Algoritmo 5.4. La moltiplicazione del contadino russo genera il prodotto P tra i due interi positivi A e B:

```
procedure MOLTRUSSA(A, B, P):
    begin P←0;
        while A⩾1 do
            begin if (A è dispari) then P←P+B;
                A←⌊A/2⌋;
                B←B×2
            end
    end;
```

I passi eseguiti dall'algoritmo 5.4 per $A=37$ e $B=23$ sono indicati nella tabella 5. Si ha $P=A\times B=851$. Lasciamo all'intraprendenza del lettore la verifica della validità di questo algoritmo (la dimostrazione è grandemente facilitata se si esprimono A e B nella base 2: invero in questo modo l'algoritmo coincide con quello eseguito dai più semplici calcolatori elettronici).

Per studiare la complessità in tempo dell'algoritmo restiamo per semplicità aderenti all'ipotesi che A e B siano *interi di n cifre*, espressi nella base 10; in effetti rispetteremo implicitamente questa ipotesi per tutti i metodi di moltiplicazione esaminati in questo paragrafo. Notiamo tuttavia che nulla cambierebbe sostanzialmente se A e B non fossero interi, salvo disporre la virgola in posizione opportuna nel prodotto, né un cambiamento di base influenzerebbe minimamente i ragionamenti che seguono.

Poiché nell'algoritmo 5.4 il valore di A viene dimezzato ad ogni ciclo, la frase **while** si chiude per $A=1$ dopo $\lfloor \log_2 A \rfloor$ cicli (similmente a quanto avviene in un albero binario perfettamente bilanciato, ove si dimezza il numero di nodi ad ogni salto di livello dalle foglie verso la radice, e il numero di questi salti è pari al logaritmo del numero di foglie). Poiché il legame

Tabella 5
$37\times 23=851$, secondo il metodo del contadino russo

A	B	$P+B\to P$
37	23	0 + 23 = 23
18	46	
9	92	23 + 92 = 115
4	184	
2	368	
1	736	115 + 736 = 851
0		

tra A e il suo numero di cifre n è $n=\lfloor \log_{10} A \rfloor+1$, otteniamo infine che il ciclo di **while** è eseguito $O(n)$ volte.

Per procedere in un calcolo accurato della complessità, sarebbe necessario stabilire il costo reciproco delle diverse operazioni contenute nel ciclo di **while**: ci accontenteremo però di una valutazione in ordine di grandezza. Facendo l'ipotesi, congruente con le caratteristiche di molti sistemi di elaborazione, che un tipico costo di riferimento sia il costo costante richiesto per eseguire una operazione aritmetica tra due singole cifre (decimali, nel nostro caso), le tre frasi aritmetiche contenute nel ciclo hanno ciascuna un costo complessivo di $O(n)$, proporzionale al numero di cifre degli operandi. La complessità in tempo dell'algoritmo 5.4 è quindi di ordine $O(n^2)$.

Se torniamo al presente per esaminare l'algoritmo di moltiplicazione che ognuno di noi usa con carta e penna, scopriamo facilmente che ha anch'esso complessità in tempo di $O(n^2)$: infatti vi si eseguono n^2 moltiplicazioni tra singole cifre e vi si addizionano n prodotti parziali di $2n$ cifre al più.

Per ottenere risultati migliori dobbiamo ricorrere a un metodo di "divide et impera", le cui incerte origini si perdono indietro nel tempo.
Per comprenderne la struttura immaginiamo che sia $n=2^h$ per h intero, e denotiamo con A_1, A_2, B_1, B_2 i numeri costituiti dalle prime e dalle ultime $n/2$ cifre di A e B; cioè

$$A=A_1\times 10^{n/2}+A_2,$$
$$B=B_1\times 10^{n/2}+B_2.$$

Abbiamo così

$$\begin{aligned} A\times B&=(A_1\,10^{n/2}+A_2)(B_1\,10^{n/2}+B_2)\\ &=A_1B_1\,10^n+(A_1B_2+A_2B_1)\,10^{n/2}+A_2B_2, \end{aligned} \qquad [5.27]$$

