Progetto e analisi di algoritmi

Roberto Cordone

DTI - Università degli Studi di Milano

Polo Didattico e di Ricerca di Crema

Tel. 0373 / 898**089**

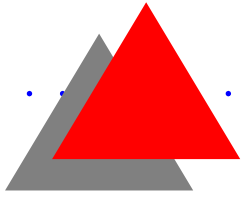
E-mail: **cordone@dti.unimi.it**

Ricevimento: **su appuntamento**

Web page: **http://www.dti.unimi.it/~cordone**

Lezioni: **Martedì dalle 11.00 alle 13.00**

Giovedì dalle 11.00 alle 13.00

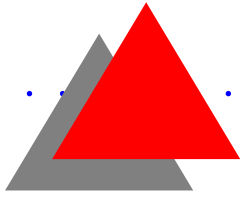


Moltiplicazione di numeri binari

$$
\begin{array}{cccccccccccc}
 & & & & & 1 & 0 & 1 & 1 & 0 & 1 & \times \\
 & & & & & & 1 & 1 & 1 & 1 & 0 & = \\
\hline
 & & & & & 0 & 0 & 0 & 0 & 0 & 0 & \\
 & & & & 1 & 0 & 1 & 1 & 0 & 1 & & \\
 & & & 1 & 0 & 1 & 1 & 0 & 1 & & & \\
 & & 1 & 0 & 1 & 1 & 0 & 1 & & & & \\
 & 1 & 0 & 1 & 1 & 0 & 1 & & & & & \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 &
\end{array}
$$

Prodotto di due bit = $\Theta(1)$; somma di n bit = $\Theta(n)$; n somme

nel caso pessimo: l'algoritmo classico ha $T(n) = \Theta(n^2)$

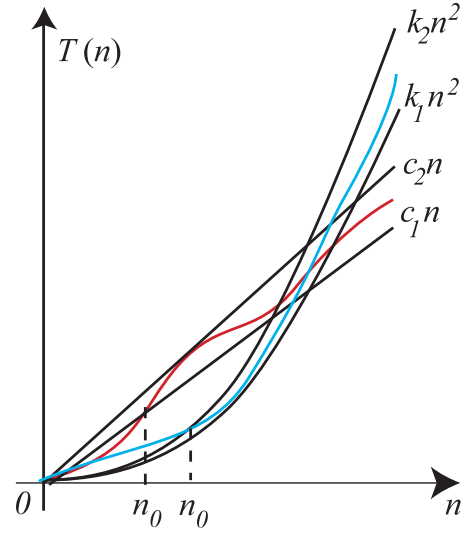


Moltiplicazione di numeri binari

L'algoritmo classico per la somma è lineare

È ottimo (stima per difetto lineare)

Il problema della somma è lineare



L'algoritmo classico per la moltiplicazione è quadratico

Non è ottimo (stima per difetto lineare, come per la somma)

Che complessità ha il problema della moltiplicazione?

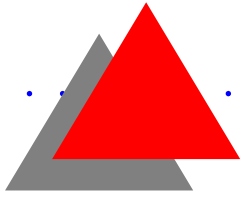
Complessità della moltiplicazione

Il problema della moltiplicazione è ancora aperto

Tuttavia sono noti algoritmi migliori di quello classico

- algoritmo di Karatsuba (1961): $\Theta\left(n^{\log_2 3}\right)$
- miglior algoritmo noto: $\Theta\left(n \log n \log\log n\right)$

Il primo è facile da progettare per induzione matematica



Moltiplicazione bit a bit

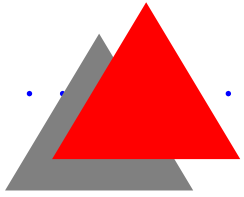
Due numeri x e y da n bit si possono scomporre

- $x = [\ a \mid b\ ]$
- $y = [\ c \mid d\ ]$

nei numeri a, b, c e d da $n/2$ bit

- $x = a\ 2^{n/2} + b$
- $y = c\ 2^{n/2} + d$

Il loro prodotto è $xy = ac\ 2^n + (ad + bc)\ \ 2^{n/2} + bd$



Un algoritmo

$$xy = ac\ 2^n + (ad + bc)\ \ 2^{n/2} + bd$$

Moltiplica(x, y)

If $|x| = 1$ and $|y| = 1$

then Return xy;

else Spezza x in (a, b) e y in (c, d)

$p_1 =$ Moltiplica(a, c); Shift(p_1, n); $\{\ p_1 := p_1 * 2^n\ \}$

$p_2 =$ Moltiplica(a, d) + Moltiplica(b, c);

Shift$(p_2, n/2)$; $\{\ p_2 := p_2 * 2^{n/2}\ \}$

$p_3 =$ Moltiplica(b, d);

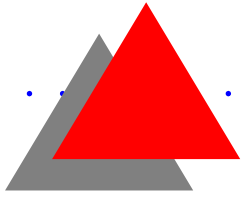
Return $p_1 + p_2 + p_3$;

Complessità

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 4T(n/2) + \Theta(n) & \text{se } n = 1 \end{cases}$$

dove $\Theta(n)$ deriva dalle operazioni di shift (moltiplicazione per 2)

$$T(n) \in \Theta(n^2)$$



Il trucco di Gauss

- $p_1 = ac$
- $p_3 = bd$
- $(a+b)(c+d) = ac + ad + bc + bd = p_1 + p_2 + p_3$

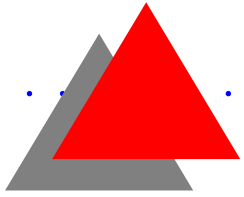
Quindi $p_2 = (a+b)(c+d) - p_1 - p_3$

Si calcola tutto con 3 prodotti (e un po' di somme)

Gauss usava questo trucco per il prodotto di numeri complessi:

$(a + i\,b)(c + i\,d) = ac - bd + i\ (ad + bc)$

Come può un risparmio del -25% diventare drammatico?



L'algoritmo di Karatsuba

Moltiplica(x, y)

If $|x| = 1$ and $|y| = 1$

then Return xy;

else Spezza x in (a, b) e y in (c, d)

$p_1 =$ Moltiplica(a, c); Shift(p_1, n); $\{\ p_1 := p_1 * 2^n\ \}$

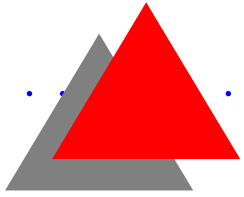
$s_1 = a + b$; $s_2 = c + d$;

$p_3 =$ Moltiplica(b, d);

$p_2 =$ Moltiplica$(s_1, s_2) - p_1 - p_3$;

Shift$(p_2, n/2)$; $\{\ p_2 := p_2 * 2^{n/2}\ \}$

Return $p_1 + p_2 + p_3$;



Complessità

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 2T(n/2) + T(n/2+1) + \Theta(n) & \\ \quad \approx 3T(n/2) + \Theta(n) & \text{se } n = 1 \end{cases}$$

$$T(n) \in \Theta\left(n^{\log_2 3}\right)$$