Equazioni di ricorrenza - Esercizi

Pietro Di Lena

Dipartimento di Informatica – Scienza e Ingegneria
Università di Bologna

Algoritmi e Strutture di Dati
Anno Accademico 2022/2023

Esercizio 1

- Risolvere la seguente equazione di ricorrenza

$$T(n) = \begin{cases} d & n \leq 1 \\ T(n/4) + c & n > 1 \end{cases}$$

con

- Master Theorem
- Metodo dell'iterazione
- Metodo della sostituzione

Esercizio 1 - Soluzione

- Master Theorem

$$T(n) = \begin{cases} d & n \leq 1 \\ T(n/4) + c & n > 1 \end{cases}$$

Abbiamo $a = 1, b = 4, \alpha = \log_b a = 0, \beta = 0$

$$\alpha = \beta \Rightarrow T(n) = \Theta(n^0 \log n) = \Theta(\log n)$$

Esercizio 1 - Soluzione

- Metodo dell'iterazione

$$T(n) = \begin{cases} d & n \leq 1 \\ T(n/4) + c & n > 1 \end{cases}$$

Abbiamo

$$\begin{aligned} T(n) &= T(n/4) + c \\ &= T(n/4^2) + c + c \\ &= T(n/4^3) + c + c + c \\ &.. \\ &= T(n/4^i) + c \cdot i \end{aligned}$$

La ricorsione termina quando $n/4^i = 1 \Rightarrow i = \log_4 n$. Quindi

$$T(n) = T(1) + c\log_4 n = d + c\log_4 n = \Theta(\log n)$$

Esercizio 1 - Soluzione

- Metodo della sostituzione

$$T(n) = \begin{cases} d & n \leq 1 \\ T(n/4) + c & n > 1 \end{cases}$$

Ipotizziamo $T(n) = O(\log n)$, che implica ($\log = \log_2$)

$$\exists k > 0, \exists n_0 \geq 0 \text{ tale che } \forall n \geq n_0, T(n) \leq k \log n$$

1. Base: $T(4) = d + c \leq k \log_2 4$ vero per $k \geq (d+c)/2, n_0 = 4$
2. Induzione. Assumiamo che l'ipotesi sia vera per $T(n/4)$

$$\begin{aligned} T(n) &= T(n/4) + c \\ &\leq k \log(n/4) + c \\ &= k \log n - k \log 4 + c \\ &= k \log n - 2k + c \end{aligned}$$

Il passo induttivo è vero se esiste $k > 0$ tale che

$$k \log n - 2k + c \leq k \log n \Rightarrow k \geq c/2.$$

Otteniamo che $T(n) = O(\log n)$ (vera per $k \geq (d+c)/2$ e $n_0 = 4$)

Esercizio 2

- Risolvere la seguente equazione di ricorrenza

$$T(n) = \begin{cases} d & n \leq 1 \\ 2T(n/4) + c & n > 1 \end{cases}$$

con

  - Master Theorem
  - Metodo dell'iterazione
  - Metodo della sostituzione

Esercizio 2 - Soluzione

- Master Theorem

$$T(n) = \begin{cases} d & n \leq 1 \\ 2T(n/4) + c & n > 1 \end{cases}$$

Abbiamo $a = 1, b = 4, \alpha = \log_4 2 = 1/2, beta = 0$

$$\alpha > \beta \Rightarrow T(n) = \Theta(n^{\alpha}) = \Theta(\sqrt{n})$$

Esercizio 2 - Soluzione

- Metodo dell'iterazione

$$T(n) = \begin{cases} d & n \leq 1 \\ 2T(n/4) + c & n > 1 \end{cases}$$

Abbiamo

$$\begin{aligned} T(n) &= 2T(n/4) + c \\ &= 2^2 T(n/4^2) + 2c + c \\ &= 2^3 T(n/4^3) + 2^2 c + 2c + c \\ &.. \\ &= 2^i T(n/4^i) + \sum_{k=0}^{i-1} 2^k c \end{aligned}$$

