Esempi di algoritmi

8 dicembre 2025

1 Numeri

1.1 Numeri naturali

Cifre mancanti.

Problema: Sono dati due numeri di due cifre, $a2$ e $4b$ in cui a rappresenta la cifra delle decine del primo numero e b la cifra delle unità del secondo. Sapendo che $23 + 4b = a2$, quanto vale la somma di a e b? (INVALSI 2006, n35 p18)

Soluzione: 16. $a = 7, b = 9$.

Generalizzare: date quattro cifre decimali a, b, c, d, trovare le due cifre decimali x e y tali che $ab + cx = yd$.

Una possibile soluzione è l'algoritmo 1.

Algorithm 1 Determinazione delle cifre mancanti.

procedure CIFRE MANCANTI(IN: a, b, c, d. OUT: $x, y : ab + cx = yd$)
    $x \leftarrow d - b$
    if $x < 0$ **then**
        $x \leftarrow x + 10$
        $r \leftarrow 1$
    else
        $r \leftarrow 0$
    $y \leftarrow a + c + r$
    if $y > 9$ **then**
        $Return(Impossibile)$
    else
        $Return(x, y)$

Numeri a somma data.

Problema: La somma tra un numero ed il suo successivo è 45. Trova i due numeri (n37 p18).

Soluzione: 22 e 23.

Generalizzare: dato un numero naturale n, trovare i due numeri naturali consecutivi la cui somma sia n.

Una possibile soluzione è l'algoritmo 2.

Algorithm 2 Ricerca di numeri consecutivi con somma data.

```
procedure NUMERI CONSECUTIVI DI SOMMA DATA(IN: n. OUT: x, y : x + y = n, y = x + 1)
    if Resto(n, 2) = 0 then
        Return(Impossibile)
    else
        x ← Quoziente(n − 1, 2)
        y ← x + 1
        Return(x, y)
```

Problema: La somma tra due numeri pari consecutivi è 54. Trova i due numeri (n40 p18).

Soluzione: 26 e 28.

Somma e differenza date. Dati un numero naturale s e un numero naturale d, trovare i due numeri naturali la cui somma è s e la cui differenza è d.

Una possibile soluzione è l'algoritmo 3.

Algorithm 3 Ricerca di due numeri di date somma e differenza.

procedure NUMERI CON SOMMA E DIFFERENZA DATE(IN: s, d. OUT: $x, y : x + y = s, y - x = d$)
    $m \leftarrow s - d$
    if $Resto(m, 2) = 0$ **then**
        $x \leftarrow Quoziente(m, 2)$
        $y \leftarrow x + d$
        $Return(x, y)$
    else
        $Return(Impossibile)$

Enumerazione: coppie di numeri a somma costante.

Problema: Scrivi tutte le coppie di numeri naturali la cui somma è 6. (n38 p18).

Osservazione 1. Distinguere tra *coppie* e *coppie ordinate*.

Problema riformulato: Enumera le coppie ordinate di numeri naturali la cui somma è 6.

Soluzione: $(1,5),(2,4),(3,3),(4,2),(5,1)$.

Generalizzare: dato un numero naturale n, enumerare le coppie ordinate di numeri naturali la cui somma è n.

Una possibile soluzione è l'algoritmo 4.

Algorithm 4 Enumerazione di coppie di numeri di somma data.

```
procedure COPPIE CON SOMMA DATA(IN: n. OUT: S = {(x, y) ∈ ℕ² : x + y = n})
    S ← ∅
    for x = 1, ..., n − 1 do
        y ← n − x
        S ← S ∪ {(x, y)}
    Return(S)
```

Enumerazione: coppie di numeri con prodotto dato, scomposizione in fattori.

Problema: Scrivi tutte le coppie di numeri naturali il cui prodotto è 44. (n39 p18).

Soluzione (nel caso di coppie ordinate): $(1, 44), (2, 22), (4, 11), (11, 4), (22, 2), (44, 1)$.

