Equazioni di ricorrenza

Pietro Di Lena

Dipartimento di Informatica – Scienza e Ingegneria
Università di Bologna

Algoritmi e Strutture di Dati
Anno Accademico 2022/2023

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 tre metodi per risolvere equazioni di ricorrenza
  - Metodo dell'iterazione
  - Metodo della sostituzione
  - Master Theorem

Nozioni preliminari

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

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

- Tipicamente sostituiamo la notazione asintotica con espressioni positive

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

- Tipicamente utilizziamo la costante 1 invece di costanti simboliche

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

- Il comportamento asintotico non è influenzato da arrotondamenti

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

Metodo dell'iterazione

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

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

La ricorsione termina quando $n/2^i = 1 \Longrightarrow i = \log_2 n$. Quindi

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

Otteniamo lo stesso risultato se sostituiamo $c = 1$

Metodo della sostituzione

- 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

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

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 $n_0 = 1$)

Metodo della sostituzione: Fibonacci

- Cerchiamo un limite superiore alla ricorrenza di Fibonacci

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

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

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

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

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

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

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 $T(n) = \Theta(2F_n - 1) = \Theta(F_n)$
- L'n-esimo numero di Fibonacci è definito da $F_n = \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right)$

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

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

Master Theorem

- 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

Mather Theorem

Theorem (Master Theorem)

Si consideri la seguente equazione di ricorrenza

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

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)

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* c, d *costanti. Sia* $\alpha = \log_b a = \dfrac{\log a}{\log b}$. *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* $T(n) = \Theta(n^{\beta})$

N.B. Meno generale della versione precedente: ammette solo funzioni $f(n)$ della forma cn^{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 $a = 1, b = 2, \alpha = \log_b a = \log_2 1 = 0, \beta = 0$

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

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

   Abbiamo $a = 3, b = 4, \alpha = \log_4 3 \approx 0.79, \beta = 2$

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

ESEMPIO: RICORRENZA DELLA RICERCA BINARIA

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

```
function BINSEARCH(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)
```

- La prima chiamata è invocata con parametri $\textsc{search}(A, x, 1, n)$
- Caso ottimo (x è nel centro dell'array): $O(1)$
- Caso pessimo (x non è nell'array): ?

Esempio: ricorrenza della ricerca binaria

```
function BINSEARCH(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

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

- Soluzione col Master Theorem:

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