Equazioni di ricorrenza

Pietro Di Lena

Dipartimento di Informatica – Scienza e Ingegneria
Università di Bologna

Algoritmi e Strutture di Dati
Anno Accademico 2021/2022

Introduzione

- Una equazione di ricorrenza descrive ogni elemento in una sequenza in termini degli elementi precedenti
- Abbiamo già visto l'equazione di ricorrenza di Fibonacci

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

- Vogliamo determinare l'ordine di crescita delle equazioni di ricorrenza
  - Determinare la crescita asintotica degli algoritmi ricorsivi
- Vedremo quattro metodi per risolvere equazioni di ricorrenza
  - Metodo dell'iterazione
  - Metodo della sostituzione
  - Metodo dell'albero di ricorsione
  - Master Theorem

Nozioni preliminari

- Le equazioni di ricorrenza di algoritmi ricorsivi possono essere del tipo

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 2T(\lfloor n/3 \rfloor) + O(n) & n > 1 \end{cases}$$

- Per convenienza, tipicamente sostituiamo la notazione asintotica con valori costanti

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

- Il comportamento asintotico non è influenzato da arrotondamenti

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

Metodo dell'iterazione

1° metodo "BRUTE-FORCE"

- Il metodo dell'iterazione è un approccio di tipo *brute force*
- Idea: sostituiamo iterativamente la parte ricorsiva nell'equazione finché non appare uno schema ricorsivo legato al passo di iterazione
- Esempio. Consideriamo la ricorrenza $T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + c & n > 1 \end{cases}$

non sempre è utile esempio Fibonacci (in cui cercavamo un lowerbound)

$$
\begin{aligned}
T(n) &= T(n/2) + c & &\text{passo 1} \\
&= T(n/4) + c + c & &\text{passo 2} \\
&= T(n/8) + c + c + c & &\text{passo 3} \\
&\dots \\
&= T(n/2^i) + c \cdot i & &\text{passo } i
\end{aligned}
$$

sviluppo T(n/2)

⋮

La ricorsione termina quando $n/2^i = 1 \xRightarrow{\text{cerco } i} i = \log_2 n$. Quindi

$$T(n) = T(1) + c \cdot \log_2 n = 1 + c \cdot \log_2 n = \Theta(\log n)$$

Metodo della sostituzione

2° metodo
"per VALIDARE un IPOTESI"

- Il metodo della sostituzione può essere usato per *validare* un'ipotesi
- Idea: 1) ipotizziamo una soluzione 2) validiamo induttivamente l'ipotesi
- Esempio. Consideriamo la ricorrenza $T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + n & n > 1 \end{cases}$

Ipotizziamo $T(n) = O(n)$, che implica (per definizione di $O(n)$)

$$\exists c > 0 \text{ e } \exists n_0 \geq 0 \text{ tale che } \forall n \geq n_0,\ T(n) \leq cn$$

Dim

1. Base. $T(1) = 1 \leq c \cdot 1$, per ogni $c \geq 1$
2. Induzione. Assumiamo che l'ipotesi sia vera per $T(n/2)$

$$\begin{aligned} T(n) &= T(n/2) + n \\ &\leq cn/2 + n && \text{(assumiamo } T(n/2) \leq cn/2\text{)} \\ &= (c/2 + 1)n && \text{(dobbiamo mostrare che } (c/2+1)n \leq cn\text{)} \end{aligned}$$

Il passo induttivo è vero se $\exists c > 0$ tale che $(c/2 + 1) \leq c \Rightarrow c \geq 2$

Concludiamo che $T(n) = O(n)$ (vera $\forall c \geq 2$ e $n_0 = 0$)

Metodo della sostituzione: Fibonacci

- Cerchiamo un limite superiore alla ricorrenza di Fibonacci

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

Ipotizziamo $T(n) = O(2^n)$, che implica

$$\exists c > 0 \text{ e } \exists n_0 \geq 0 \text{ tale che } \forall n \geq n_0,\ T(n) \leq c2^n$$

1. Base. $T(1) = 1 \leq c \cdot 2^1$, $T(2) = 1 \leq c \cdot 2^2$ vera $\forall c \geq 1/2 > 1/4$
2. Induzione. Assumiamo che l'ipotesi sia vera per $T(n-1), T(n-2)$

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 \\ &\leq c2^{n-1} + c2^{n-2} + 1 \\ &= c2^{n-2}(2+1) + 1 \\ &\leq c2^{n-2}(2+2) && \text{(vera } \forall n \geq 2 - \log_2 c\text{)} \\ &= c2^n \end{aligned}$$