Generalizzare: dato un numero naturale n, enumerare le coppie (non ordinate) di numeri naturali il cui prodotto è n.

Una possibile soluzione è l'algoritmo 5.

Algorithm 5 Enumerazione coppie di numeri di prodotto dato.

procedure COPPIE CON PRODOTTO DATO (V1)(IN: n. OUT: $S = \{[x, y] \in \mathbb{N}^2 : xy = n\}$)
    $S \leftarrow \emptyset$
    $x \leftarrow 1$
    while $x^2 \leq n$ **do**
        if $Resto(n, x) = 0$ **then**
            $y \leftarrow Quoziente(n, x)$
            $S \leftarrow S \cup \{[x, y]\}$
        $x \leftarrow x + 1$
    $Return(S)$

Osservazione. L'algoritmo 5 è molto inefficiente: il ciclo viene eseguito $\lfloor\sqrt{n}\rfloor$ volte, persino se n è un numero primo e l'unica coppia è $(1, n)$. Questo motiva l'introduzione dell'algoritmo di scomposizione in fattori primi per trovare tutti i fattori primi di un numero dato.

Scomposizione in fattori primi.

I fattori primi di un numero naturale non formano un insieme, bensì un multi-insieme, i cui elementi appartengono al multi-insieme con una certa molteplicità. Ad esempio il multi-insieme dei fattori primi di 60 è $\{2, 2, 3, 5\}$. Un modo alternativo di rappresentarlo ("struttura-dati") è una coppia di vettori (f, m), un vettore f con i valori dei fattori primi e un vettore m con la molteplicità di ciascuno. Così, ad esempio, la scomposizione in fattori primi di 60 corrisponde ai vettori $f = [2, 3, 5]$, $m = [2, 1, 1]$.

La procedura SFP genera il multi-insieme dei fattori primi di un numero naturale n. Nella versione dell'algoritmo 6 F è un multi-insieme; nella versione dell'algoritmo 7 f e m sono i due vettori con i fattori e le molteplicità.

Algorithm 6 Scomposizione in fattori primi, versione 1a (multi-insieme).

```
procedure SFP (V1A)(IN: n ∈ ℕ. OUT: multi-insieme F dei fattori primi di n.)
    F ← ∅
    j ← 2
    while n > 1 do
        while Resto(n, j) = 0 do
            n ← Quoziente(n, j)
            F ← F ∪ {j}
        j ← j + 1
    Return(F)
```

Algorithm 7 Scomposizione in fattori primi, versione 1b (coppia di vettori).

```
procedure SFP (V1B)(IN: n ∈ ℕ. OUT: vettore f con i fattori primi di n; vettore m con la loro molteplicità.)
    k ← 0
    j ← 2
    while n > 1 do
        /* Cerco il prossimo divisore j */
        while Resto(n, j) ≠ 0 do
            j ← j + 1
        k ← k + 1
        f[k] ← j
        m[k] ← 0
        /* Divido il numero residuo n per il fattore j */
        while Resto(n, j) = 0 do
            n ← Quoziente(n, j)
            m[k] ← m[k] + 1
    Return(f, m)
```

Questa versione di SFP (algoritmi 6 e 7) è ancora inefficiente, perché tenta di utilizzare come divisori di n tutti i numeri naturali fino a n. Basterebbe usare i soli numeri primi, come nella versione migliorata dell'algoritmo, che suppone la conoscenza di un vettore $Primi$ contenente i numeri primi ordinati da 2 in poi.

Nell'algoritmo 8 e 9 j non indica più il divisore ma l'indice del divisore, che è sempre un numero primo.