La ricorsione termina quando $n/4^i = 1 \Rightarrow i = \log_4 n$. Quindi

$$T(n) = 2^{\log_4 n} T(1) + c\frac{2^{\log_4 n} - 1}{2 - 1} = \sqrt{n}d + c(\sqrt{n} - 1) = \Theta(\sqrt{n})$$

Esercizio 2 - Soluzione

- Metodo della sostituzione

$$T(n) = \begin{cases} d & n \leq 1 \\ 2T(n/4) + c & n > 1 \end{cases}$$

Ipotizziamo $T(n) = \Omega(\sqrt{n})$ (difficile dimostrare $O(\sqrt{n})$), che implica

$$\exists k > 0, \exists n_0 \geq 0 \text{ tale che } \forall n \geq n_0, T(n) \geq k\sqrt{n}$$

1. Base: $T(1) = d \geq k\sqrt{1}$ vero per ogni $k \leq d, n_0 = 1$
2. Induzione. Assumiamo che l'ipotesi sia vera per $T(n/4)$

$$\begin{aligned} T(n) &= 2T(n/4) + c \\ &\geq 2k\sqrt{n/4} + c \\ &= k\sqrt{n} + c \\ &\geq k\sqrt{n} \end{aligned}$$

Concludiamo che $T(n) = \Omega(\sqrt{n})$ (vera per $0 < k \leq d$ e $n_0 = 1$)

Esercizio 3

- Risolvere la seguente equazione di ricorrenza

$$T(n) = \begin{cases} d & n \leq 1 \\ 4T(n/4) + c & n > 1 \end{cases}$$

con

  - Master Theorem
  - Metodo dell'iterazione
  - Metodo della sostituzione

Esercizio 3 - Soluzione

- Master Theorem

$$T(n) = \begin{cases} d & n \leq 1 \\ 4T(n/4) + c & n > 1 \end{cases}$$

Abbiamo $a = 1, b = 4, \alpha = \log_4 4 = 1, \beta = 0$

$$\alpha > \beta \Rightarrow T(n) = \Theta(n^\alpha) = \Theta(n)$$

Esercizio 3 - Soluzione

- Metodo dell'iterazione

$$T(n) = \begin{cases} d & n \leq 1 \\ 4T(n/4) + c & n > 1 \end{cases}$$

Abbiamo

$$\begin{aligned} T(n) &= 4T(n/4) + c \\ &= 4^2 T(n/4^2) + 4c + c \\ &= 4^3 T(n/4^3) + 4^2 c + 4c + c \\ &.. \\ &= 4^i T(n/4^i) + \sum_{k=0}^{i-1} 4^k c \end{aligned}$$

La ricorsione termina quando $n/4^i = 1 \Rightarrow i = \log_4 n$. Quindi

$$T(n) = 4^{\log_4 n} T(1) + c\frac{4^{\log_4 n} - 1}{4 - 1} = nd + c\frac{n-1}{3} = \Theta(n)$$

Esercizio 3 - Soluzione

- Metodo della sostituzione

$$T(n) = \begin{cases} d & n \leq 1 \\ 4T(n/4) + c & n > 1 \end{cases}$$

Ipotizziamo $T(n) = \Omega(n)$ (difficile dimostrare $O(n)$), che implica

$$\exists k > 0, \exists n_0 \geq 0 \text{ tale che } \forall n \geq n_0, T(n) \geq kn$$

1. Base: $T(1) = d \geq k \cdot 1$ vero per ogni $k \leq d, n_0 = 1$
2. Induzione. Assumiamo che l'ipotesi sia vera per $T(n/4)$

$$\begin{aligned} T(n) &= 4T(n/4) + c \\ &\geq 4k(n/4) + c \\ &= kn + c \\ &\geq kn \end{aligned}$$

Concludiamo che $T(n) = \Omega(n)$ (vera per $0 < k \leq d$ e $n_0 = 1$)