Questo primo risultato mostra come si possa eseguire la moltiplicazione tra due numeri di n cifre mediante quattro moltiplicazioni tra numeri di $n/2$ cifre, più alcune addizioni e alcune traslazioni (o aggiunte di zeri) per eseguire le moltiplicazioni per 10^n e $10^{n/2}$. Per quanto vedremo questo risultato non è ancora interessante, ma viene migliorato notando che

$$(A_1+A_2)(B_1+B_2)=A_1B_1+A_1B_2+A_2B_1+A_2B_2;$$

che può essere impiegata per trasformare la [5.27] nella nuova espressione

$$\begin{aligned} A\times B=A_1B_1\,10^n+((A_1+A_2)(B_1+B_2)-A_1B_1-\\ -A_2B_2)\,10^{n/2}+A_2B_2. \end{aligned} \qquad [5.28]$$

La [5.28] indica come la nostra moltiplicazione possa essere eseguita mediante tre moltiplicazioni tra numeri di $n/2$ cifre $(A_1B_1, (A_1+A_2)(B_1+B_2)$ e $A_2B_2)$,[6] più alcune addizioni, sottrazioni e traslazioni.

Poniamo per esempio di avere $A=2314$ e $B=3749$. Il prodotto $A \times B$ si ottiene nel modo seguente:

$$A_1=23, A_2=14; \qquad B_1=37, B_2=49;$$
$$A_1B_1=23\times 37=851;$$
$$A_2B_2=14\times 49=686;$$
$$A_1+A_2=23+14=37; \quad B_1+B_2=37+49=86;$$
$$(A_1+A_2)(B_1+B_2)=37\times 86=3182;$$
$$A\times B=851\times 10^4+(3182-851-686)\times 10^2+686=8\,675\,186.$$

Se le moltiplicazioni nella [5.28] sono eseguite ricorsivamente applicando la stessa espressione, si ottiene l'algoritmo di moltiplicazione 5.5, impostato come procedura funzione sulla tecnica del "divide et impera".

Algoritmo 5.5. Moltiplicazione tra numeri di n cifre attraverso tre moltiplicazioni tra numeri di $n/2$ cifre:

```
procedure MOLTRAPIDA(A, B, n):
    begin
1.      if n=1 then return A × B;
2.      * dividi A in A1, A2, e B in B1, B2, di n/2 cifre *;
3.      X ← MOLTRAPIDA(A1, B1, n/2);
4.      Y ← MOLTRAPIDA(A2, B2, n/2);
5.      Z ← MOLTRAPIDA(A1+A2, B1+B2, n/2) − X − Y;
6.      return X 10^n + Z 10^(n/2) + Y
    end;
```

Il tempo $T(n)$ richiesto dall'algoritmo 5.5 si esprime come

$$T(1)=d,$$
$$T(n)=3T(n/2)+V(n), \quad \text{per} \quad n=2^h, h>0, \qquad [5.29]$$

ove $V(n)$ è il tempo necessario a eseguire quattro addizioni (due nella frase 5 e due nella frase 6), due sottrazioni (nella frase 5), due traslazioni

[6] In alcuni casi, nella moltiplicazione $(A_1+A_2)(B_1+B_2)$ i termini potrebbero contenere $n/2+1$ cifre. Si può dimostrare che i risultati che troveremo sono validi anche in questo caso.

(nella frase 6). La frase 2 comporta solo la delimitazione dei numeri A_1, A_2, B_1, B_2, e si può ritenere che richieda tempo zero o comunque costante.

Per risolvere la relazione di ricorrenza [5.29] dobbiamo conoscere l'espressione di $V(n)$. Questa dipende dalle caratteristiche dello strumento di calcolo impiegato. Se è possibile eseguire addizioni, sottrazioni e traslazioni in tempo costante per numeri di lunghezza arbitraria, si ha $V(n)=c$ costante. Questa ammissione è in genere verificata nei sistemi di elaborazione, per valori di n non troppo grandi:[7] la relazione [5.29] assume allora la forma [5.22], cui si applica la soluzione [5.23′]:

$$T(n)\in O(n^{\log_2 3})\cong O(n^{1,59}).$$

Vi è dunque un netto guadagno nella complessità se si esegue la moltiplicazione con l'algoritmo 5.5, rispetto all'usuale algoritmo di moltiplicazione che è quadratico.[8] Poiché questo guadagno emerge asintoticamente al crescere di n, può cadere l'ipotesi $V(n)=c$; vedremo in un prossimo paragrafo che l'algoritmo 5.5 è superiore anche in questo caso.