Algorithm 8 Scomposizione in fattori primi, versione 2a (multi-insieme).

procedure SFP (v2a)(IN: $n \in \mathbb{N}$. OUT: multi-insieme F dei fattori primi di n.)
    $F \leftarrow \emptyset$
    $j \leftarrow 1$
    while $n > 1$ **do**
        while $Resto(n, Primi[j]) = 0$ **do**
            $n \leftarrow Quoziente(n, Primi[j])$
            $F \leftarrow F \cup \{Primi[j]\}$
        $j \leftarrow j + 1$
    $Return(F)$

Algorithm 9 Scomposizione in fattori primi, versione 2b (coppia di vettori).

procedure SFP (v2b)(IN: $n \in \mathbb{N}$. OUT: vettore f con i fattori primi di n; vettore m con la loro molteplicità.)
    $k \leftarrow 0$
    $j \leftarrow 2$
    while $n > 1$ **do**
        /* Cerco il prossimo numero primo divisore di n */
        while $Resto(n, Primi[j]) \neq 0$ **do**
            $j \leftarrow j + 1$
        $k \leftarrow k + 1$
        $f[k] \leftarrow Primi[j]$
        $m[k] \leftarrow 0$
        /* Divido il numero residuo n per il j-esimo numero primo */
        while $Resto(n, Primi[j]) = 0$ **do**
            $n \leftarrow Quoziente(n, Primi[j])$
            $m[k] \leftarrow m[k] + 1$
    $Return(f, m)$

Numeri primi.

L'algoritmo noto come crivello di Eratostene (algoritmo 10) produce l'elenco dei numeri primi fino ad un dato valore.

Algorithm 10 Crivello di Eratostene

```
procedure CRIVELLO DI ERATOSTENE(IN: n. OUT: k: numero di numeri primi; Primi, vettore dei numeri primi non superiori a n.)
    for i = 2, ..., n do
        p[i] ← true                          ▷ p[i] indica se i può essere un numero primo.
    k ← 0                                    ▷ k: contatore di numeri primi.
    i ← 2                                    ▷ Ad inizio ciclo i è un numero primo.
    repeat
        k ← k + 1
        Primi[k] ← i
        // Cancella tutti i multipli di i //
        j ← i
        while j + i ≤ n do
            j ← j + i
            p[j] ← false
        // Cerca il prossimo numero primo //
        repeat
            i ← i + 1
        until p[i] ∨ (i > n)
    until i > n
    Return(k, Primi)
```

Coppie di numeri naturali di prodotto dato.

Ora che sappiano scomporre in fattori primi un numero n qualsiasi, possiamo definire un algoritmo più efficiente per enumerare le coppie di numeri naturali che hanno un prodotto dato. Questo algoritmo (algoritmo 11) si può ottenere enumerando le *combinazioni* dei fattori primi di n.

Algorithm 11 Enumerazione coppie di numeri naturali di prodotto dato (versione 2).

procedure COPPIE CON PRODOTTO DATO (V2)(IN: n. OUT: $S = \{[x, y] \in \mathbb{N}^2 : xy = n\}$)
    $F \leftarrow SFP(n)$ ▷ F è un multi-insieme: ogni suo elemento ha una *molteplicità*
    $k \leftarrow |F|$ ▷ k è la cardinalità di F, cioè la somma di tutte le molteplicità.
    $S \leftarrow \{[1, n]\}$
    $z \leftarrow 1$ ▷ z: numero la cui codifica binaria corrisponde ad un sottinsieme di F
    while $z < 2^k$ **do**
        $v \leftarrow Base2(z)$
        $x \leftarrow 1$
        for $i = 1, \ldots, k$ **do**
            $x \leftarrow xF[i]^{v[i]}$
        $y \leftarrow Quoziente(n, x)$
        $S \leftarrow S \cup \{[x, y]\}$
        $z \leftarrow z + 1$
    $Return(S)$

Osservazione. L'algoritmo 11 è molto più efficiente dell'algoritmo 5, perché utilizza solo i fattori primi di n. Tuttavia, genera inutilmente "doppioni" ogni volta che i fattori primi compaiono in F con molteplicità maggiore di 1. Quindi si può ulteriormente migliorare. La versione migliorata (algoritmo 13) richiede di generare iterativamente tutte le combinazioni di un multi-insieme, evitando le ripetizioni. Esso si basa sull'algoritmo 12 per generare iterativamente tutte le combinazioni degli elementi di un insieme, che quindi va introdotto prima.