Esercizio 4

- Risolvere la seguente equazione di ricorrenza

$$T(n) = \begin{cases} d & n \leq 1 \\ 8T(n/4) + c & n > 1 \end{cases}$$

con

  - Master Theorem
  - Metodo dell'iterazione
  - Metodo della sostituzione

Esercizio 4 - Soluzione

- Master Theorem

$$T(n) = \begin{cases} d & n \leq 1 \\ 8T(n/4) + c & n > 1 \end{cases}$$

Abbiamo $a = 1, b = 4, \alpha = \log_4 8 = 3/2, \beta = 0$

$$\alpha > \beta \Rightarrow T(n) = \Theta(n^{3/2}) = \Theta(\sqrt{n^3})$$

Esercizio 4 - Soluzione

- Metodo dell'iterazione

$$T(n) = \begin{cases} d & n \leq 1 \\ 8T(n/4) + c & n > 1 \end{cases}$$

Abbiamo

$$\begin{aligned} T(n) &= 8T(n/4) + c \\ &= 8^2 T(n/4^2) + 8c + c \\ &= 8^3 T(n/4^3) + 8^2 c + 8c + c \\ &.. \\ &= 8^i T(n/4^i) + \sum_{k=0}^{i-1} 8^k c \end{aligned}$$

La ricorsione termina quando $n/4^i = 1 \Rightarrow i = \log_4 n$. Quindi

$$T(n) = 8^{\log_4 n} T(1) + c\frac{8^{\log_4 n} - 1}{8-1} = \sqrt{n^3}d + c\frac{\sqrt{n^3} - 1}{7} = \Theta(\sqrt{n^3})$$

Esercizio 4 - Soluzione

- Metodo della sostituzione

$$T(n) = \begin{cases} d & n \leq 1 \\ 8T(n/4) + c & n > 1 \end{cases}$$

Ipotizziamo $T(n) = O(n^2)$ (difficile dimostrare $O(\sqrt{n^3})$), che implica

$$\exists k > 0, \exists n_0 \geq 0 \text{ tale che } \forall n \geq n_0, T(n) \leq kn^2$$

1. Base: $T(1) = d \geq k \cdot 1$ vero per ogni $k \geq d, n_0 = 1$
2. Induzione. Assumiamo che l'ipotesi sia vera per $T(n/4)$

$$\begin{aligned} T(n) &= 8T(n/4) + c \\ &\leq 8k(n^2/16) + c \\ &= k(n^2/2) + c \end{aligned}$$

Il passo induttivo è vero se esiste $k > 0$ tale che

$$k(n^2/2) + c \leq kn^2 \Rightarrow kn^2 \geq 2c \Rightarrow k \geq \frac{2c}{n^2}$$

Concludiamo che $T(n) = O(n^2)$ ($k \geq \max\{2c, d\}$ e $n_0 = 1$)

Esercizio 5

Analizzare il costo computazionale $T(n)$ dell'algoritmo FibMathPow per calcolare l'n-esima potenza della matrice di Fibonacci con la tecnica *exponentiaton by squaring*

function FibMatPow(int n) $\rightarrow$ FibMat
    $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
    if $n > 1$ **then**
        $M =$ FibMatPow($n/2$)
        $A = M \times M$
        if $n \mod 2 \neq 0$ **then**
            $A = A \times \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
    return A

Esercizio 5 - Soluzione

- Il costo di FibMatPow può essere descritto dalla seguente equazione di ricorrenza

$$T(n) = \begin{cases} 1 & n \leq 1 \\ T(n/2) + 1 & n > 1 \end{cases}$$

Secondo il Master Theorem

$$\alpha = \log_2 1 = 0 = \beta \Rightarrow T(n) = \Theta(n^0 \log n) = \Theta(\log n)$$

Esercizio 6

- Analizzare il costo computazionale $T(n)$ dell'algoritmo FibMathPow per calcolare l'n-esima potenza della matrice di Fibonacci con la tecnica *exponentiation by cubing*
- E' più efficiente di quella implementata con exponentiation by squaring?

