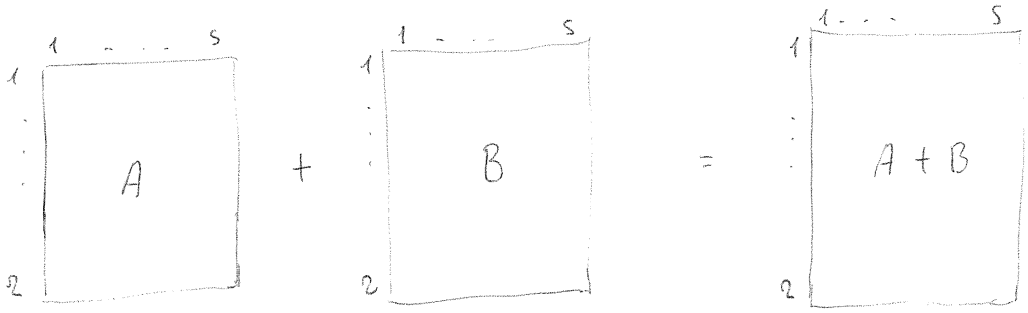
PRODOTTO ↓ parziale ↓ P2 ↓

$$x \cdot y = x_s \cdot y_s \cdot 10^4 + (x_s y_d + x_d y_s) \cdot 10^2 + x_d \cdot y_d$$

PRODOTTO =

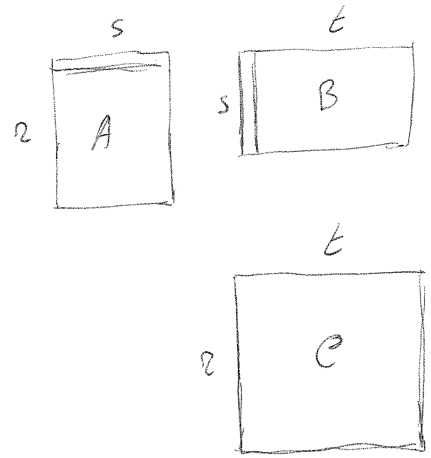
−	0	3	3	0	8	8	5	0

Moltiplicazione veloce di due matrici



L'algoritmo immediato per la somma tra due matrici effettua esattamente $2 \times s$ somme. Poiché ogni algoritmo deve sempre esaminare $\Omega(2 \times s)$ elementi, discende che l'algoritmo è ottimo.

L'algoritmo immediato per il prodotto tra due matrici non è ottimo




```
ProdottoMatrici (A, B)
    for (i=0; i<2; i=i+1)
        for (j=0; j<t; j=j+1) {
            C[i][j] = 0;
            for (K=0; K<s; K=K+1)
                C[i][j] = C[i][j] + A[i][K] x B[K][j];
        }
    RETURN C;
```

Richiede $O(s)$ operazioni per ognuno degli $2 \times t$ elementi della matrice prodotto C.

Pertanto l'algoritmo ha complessità $O(2 \times s \times t)$.

D'altra parte, il limite inferiore, ottenuto considerando il numero di elementi che ogni algoritmo deve considerare è $\Omega(2 \times s + s \times t)$.

↑ dimensioni di A ↑ dimensioni di B

Se restringiamo l'attenzione a matrici quadrate che hanno $n = 2 = s = t$ righe e colonne

$O(n^3)$

limite superiore fornito dal noto algoritmo

$\Omega(n^2)$

limite inferiore considerando gli elementi che devono essere valutati.

Esistono algoritmi più efficienti?

Può essere "colmato il gap"?

Algoritmo di Strassen

Descriviamo nel caso di matrici quadrate con n potenza di 2

Osservazione di base. Il prodotto di due matrici di dimensioni 2×2 può essere effettuato come segue: sia

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}.$$

COSTO
← 8 moltiplicazioni
4 addizioni (somme)

Se introduciamo i valori:

$$V_0 = (b-d)(g+h)$$
$$V_1 = (a+d)(e+h)$$
$$V_2 = (a-c)(e+f)$$
$$V_3 = (a+b)\cdot h$$
$$V_4 = a(f-h)$$
$$V_5 = d(g-e)$$
$$V_6 = (c+d)e$$

COSTO
← 7 moltiplicazioni
10 addizioni (somme)

allora risulta:

$$\begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix} = \begin{bmatrix} V_0+V_1-V_3+V_5 & V_3+V_4 \\ V_5+V_6 & V_1-V_2+V_4-V_6 \end{bmatrix}$$

COSTO
← 8 addizioni (4 somme)

Per esempio: $V_3 + V_4 = (a+b)\cdot h + a(f-h) = ah + bh + af - ah$

$$= af + bh$$

Gli altri casi possono essere verificati immediatamente.

Metodo standard	Metodo nuovo
↓	↓
8 moltiplicazioni	7 moltiplicazioni
4 somme	18 somme

Non sembra particolarmente vantaggioso.

Consideriamo però una matrice $n \times n$, $n > 2$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

Se guardiamo ad entrambe come a matrici 2×2 in cui gli elementi a, b, c, d, e, f, g, h sono matrici $n/2 \times n/2$, allora le somme e prodotti tra gli elementi, e.g. $ae + bg$, diventano somme e prodotti tra matrici.

Stiamo cioè applicando il paradigma divide et impera, definendo il problema del prodotto di due matrici $n \times n$ in termini dei prodotti di matrici di dimensioni $n/2 \times n/2$ e della loro "combinazione" in modo opportuno.

Pertanto, implementando la strategia di calcolo precedente, ed indicando con $T(n)$ il costo della moltiplicazione tra due matrici $n \times n$, possiamo scrivere:

$$T(n) = \begin{cases} c_0 & \text{se } n \leq 2 \\ 7\,T(n/2) + c \cdot n^2 & \text{altrimenti} \end{cases}$$

$7\,T(n/2)$: # moltiplicazioni tra matrici $n/2 \times n/2$

$c \cdot n^2$: costo delle somme di matrici $n/2 \times n/2$, numero costante e ognuna di esse costa $O(n^2)$.

Applicando il Master Theorem, risultano:

$\alpha = 7$, $\beta = 2$, $f(n) = n^2$ per cui

$$7\left(\frac{n}{2}\right)^2 = \frac{7}{4} n^2 \leq \frac{7}{4} n^2 = \gamma \cdot n^2 \qquad \forall n \geq n_0' = 1$$

$$\gamma = \frac{7}{4} > 1$$

Essendo $\alpha > 1$ e $\gamma > 1$ siamo nel caso 3.

e quindi

$$T(n) = O(n^{\log_2 7}) = O(n^{2,807}).$$

È tuttora ignota la complessità della mult. tra matrici $n \times n$ anche se si congettura che sia $O(n^\varepsilon)$ con $2 \leq \varepsilon < 2,37$.