Enumerazione delle combinazioni degli elementi di un insieme.

Algorithm 12 Generazione delle combinazioni degli elementi di un insieme.

```
procedure COMBINAZIONI DI UN INSIEME(IN: un insieme N. OUT: S = {M ⊆ N} : ∀M ⊆ N, M ∈ S)
    S ← ∅
    k ← |N|
    for i = 1, ..., k do
        v[i] ← 0
    repeat
        // Genera il sottinsieme M corrispondente al vettore binario v //
        M ← ∅
        for i = 1, ..., k do
            if v[i] = 1 then
                M ← M ∪ {i}
        S ← S ∪ M
        // Cerca la prossima combinazione //
        i ← 0
        repeat
            i ← i + 1
            v[i] ← 1 − v[i]
        until (v[i] = 1) ∨ (i = k)
    until (v[i] = 0)
```

Enumerazione delle combinazioni degli elementi di un multi-insieme.

La stessa idea si utilizza nell'algoritmo 13, che però lavora non su un insieme ma su un multi-insieme.

Algorithm 13 Enumerazione delle coppie di numeri naturali di prodotto dato (versione 3).

procedure COPPIE CON PRODOTTO DATO (V3)(IN: n. OUT: $S = \{[x, y] \in \mathbb{N}^2 : xy = n\}$)
    $(f, m, k) \leftarrow SFP(n)$ ▷ k è la cardinalità dei vettori f (fattori) e m (molteplicità)
    $S \leftarrow \emptyset$
    for $i = 1, \ldots, k$ **do**
        $s[i] \leftarrow 0$
    repeat
        // Genera il sottinsieme M corrispondente al vettore intero s //
        $x \leftarrow Prodotto(f, s)$ ▷ x contiene i fattori di f con molteplicità s
        $y \leftarrow Quoziente(n, x)$
        $S \leftarrow S \cup \{[x, y]\}$
        // Genera la prossima combinazione //
        $i \leftarrow 0$
        repeat
            $i \leftarrow i + 1$
            $s[i] \leftarrow Resto(s[i] + 1, m[i] + 1)$
        until $(s[i] > 0) \vee (i = k)$
    until $(s[i] = 0)$
    $Return(S)$

Da combinazione di fattori a numero.

La procedura $Prodotto(f, s)$ (algoritmo 14) genera dai due vettori, uno con i fattori primi e l'altro con la corrispondente molteplicità, il numero corrispondente.

Algorithm 14 Generazione del prodotto.

```
procedure PRODOTTO(IN: vettori f, m di cardinalità k con i fattori primi e la loro molteplicità. OUT: il numero x che ha la data scomposizione)
    x ← 1
    for i = 1, ..., k do
        x ← x × f[i]^m[i]
    Return(x)
```

Rappresentazione dei numeri in base 2.

L'algoritmo 15 trasforma un numero intero non-negativo nella sua rappresentazione in base 2 in un vettore binario v.

Algorithm 15 Rappresentazione di un numero in base 2.

```
1: procedure PRODOTTO(IN: un numero intero non-negativo n. OUT: vettore binario v con la sua rappresentazione in base
   2; b: indice bit più significativo.)
2:     i ← −1
3:     x ← n
4:     repeat
5:         i ← i + 1
6:         v[i] ← Resto(x, 2)
7:         x ← Quoziente(x, 2)
8:     until x = 0
9:     Return(v, i)
```

Per dimostrare la correttezza dell'algoritmo, fissiamo un punto preciso nel ciclo Repeat: quello al termine del corpo del ciclo, dopo l'istruzione 7. L'invariante di ciclo è la quantità

$$H = \sum_{j=0}^{i} 2^j v[j] + 2^{i+1} x = n,$$

che resta costante. Essa è data dalla somma di due termini, entrambi non-negativi: il termine $\sum_{j=0}^{i} 2^j v[j]$ indica il valore totale dei bit meno significativi già calcolati, da quello con indice 0 e peso 2^0 a quello con indice i e peso 2^i; il termine $2^{i+1}x$ indica il valore residuo da tradurre nella codifica binaria usando i bit più significativi a partire dal bit di indice $i+1$ e peso 2^{i+1}.