Notiamo per concludere che il metodo esposto può essere facilmente esteso per generici valori di n (cioè non necessariamente $n=2^h$); l'ordine di grandezza della complessità non cambia.

5.4 Altre partizioni

Nei paragrafi precedenti abbiamo discusso alcune famiglie di relazioni di ricorrenza che nascono da strutture ricorsive tipiche, in particolare nello studio di complessità di alcuni algoritmi fondamentali.

Moltissimi algoritmi non sono regolati dalle relazioni suddette, e dovremo costruire strumenti appositi per la valutazione di complessità, in particolare per la risoluzione di ricorrenze nuove. Lo sviluppo di successioni potrà aiutarci, similmente a quanto abbiamo fatto per risolvere le relazioni [5.22] e [5.30]: questo metodo si rivelerà utile nel caso assai frequente di relazioni di ordine non costante, che differiscano dalle [5.22] e [5.30] per la funzione che esprime il lavoro di combinazione. In particolare si dimostra per questa via che le relazioni con lavoro di combinazione polinomiale di grado $p \geqslant 0$, del tipo

$$\begin{aligned} x_1 &= d, \\ x_n &= a x_{n/b} + c_0 n^p + c_1 n^{p-1} + \ldots + c_{p-1} n + c_p, \quad n > 1, \end{aligned} \qquad [5.36]$$

hanno soluzione

$$x_n \in O(n^{\log_b a}), \quad \text{per} \quad a > b^p. \qquad [5.37]$$

Le soluzioni [5.23′] e [5.31″] già trovate per le relazioni con lavoro di combinazione costante o lineare sono quindi casi particolari [5.37] per $p = 0$ o $p = 1$.

Uno studio completo e sistematico delle relazioni di ricorrenza manca purtroppo a tutt'oggi. Un articolo di Lueker (1980) espone un vasto insieme di metodologie di risoluzione, ma per relazioni di forma generale non vi è altro mezzo che tentare una soluzione per prove: una elegante esposizione su come regolarsi seguendo questa strada si può trovare in Reingold e altri (1977, cap. 3).

5.4.1 La moltiplicazione di matrici. Un importante esempio di relazioni con lavoro polinomiale di combinazione nasce nel problema della moltiplicazione di matrici, che esamineremo limitandoci per semplicità alle matrici quadrate.

Siano $A = [a_{ij}]$ e $B = [b_{ij}]$ due matrici $n \times n$, i cui elementi siano interi di q cifre. Il prodotto $C = AB$ è notoriamente definito come $C = [c_{ij}]$, con

$$c_{ij} = \sum_{1}^{n}{}_k \, a_{ik} b_{kj}. \qquad [5.38]$$

Per calcolare il valore di c_{ij} mediante la [5.38] occorrono n moltiplicazioni e $n-1$ addizioni tra numeri di q cifre (d'ora in avanti non indicheremo più il numero di cifre su cui si opera, uguale a q per tutte le operazioni considerate). Poiché C contiene n^2 elementi, calcolare C per questa via richiede n^3 moltiplicazioni e $n^2(n-1) = n^3 - n^2$ addizioni.

Numerosissimi sono stati gli studi diretti a ridurre il numero di operazioni per calcolare C. Un metodo dovuto a Winograd, per esempio, consente di ridurre a $n^3/2 + n^2$ il numero di moltiplicazioni, pur aumentando a $3n^3/2 + 2n^2 - 2n$ il numero di addizioni: a causa dell'alto costo della moltiplicazione rispetto all'addizione (vedi § 5.2.2) il metodo di Winograd è spesso usato con ottimi risultati (per una esposizione del metodo vedi per esempio Baase, 1978).