function FibMatPow(int n) $\rightarrow$ FibMat
 $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$, $A^2 = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$
 if $n \leq 1$ **then**
  return A
 else if $n == 2$ **then**
  return A^2
 else
  $M =$ FibMatPow($n/3$)
  if $n \mod 3 == 0$ **then**   ▷ *n multiplo di 3*
   return $M \times M \times M$
  else if $n \mod 3 == 1$ **then**   ▷ *$n = 3k + 1$ per qualche $k \geq 1$*
   return $M \times M \times M \times A$
  else   ▷ *$n = 3k + 2$ per qualche $k \geq 1$*
   return $M \times M \times M \times A^2$

Esercizio 6 - Soluzione

- Il costo di FibMatPow può essere descritto dalla seguente equazione di ricorrenza

$$T(n) = \begin{cases} 1 & n \leq 2 \\ T(n/3) + 1 & n > 2 \end{cases}$$

  Secondo il Master Theorem

$$\alpha = \log_3 1 = 0 = \beta \Rightarrow T(n) = \Theta(n^0 \log n) = \Theta(\log n)$$

- La tecnica exponentiation by cubing non è asintoticamente più efficiente di exponentiation by squaring e comporta un algoritmo più complesso dal punto di vista implementativo

Esercizio 7

Analizzare il costo computazionale $T(n)$ dell'algoritmo MYSTERY1 in funzione del valore in input n

```
function MYSTERY1(INT n) → INT
    if n ≤ 1 then
        return 123
    else
        k = MYSTERY2(n/2) + MYSTERY1(n/3)
        return k + MYSTERY1(n/3)

function MYSTERY2(INT n) → INT
    if n == 0 then
        return 321
    else
        return 2 * MYSTERY2(n/4) − MYSTERY2(n/4)
```

Esercizio 7 - Soluzione

- Il costo di $\textsc{Mystery2}$ può essere descritto dalla seguente equazione di ricorrenza

$$T'(n) = \begin{cases} 1 & n = 0 \\ 2T'(n/4) + 1 & n > 0 \end{cases}$$

  Secondo il Master Theorem
  $\alpha = \log_4 2 = 1/2 > 0 = \beta \Rightarrow T'(n) = \Theta(\sqrt{n})$

- Il costo di $\textsc{Mystery1}$ può essere descritto dalla seguente equazione di ricorrenza

$$T(n) = \begin{cases} 1 & n \leq 1 \\ 2T(n/3) + n^{1/2} & n > 1 \end{cases}$$

  Secondo il Master Theorem

$$\alpha = \log_3 2 \approx 0.63 > 0.5 = \beta \Rightarrow T(n) = \Theta(n^{\log_3 2})$$

Esercizio 8

Analizzare il costo computazionale $T(n)$ dell'algoritmo MYSTERY1 in funzione del valore in input n

```
function MYSTERY1(INT n) → INT
    if n ≤ 1 then
        return 32
    else
        return MYSTERY2(n/2)+MYSTERY1(n/2)

function MYSTERY2(INT n) → INT
    if n == 1 then
        return 2
    else
        return 2*MYSTERY2(n − 1)
```

Esercizio 8 - Soluzione 1/2

- Il costo di MYSTERY2 può essere descritto dalla seguente equazione di ricorrenza

$$T'(n) = \begin{cases} 1 & n = 1 \\ T'(n-1) + 1 & n > 1 \end{cases}$$

Possiamo risolvere tale equazione di ricorrenza utilizzando il metodo iterativo

$$\begin{aligned} T'(n) &= T'(n-1) + 1 \\ &= T'(n-2) + 1 + 1 \\ &= T'(n-3) + 1 + 1 + 1 \\ &.. \\ &= T'(n-i) + i \end{aligned}$$

La ricorsione termina quando $n - i = 1 \Rightarrow i = n - 1$. Concludiamo quindi che

$$T'(n) = 1 + n - 1 = \Theta(n)$$

Esercizio 8 - Soluzione 2/2

- Il costo di MYSTERY1 può essere descritto dalla seguente equazione di ricorrenza

$$T(n) = \begin{cases} 1 & n \leq 1 \\ T(n/2) + n & n > 1 \end{cases}$$

Secondi il Master Theorem $\alpha = \log_2 1 = 0 < 1 = \beta \Rightarrow T(n) = \Theta(n)$