Identifichiamo ogni iterazione con il valore della variabile i, che inizia da 0. Sia $x(i)$ il valore di x al termine dell'iterazione i. Le istruzioni del ciclo implicano

$$v[i] = Resto(x(i-1), 2) \ \ \forall i \geq 0 \tag{1}$$

e

$$x(i) = Quoziente(x(i-1), 2) \ \ \forall i \geq 0 \tag{2}$$

Quando $i = -1$, cioè prima della prima iterazione del ciclo, $x(-1) = n$.

Per le proprietà delle divisioni, la divisione intera $Quoziente(a, b)$ è data da

$$Quoziente(a, b) = \frac{a - Resto(a, b)}{b}.$$

Quindi si può riscrivere la (2) come

$$x(i) = \frac{x(i-1) - Resto(x(i-1), 2)}{2} \ \ \forall i \geq 0$$

e per sostituzione della (1) si ha

$$x(i) = \frac{x(i-1) - v[i]}{2} \ \ \forall i \geq 0. \tag{3}$$

Consideriamo ora l'invariante di ciclo

$$H(i) = \sum_{j=0}^{i} 2^j v[j] + 2^{i+1} x(i).$$

Quando $i = -1$ il suo valore è dato da

$$H(-1) = \sum_{j=0}^{-1} 2^j v[j] + 2^0 x(-1) = n.$$

Infatti la sommatoria vale 0, poiché non ha addendi, e $x(-1) = n$, come osservato sopra.

Dimostriamo quindi che si tratta di un invariante, cioè che $H(i) = H(i-1)$ ad ogni iterazione $i \geq 0$:

$$\begin{aligned} H(i) &= \sum_{j=0}^{i} 2^j v[j] + 2^{i+1} x(i) = \\ &= \sum_{j=0}^{i-1} 2^j v[j] + 2^i v[i] + 2\, 2^i \frac{x(i-1) - v[i]}{2} = \\ &= \sum_{j=0}^{i-1} 2^j v[j] + 2^i v[i] + 2^i (x(i-1) - v[i]) = \\ &= \sum_{j=0}^{i-1} 2^j v[j] + 2^i x(i-1) = H(i-1). \end{aligned}$$

Per completare la dimostrazione di correttezza, osserviamo che la terminazione è garantita dal fatto che i è monotonicamente crescente e intero. Perciò 2^{i+1} prima o poi supera il valore n, imponendo che il secondo termine dell'invariante sia nullo, cioè che si abbia $x(i) = 0$ e questa è appunto la condizione di terminazione del ciclo.

All'uscita dal ciclo, dato che l'invariante vale n ed il suo secondo termine vale 0, si ha quindi

$$\sum_{j=0}^{i} 2^j v[j] = n,$$

cioè il vettore v contiene la codifica in base 2 di n, dal bit meno significativo in posizione 0 al bit più significativo in posizione i.

Addizione di due numeri in base 2.

Algorithm 16 Addizione di due numeri rappresentati in base 2.

```
procedure PRODOTTO(IN: due numeri interi non-negativi a e b rappresentati in base 2 con M + 1 bit. OUT: numero intero c = a + b rappresentato in base 2.)
    r ← 0                                   ▷ Riporto dalle somme nelle posizioni precedenti
    for k = 0, . . . , M do
        s ← a[k] + b[k] + r
        c[k] ← Resto(s, 2)
        r ← Quoziente(s, 2)
    c[M + 1] ← r
    Return(c, M + 1)
```

2 Frazioni e numeri razionali

Se si utilizza una coppia di interi (n, d) per rappresentare un numero razionale, si possono definire gli algoritmi che operano sulle frazioni.