Concludiamo che $T(n) = O(2^n)$ (vera $\forall c \geq 1/2, n_0 \geq 2 - \log_2 c$)

Metodo della sostituzione: Fibonacci

- Cerchiamo un limite inferiore alla ricorrenza di Fibonacci
- Ipotizziamo $T(n) = \Omega(2^{n/2})$, che implica

$$\exists c > 0 \text{ e } \exists n_0 \geq 0 \text{ tale che } \forall n \geq n_0,\ T(n) \geq c2^{n/2}$$

1. Base. $T(1) = 1 \geq c \cdot 2^{1/2}$, $T(2) = 1 \geq 2c$ vera per $0 < c \leq 1/\sqrt{2}$
2. Induzione. Assumiamo che l'ipotesi sia vera per $T(n-1)$, $T(n-2)$

$$\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&\geq c2^{\frac{n-1}{2}} + c2^{\frac{n-2}{2}} + 1 \\
&= c2^{\frac{n-2}{2}} \cdot (2^{\frac{1}{2}} + 1) + 1 && \left(2^{\frac{n-1}{2}} = 2^{\frac{n-2}{2}} \cdot 2^{\frac{1}{2}}\right) \\
&\geq c2^{\frac{n-2}{2}} \cdot (2^{\frac{1}{2}} + 1) \\
&\geq c2^{\frac{n-2}{2}} \cdot 2 && \left(2^{\frac{1}{2}} + 1 > 2\right) \\
&= c2^{n/2}
\end{aligned}$$

Concludiamo che $T(n) = \Omega(2^{n/2})$ (vera $\forall 0 < c \leq 1/\sqrt{2}$ e $n_0 = 0$)

Limite stretto per la ricorrenza di Fibonacci

- Abbiamo dimostrato che la ricorrenza di Fibonacci è limitata da

$$T(n) = \Omega(\sqrt{2}^n) \approx \Omega(1.41^n) \text{ e } T(n) = O(2^n)$$

- Possiamo trovare un limite più stretto?

Teorema

Sia $T(n)$ l'equazione di ricorrenza di Fibonacci. Allora, $T(n) = 2F_n - 1$ **(Dimostrazione per induzione)**

- Dal Teorema sopra abbiamo che $T(n) = \Theta(F_n)$
- Ricordiamo che $F_n = \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right)$ dove

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618 \text{ and } \hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -0.618$$

- Concludiamo che $T(n) = \Theta(\phi^n) \approx \Theta(1.62^n)$
- Difficile *indovinare* il valore ϕ^n con il metodo della sostituzione

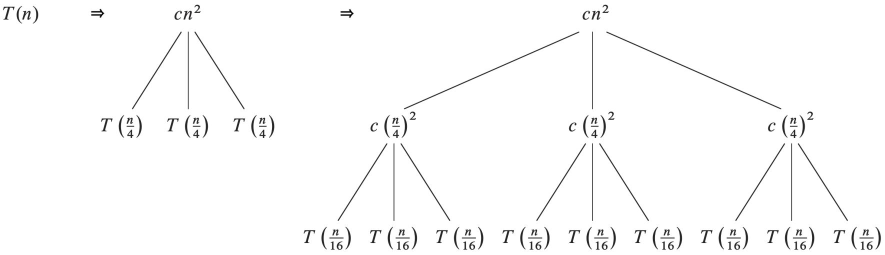
Metodo dell'albero di ricorsione

3° metodo

- In un albero di ricorsione un nodo esprime il costo di un sottoproblema
- Idea: può essere visto come la *versione su albero* del metodo iterativo
  1. Generiamo l'albero di ricorsione dall'equazione di ricorrenza
  2. Calcoliamo il numero di nodi ad ogni livello dell'albero
  3. Identifichiamo qualche schema ricorrente legato al livello dell'albero
- Può essere complesso formulare un'ipotesi (metodo della sostituzione)
  - L'albero di ricorsione può essere usato per generare ipotesi
  - Tali ipotesi possono poi essere validate col metodo di sostituzione