Il numero di operazioni può essere ridotto in ordine di grandezza facendo uso di un famoso algoritmo, le cui basi sono state poste da Strassen (1969) in un brevissimo scritto. Immaginiamo che sia $n = 2^h$, e ripartiamo A, B e C in quattro sottomatrici $(n/2) \times (n/2)$; il prodotto $C = AB$ è espresso come

$$\left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]. \qquad [5.39]$$

La matrice C è nota, una volta note le sue sottomatrici C_{ij}. Ammettiamo ora che tali sottomatrici possano essere costruite mediante moltiplicazioni tra le sottomatrici di A e di B, più un certo numero di altre operazioni più semplici: ciò significa che il prodotto di due matrici $n \times n$ può essere calcolato mediante moltiplicazioni di matrici $(n/2) \times (n/2)$. Si può allora procedere ricorsivamente calcolando i prodotti tra matrici $(n/2) \times (n/2)$ mediante moltiplicazioni tra matrici $(n/4) \times (n/4)$, e così via, fino alla valutazione dei prodotti di matrici 1×1 mediante moltiplicazione diretta degli

elementi di A e B. Ciò conduce all'algoritmo 5.8, ove la funzione f per il calcolo delle C_{ij}, che deve essere ancora specificata, contiene le chiamate ricorsive della procedura per moltiplicare matrici $(n/2)\times(n/2)$.

Algoritmo 5.8. Moltiplicazione di matrici $n\times n$: $C\leftarrow AB$; la funzione f, da definire, contiene le chiamate ricorsive della procedura su matrici $(n/2)\times(n/2)$:

```
procedure MOLTMATRICI(A, B, n):
    begin
1.      if n=1 then C←[a11 × b11];
2.      * ripartisci A in A11, A12, A21, A22 e B in B11, B12, B21, B22
          (come nell'espressione [5.39]) *;
3.      {C11, C12, C21, C22}←f(A11, A12, A21, A22, B11, B12, B21, B22);
4.      * costruisci C da C11, C12, C21, C22 *
    end;
```

Per dare una forma alla funzione f notiamo che, dividendo idealmente in due parti la sommatoria che compare nell'espressione [5.38], si può esprimere c_{ij} come

$$c_{ij}=\sum_{1}^{n/2} a_{ik}b_{kj}+\sum_{n/2+1}^{n} a_{ik}b_{kj},$$

da cui deriva che ogni sottomatrice C_{ij} può essere espressa come somma delle due sottomatrici

$$C_{ij}=A_{i1}B_{1j}+A_{i2}B_{2j}. \qquad [5.40]$$

Questa espressione è formalmente identica alla [5.38], per $n=2$; essa indica che, nella moltiplicazione tra A e B, le sottomatrici della scomposizione [5.39] si possono formalmente trattare come elementi di matrici 2×2. Le C_{ij}, e da queste la C, sono quindi costruibili mediante m moltiplicazioni e g addizioni (o, come vedremo, sottrazioni) di matrici $(n/2)\times(n/2)$, con m e g da specificare. Notando che un'addizione (o sottrazione) tra matrici richiede tante addizioni tra singoli elementi quanti sono gli elementi delle matrici stesse, possiamo esprimere la complessità in tempo dell'algoritmo 5.8 con la relazione di ricorrenza

$$\begin{aligned} &T(1)=d,\\ &T(n)=mT(n/2)+gc_1(n/2)^2+c_2, \quad \text{per} \quad n=2^h, h>0, \end{aligned} \qquad [5.41]$$

ove d è il tempo costante per eseguire una moltiplicazione tra elementi di

matrice;[10] c_1 è il tempo costante per eseguire una addizione (o sottrazione) tra elementi; c_2 è il tempo costante per eseguire le frasi 2 e 4 dell'algoritmo, che richiedono semplicemente una interpretazione dei limiti di definizione delle sottomatrici.