Riduzione di una frazione ai minimi termini.

Confronto tra due frazioni.

Addizione di frazioni.

Moltiplicazione di frazioni.

Divisione tra frazioni.

3 Polinomi

Ogni polinomio di grado g è descritto da $g+1$ coefficienti, ordinati, cioè da un vettore. Ad esempio, il polinomio $x^3 - 2x^2 + 4x + 5$ corrisponde al vettore $[1\ -2\ 4\ 5]$. Per uniformità di rappresentazione, rappresentiamo i coefficienti in ordine inverso, in modo che ogni componente k nel vettore contenga il coefficiente di x^k: $[5\ 4\ -2\ 1]$. La prima componente ha indice 0 e rappresenta il termine noto (coefficiente di x^0).

Somma di polinomi.

Con questa convenzione, ora è facile definire gli algoritmi che calcolano la somma di due polinomi (algoritmo 17) ed il prodotto di due polinomi (algoritmo 18).

Algorithm 17 Somma di due polinomi.

procedure SOMMA DI POLINOMI(IN: due polinomi p_1 di grado g_1 e p_2 di grado g_2. OUT: un polinomio $s = p_1 + p_2$.)
  $g \leftarrow \max\{g_1, g_2\}$ ▷ Grado del polinomio s
  for $k = 0, \ldots, g$ **do**
    $s[k] \leftarrow p_1[k] + p_2[k]$
  $Return(s, g)$

Prodotto di polinomi.

Per eseguire (e per rappresentare intuitivamente) l'algoritmo che esegue il prodotto tra due polinomi, si può utilmente definire una matrice m con $g_1 + 1$ righe e $g_2 + 1$ colonne, essendo g_1 e g_2 i gradi dei due polinomi. Ogni elemento $m[i, j]$ è dato dal prodotto $p_1[i]p_2[j]$. Il coefficiente del termine di grado g nel polinomio prodotto è dato dalla somma di tutti gli elementi di $m[i, j]$ i cui indici di riga e colonna hanno somma g.

Algorithm 18 Prodotto di due polinomi.

procedure PRODOTTO DI POLINOMI(IN: due polinomi p_1 di grado g_1 e p_2 di grado g_2. OUT: un polinomio $s = p_1 p_2$.)
  $g \leftarrow g_1 + g_2$ ▷ Grado del polinomio s
  for $k = 0, \ldots, g$ **do**
    $s[k] \leftarrow 0$
  for $k_1 = 0, \ldots, g_1$ **do**
    for $k_2 = 0, \ldots, g_2$ **do**
      $s[k_1 + k_2] \leftarrow s[k_1 + k_2] + p_1[k_1] * p_2[k_2]$
  $Return(s, g)$

Divisione tra polinomi.

La rappresentazione a matrice usata per calcolare il prodotto tra due polinomi suggerisce un algoritmo per la divisione tra polinomi, probabilmente più intuitivo di quello - equivalente - illustrato comunemente sui libri di testo scolastici di matematica.

Consideriamo due polinomi $A(x)$ di grado $\alpha > 0$ e $B(x)$ di grado $\beta > 0$ con $\alpha \geq \beta$. Vogliamo trovare due polinomi $Q(x)$ di grado γ e $R(x)$ di grado ρ, tali che

$$A(x) = B(x)Q(x) + R(x),$$

con il requisito $\rho < \beta$. Quindi, in generale il grado di $Q(x)$ è $\gamma = \alpha - \beta$, mentre il grado di $R(x)$ è $\rho = \beta - 1$.

Definiamo una matrice M con $\beta + 1$ righe e $\gamma + 1$ colonne, numerate da 0. Le righe corrispondono ai coefficienti del polinomio $B(x)$, cioè $b_0, b_1, \dots, b_\beta$, le colonne ai coefficienti del polinomio $Q(x)$, cioè $q_0, q_1, \dots, q_\gamma$.