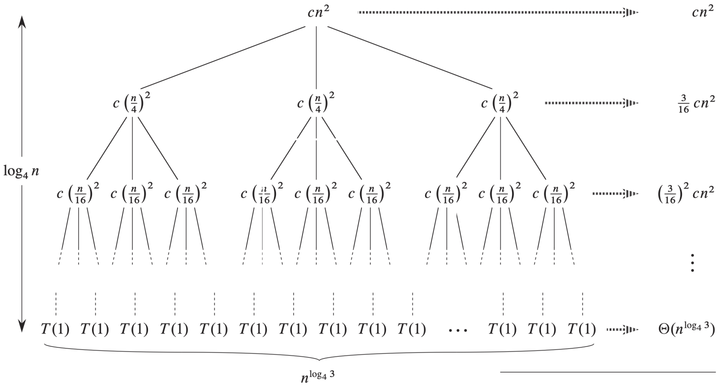
Esempio 1: albero di ricorsione

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



- La radice rappresenta il costo al livello superiore della ricorsione
- I tre sottoalberi $T\left(\frac{n}{4}\right)$ rappresentano il costo su dimensione $n/4$
- Costo di $T\left(\frac{n}{4}\right)$: $c(n/4)^2$ + costo di 3 chiamate ricorsive su $n/16$

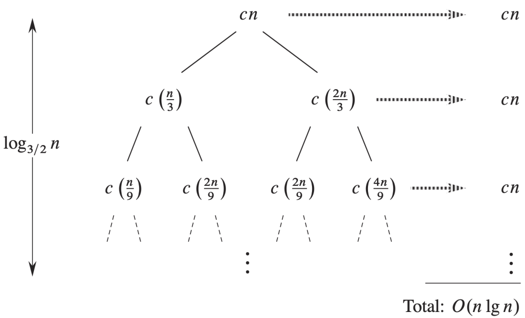
Esempio 1: albero di ricorsione



- Se la radice è a livello 0, il costo a livello i è $\left(\frac{3}{16}\right)^i cn^2$
- La ricorsione termina quando $n/4^i = 1 \Longrightarrow i = \log_4 n$
- $T(n) = \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i cn^2 \leq cn^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i = cn^2 \frac{1}{1-(3/16)} = O(n^2)$
- N.B. cn^2 è anche il costo della sola radice, quindi $T(n) = \Omega(n^2)$

Esempio 2: albero di ricorsione

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



- Il percorso più lungo radice-foglia è

$$n \to \frac{2}{3}n \to \frac{4}{9}n \cdots \to \left(\frac{2}{3}\right)^i n \to \cdots \to 1$$

- Il percorso più corto radice-foglia è

$$n \to \frac{1}{3}n \to \frac{1}{9}n \to \cdots \to \left(\frac{1}{3}\right)^i n \to \cdots \to 1$$

Esempio 2: albero di ricorsione

- La ricorsione sul percorso radice-foglia più lungo termina quando

$$\left(\tfrac{2}{3}\right)^i n = 1 \Longrightarrow i = \log_{2/3}\frac{1}{n} \underset{\text{cambio di base}}{=} \frac{-\log_{3/2} n}{\log_{3/2} 2/3} = \log_{3/2} n$$

- Upper bound per $T(n) = T(n/3) + T(2n/3) + cn$

$$T(n) \leq \sum_{i=0}^{\log_{3/2} n} cn = cn \sum_{i=0}^{\log_{3/2} n} 1 = cn \log_{3/2} n = O(n \log n)$$

- La ricorsione sul percorso radice-foglia più corto termina quando

$$\left(\tfrac{1}{3}\right)^i n = 1 \Longrightarrow i = \log_{1/3}\frac{1}{n} = \frac{-\log_3 n}{\log_3 1/3} = \log_3 n$$

- Lower bound per $T(n) = T(n/3) + T(2n/3) + cn$

$$T(n) \geq \sum_{i=0}^{\log_3 n} cn = cn \sum_{i=0}^{\log_3 n} 1 = cn \log_3 n = \Omega(n \log n)$$

- Conclusione: $T(n) = \Theta(n \log n)$

Master Theorem

metodo sottoinsieme + Kawaii

a chiamate ricorsive $\frac{n}{b}$

- Il Master Theorem è un approccio per risolvere ricorrenze della forma

$$T(n) = aT(n/b) + f(n)$$

con $a \geq 1$ e $b > 1$ constanti e $f(n)$ asintoticamente positiva

- Equazioni di ricorrenza di algoritmi che
    - dividono un problema di dimensione n in $a \geq 1$ sottoproblemi
    - tutti i sottoproblemi hanno dimensione n/b, con $b > 1$
    - il costo di ogni chiamata ricorsiva è dato da $f(n)$

- Non può essere applicato a tutte le possibili ricorrenze
    - Non può essere applicato all'equazione di ricorrenza di Fibonacci
    - Non può essere applicato all'esempio 2 (albero di ricorsione)