La relazione [5.41] ha lavoro quadratico di combinazione, e ha la forma [5.36] per $p=2, a=m, b=2$. La sua soluzione è quindi la [5.37], cioè

$$T(n)\in O(n^{\log_2 m}), \qquad \text{per} \qquad m>4.$$

Se si applica direttamente l'espressione [5.40] nella funzione f, per il calcolo delle quattro sottomatrici C_{ij}, si ottiene $m=8$ e $g=4$, e la soluzione della ricorrenza diviene $T(n)\in O(n^{\log_2 8})=O(n^3)$. Niente si guadagna dunque rispetto all'algoritmo usuale di moltiplicazione, o all'algoritmo di Winograd, ma è chiaro che si può diminuire la complessità se si trova un modo per costruire le C_{ij} con meno di otto moltiplicazioni; interviene a questo proposito il seguente metodo proposto da Strassen.

Costruiamo anzitutto le sette matrici $(n/2)\times(n/2)$

$$\begin{aligned} X_1&=(A_{11}+A_{22})(B_{11}+B_{22}),\\ X_2&=(A_{21}+A_{22})B_{11},\\ X_3&=A_{11}(B_{12}-B_{22}),\\ X_4&=A_{22}(B_{21}-B_{11}),\\ X_5&=(A_{11}+A_{12})B_{22},\\ X_6&=(A_{21}-A_{11})(B_{11}+B_{12}),\\ X_7&=(A_{12}-A_{22})(B_{21}+B_{22}). \end{aligned} \qquad [5.42]$$

Le matrici C_{ij}, espresse nelle [5.40] in funzione delle A_{ik} e B_{kj}, possono ora essere costruite mediante le [5.42], con le operazioni

$$\begin{aligned} C_{11}&=X_1+X_4-X_5+X_7,\\ C_{12}&=X_3+X_5,\\ C_{21}&=X_2+X_4,\\ C_{22}&=X_1+X_3-X_2+X_6. \end{aligned} \qquad [4.43]$$

In complesso le espressioni [5.42] e [5.43] consentono di costruire le

[10] Come già osservato, si prescinde dal numero q di cifre con cui si rappresentano i valori numerici degli elementi di matrice, considerando che q abbia un prefissato valore costante. E' quindi costante il tempo per eseguire una qualsiasi operazione aritmetica tra elementi.

quattro matrici C_{ij} con sette moltiplicazioni e diciotto addizioni e sottrazioni. Si noti che la commutatività della moltiplicazione, non valida se si opera tra matrici, non interviene mai in questi calcoli. Realizzando la f per questa via, l'algoritmo 5.8 ha complessità

$$T(n) \in O(n^{\log_2 7}) \cong O(n^{2,81}).$$

Questo famosissimo risultato è rimasto insuperato per circa dieci anni; solo nel 1978 Pan ha annunciato un metodo di complessità lievemente inferiore, e da allora vari risultati si sono susseguiti rapidamente, introducendo ulteriori lievi miglioramenti nella complessità asintotica. Tra i metodi proposti il più interessante e originale è quello, già citato, dovuto a Bini e altri (1979).

Concludiamo con alcune osservazioni. Il risultato di Strassen è valido per matrici di dimensioni qualsiasi, cioè anche se n non è potenza di due: si possono infatti bordare di zeri le matrici fino ad avere $n = 2^h$, e applicare il metodo a queste nuove matrici: è facile costatare che l'ordine di grandezza del risultato non cambia.

Ad accrescere l'importanza degli studi sulla moltiplicazione contribuisce in modo determinante il fatto che un algoritmo di moltiplicazione tra matrici può essere semplicemente trasformato, senza cambiarne l'ordine di complessità, per invertire una matrice o calcolarne il determinante. Operazioni che, in forza al risultato di Strassen e dei successivi rifinimenti, possono essere eseguite in $O(n^{2,81})$ passi o meno, anziché gli $O(n^3)$ passi richiesti dai metodi diretti di calcolo. Per approfondire questi punti si consiglia di consultare per esempio il testo di Aho, Hopcroft e Ullman (1974).

Osserviamo infine che il metodo di Strassen è superiore ai metodi tradizionali in senso asintotico, e non necessariamente il suo impiego è conveniente per piccoli valori di n. Come sempre, un confronto tra i metodi, per n assegnato, non può prescindere dal modo in cui il programma è formalizzato e dalle caratteristiche del linguaggio e dell'elaboratore.