Definiamo una somma parziale σ_k per tutti i coefficienti del polinomio $A(x)$, cioè per $k = 0, 1, \dots, \alpha$.

Basta ora eseguire al contrario la procedura di moltiplicazione, definita in precedenza. Per ogni diagonale della matrice, dove gli indici di riga e colonna sommano a k, partendo da $k = \alpha$ e scendendo fino a $k = \beta$, la somma parziale σ_k deve risultare pari ad a_k. Ad ogni iterazione k, tutti i termini della somma sono già stati calcolati tranne quello sulla riga β. Pertanto si ricava $M[\beta, k - \beta] = a_k - \sigma_k$. Tale elemento della matrice M deve essere il risultato della moltiplicazione tra $b[\beta]$ e $q[k - \beta]$. Poiché $b[\beta]$ è noto, si ricava $q[k - \beta]$. Una volta ricavato $q[k - \beta]$, tutti gli elementi della colonna $k - \beta$ della matrice si possono calcolare, aggiungendo ogni elemento così calcolato alla somma parziale della sua diagonale: per ogni riga $i = 0, \dots, \beta - 1$, si ha $M[i, k - \beta] = b[i]q[k - \beta]$ e tale valore viene aggiunto a $\sigma[i + k - \beta]$.

Infine, per k che va da ρ a 0, si determinano i coefficienti del resto $R(x)$: $r[k] = a[k] - \sigma[k]$.

Algorithm 19 Divisione tra due polinomi.

procedure DIVISIONE TRA DUE POLINOMI(IN: due polinomi A di grado α e B di grado $\beta \leq \alpha$. OUT: due polinomi Q di grado γ e R di grado ρ, tali che $A = BQ + R$ con $\rho < \beta$.)
    for $k = 0, \dots, k$ **do**
        $\sigma[k] \leftarrow 0$
    for $k = \alpha, \dots, \beta$ **do**
        $M[\beta, k - \beta] \leftarrow a[k] - \sigma[k]$
        $q[k - \beta] \leftarrow M[\beta, k - \beta]/b[\beta]$
        for $i = 0, \dots, \beta - 1$ **do**
            $M[i, k - \beta] \leftarrow b[i]q[k - \beta]$
            $\sigma[i + k - \beta] \leftarrow \sigma[i + k - \beta] + M[i, k - \beta]$
    for $k = \beta - 1, \dots, 0$ **do**
        $r[k] \leftarrow a[k] - \sigma[k]$
    $Return(q, r)$

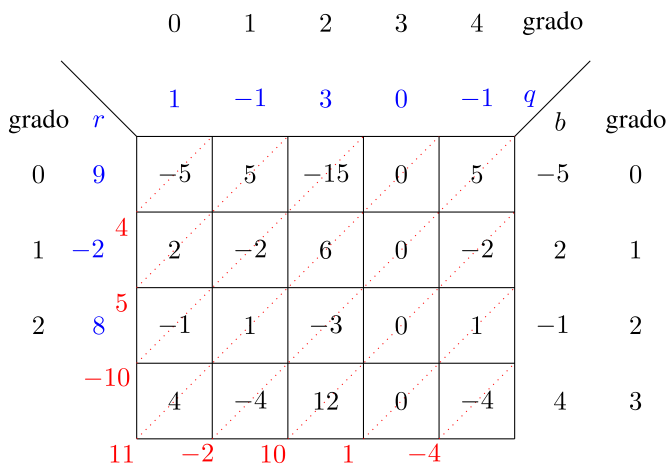

Figura 1: Esempio di divisione tra polinomi: $A(x) = -4x^7+x^6+10x^5-2x^4+11x^3-10x^2+5x+4$, $B(x) = 4x^3-x^2+2x-5$. Il risultato è $Q(x) = -x^4+3x^2-x+1$ con resto $R(x) = 8x^2-2x+9$.