Mather Theorem

Theorem (Master Theorem) Versione NOSTALGICA suo

Si consideri la seguente equazione di ricorrenza + generico

$$T(n) = \begin{cases} d & n = 1 \\ aT(n/b) + f(n) & n > 1 \end{cases}$$

dove $a \geq 1$, $b > 1$, d costante e $f(n)$ è una funzione di costo. Allora

1. *Se $f(n) = O(n^{\log_b a - \epsilon})$ per qualche $\epsilon > 0$ allora $T(n) = \Theta(n^{\log_b a})$*
2. *Se $f(n) = \Theta(n^{\log_b a})$ allora $T(n) = \Theta(n^{\log_b a} \log n)$*
3. → *Se $f(n) = \Omega(n^{\log_b a + \epsilon})$ per qualche $\epsilon > 0$, e se $a \cdot f(n/b) \leq c \cdot f(n)$ per qualche costante $c < 1$ e per tutti gli n sufficientemente grandi, allora $T(n) = \Theta(f(n))$*

N.B. In tutti e tre i casi confrontiamo $f(n)$ con $n^{\log_b a}$

Mather Theorem: versione semplificata

Theorem (Master Theorem) Versione da Ricordare

Si consideri la seguente equazione di ricorrenza

$$T(n) = \begin{cases} d & n = 1 \\ aT(n/b) + cn^\beta & n > 1 \end{cases}$$

dove $a \geq 1$, $b > 1$ *e* d *costante. Sia* $\alpha = \log_b a = \frac{\log(a)}{\log(b)}$ *Allora*

1. *Se* $\alpha > \beta$ *allora* $T(n) = \Theta(n^\alpha)$
2. *Se* $\alpha = \beta$ *allora* $T(n) = \Theta(n^\alpha \log n)$
3. *Se* $\alpha < \beta$ *allora* $T(n) = \Theta(n^\beta)$

il maggiore tra α e β "vince" (è l'esponente).

N.B. Meno generale della versione precedente: ammette solo funzioni $f(n)$ della forma n^β

Esempi: Master Theorem

1. Risolvere col Master Theorem $T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + c & n > 1 \end{cases}$

   Abbiamo $a = 1, b = 2, \alpha = \log_b a = \log_2 1 = 0, \beta = 0$ poiché $n^0 = 1$

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

2. Risolvere col Master Theorem $T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + n & n > 1 \end{cases}$

   Abbiamo $a = 1, b = 2, \alpha = \log_b a = \log_2 1 = 0, \beta = 1$

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

3. Risolvere col Master Theorem $T(n) = \begin{cases} 1 & n = 1 \\ 3T(n/4) + cn^2 & n > 1 \end{cases}$

   $\log_b a$

   Abbiamo $a = 3, b = 4, \alpha = \log_{\cancel{2}_4} \cancel{4}_3 \approx \cancel{1.26}, \beta = 2$

   $$\alpha < \beta \Rightarrow T(n) = \Theta(n^\beta) = \Theta(n^2)$$

Esempio: ricorrenza della ricerca binaria

Cercare la posizione di un valore all'interno di un array ordinato (-1 se non trovato)

valore cercato

function SEARCH(ARRAY $A[1 \cdots n]$, INT x, INT i, INT j) $\rightarrow$ INT
    if $i > j$ **then**
        return -1
    else
        $m = (i + j)/2$ posizione centrale dell'array   ▷ *Divisione intera*
        if $A[m] == x$ **then**
            return m
        else if $A[m] > x$ **then**
            return SEARCH$(A, x, i, m-1)$ seconda metà
        else
            return SEARCH$(A, x, m+1, j)$ prima metà

- La prima chiamata è invocata con parametri SEARCH$(A, x, 1, n)$
- Caso ottimo (x è esattamente in mezzo) $O(1)$
- Caso pessimo x non è presente

Esempio: ricorrenza della ricerca binaria

```
function SEARCH(ARRAY A[1···n], INT x, INT i, INT j) → INT
    if i > j then
        return −1
    else
        m = (i + j)/2                                  ▷ Divisione intera
        if A[m] == x then
            return m
        else if A[m] > x then
            return SEARCH(A, x, i, m − 1)
        else
            return SEARCH(A, x, m + 1, j)
```

- Possiamo estrarre la funzione di ricorrenza dallo pseudocodice

$$T(n) = \begin{cases} 1 & n = 0 \text{ (costo constante se spazio di ricerca è 0)} \\ T(n/2) + \underbrace{1}_{cn^0} & n > 0 \text{ (costo costante + ricerca su 1/2 spazio)} \end{cases}$$

$\Rightarrow \beta = 0$

- Soluzione col Master Theorem:

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