Esercizio 9

Analizzare il costo computazionale $T(n)$ dell'algoritmo MYSTERY1 in funzione del valore in input n

```
function MYSTERY1(INT n) → INT
    i = 1
    j = 0
    while i ≤ n do
        i = i * 2
        j = j+MYSTERY2(n/2) + MYSTERY2(n/2)
    return j

function MYSTERY2(INT n) → INT
    if n ≤ 1 then
        return 1
    else
        j = 1
        while j ≤ n do
            j = j + 1
        return j+MYSTERY2(n/3) + MYSTERY2(n/3)
```

Esercizio 9 - Soluzione

- Il costo di MYSTERY2 può essere descritto dalla seguente equazione di ricorrenza (il ciclo while viene eseguito n volte)

$$T'(n) = \begin{cases} 1 & n \leq 1 \\ 2T'(n/3) + n & n > 1 \end{cases}$$

  Per il Master Theorem $\alpha = \log_3 2 \approx 0.63 < 1 = \beta \Rightarrow T'(n) = \Theta(n)$

- La funzione MYSTERY1 richiama iterativamente MYSTERY2 due volte. Ogni chiamata costa $\Theta(n/2) + \Theta(n/2) = \Theta(n/2)$. Per completare l'analisi dobbiamo calcolare quante volte viene eseguito il ciclo while (right 4-6) in MYSTERY1. La variabile i (inizialmente 1) viene raddoppiata ad ogni iterazione e il ciclo termina quando $i > n$

iterazione	0	1	2	$\cdots$	k
valore di i	1	2	4	$\cdots$	2^k

  Quindi, il ciclo while termina all'iterazione k tale per cui

$$2^k > n \Rightarrow \log_2 2^k > log_2 n \Rightarrow k > log_2 n$$

  In altri termini, il ciclo while viene eseguito $\Theta(logn)$ volte. Quindi

$$T(n) = \Theta(\log n) \cdot (\Theta(n/2) + \Theta(n/2)) = \Theta(n \log n)$$

Analizzare il costo computazionale $T(n)$ dell'algoritmo MYSTERY1 in funzione del valore in input n

```
function MYSTERY1(INT n) → INT
    if n ≤ 1 then
        return 1
    else
        i = 1
        j = 0
        while i ≤ n do
            i = i + 2
            j = j+MISTERY2(n/2)
        return j+MYSTERY(n/4)+MYSTERY(n/4)+MYSTERY(n/4)

function MYSTERY2(INT n) → INT
    if n ≤ 1 then
        return 1
    else
        j = 1
        while j ≤ n do
            j = j + 1
        return j+MYSTERY2(n/3) + MYSTERY2(n/3)
```

Esercizio 10 - Soluzione

- Il costo di MYSTERY2 può essere descritto dalla seguente requazione di ricorrenza (il ciclo while viene eseguito n volte)

$$T'(n) = \begin{cases} 1 & n \leq 1 \\ 2T'(n/3) + n & n > 0 \end{cases}$$

Per il Master Theorem $\alpha = \log_3 2 \approx 0.63 < 1 = \beta \Rightarrow T'(n) = \Theta(n)$

- Notiamo che il ciclo while (righe 7-9) viene eseguito $n/2$ volte e ad ogni iterazione richiama la funzione MYSTERY2 con input $n/2$. Quindi il ciclo while costa complessivamente $n/2 \cdot \Theta(n/2) = \Theta(n^2)$. Il costo di MYSTERY1 può quindi essere descritto dalla seguente equazione di ricorrenza

$$T(n) = \begin{cases} 1 & n \leq 1 \\ 3T(n/4) + n^2 & n > 1 \end{cases}$$

Per il Master Theorem $\alpha = \log_4 3 \approx 0.79 < 2 = \beta \Rightarrow T(n) = \Theta(n^2)$