Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Equazioni di ricorrenza

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

Sulla base delle slides a cura di T. Calamoneri e G. Bongiovanni per il corso di informatica generale AA 2019/2020

Equazioni di ricorrenza

- Valutare il costo computazionale di un algoritmo ricorsivo è, in genere, più laborioso che nel caso degli algoritmi iterativi.
- Infatti, la natura ricorsiva della soluzione algoritmica dà luogo a una funzione di costo che, essendo strettamente legata alla struttura dell'algoritmo, è anch'essa ricorsiva.
- Trovare la funzione di costo ricorsiva è piuttosto immediato. Essa però deve essere riformulata in modo che non sia più ricorsiva, altrimenti il costo asintotico non può essere quantificato, e questa è la parte meno semplice.
- La riformulazione della funzione di costo ricorsiva in una equivalente ma non ricorsiva si affronta impostando una ***equazione di ricorrenza***, costituita dalla **formulazione ricorsiva e dal caso base**.

Esempio: costo computazionale della ricerca sequenziale ricorsiva in una lista di n elementi:

```
def Ricerca_seq_ric(A, x, i=0):
    if i == len(A):
        return -1
    if  A[i] == x:
            return I
    return Ricerca_seq_ric(A, x, i+1)
```

la funzione viene invocata per la prima volta con
`Ricerca_seq_ric(A, x)`

In generale: $T(n) = \Theta(1) + T(n-1)$

Caso base: $T(0) = \Theta(1)$

Equazioni di ricorrenza

- La parte generale dell'equazione di ricorrenza che definisce *T(n)* deve essere sempre costituita dalla somma di almeno due addendi, di cui almeno uno contiene la parte ricorsiva (nell'esempio *T(n-1)*) mentre uno rappresenta il costo computazionale di tutto ciò che viene eseguito al di fuori della chiamata ricorsiva.
- Anche se questa parte dovesse essere un solo confronto, il suo costo non può essere ignorato, altrimenti si otterrebbe che la funzione ha un costo computazionale indipendente dalla dimensione del suo input, e questa è data da quanto illustrato nel caso base.
- Deve sempre essere presente un caso base.

Equazioni di ricorrenza

Esistono alcuni metodi utili per risolvere le equazioni di ricorrenza, che illustreremo:

- metodo iterativo
- metodo di sostituzione
- metodo dell'albero
- metodo principale

Metodo iterativo

Idea

- sviluppare l'equazione di ricorrenza ed esprimerla come somma di termini dipendenti da n e dal caso base

Difficoltà

- maggiore quantità di calcoli algebrici rispetto agli altri metodi

Metodo iterativo

Esempio: Equazione relativa alla ricerca sequenziale ricorsiva

$$T(n) = T(n-1) + \Theta(1)$$
$$T(1) = \Theta(1)$$

$T(n) = T(n-1) + \Theta(1)$

ma $T(n-1) = T(n-2) + \Theta(1)$

Sostituiamo: $T(n) = T(n-2) + 2 \cdot \Theta(1)$

ma $T(n-2) = T(n-3) + \Theta(1)$

Sostituiamo: $T(n) = T(n-3) + 3 \cdot \Theta(1)$

ma $T(n-3) = T(n-4) + \Theta(1)$

Sostituiamo: $T(n) = T(n-4) + 4 \cdot \Theta(1)$, ecc.

fino a ottenere:

$T(n) = n \cdot \Theta(1) = \Theta(n)$.

Metodo iterativo

Esempio: Equazione relativa alla ricerca sequenziale ricorsiva (seconda versione):

$$T(n) = 2 \cdot T(n/2) + \Theta(1)$$
$$T(1) = \Theta(1)$$

$T(n) = 2 \cdot T(n/2) + \Theta(1)$

ma $T(n/2) = 2T(n/2^2) + \Theta(1)$

Sostituiamo: $T(n) = 2[2T(n/2^2) + \Theta(1)] + \Theta(1) = 2^2 T(n/2^2) + 2 \cdot \Theta(1) + \Theta(1) = \ldots$

$$\ldots = 2^k T(n/2^k) + \sum_{i=0}^{k-1} 2^i \cdot \Theta(1) = \ldots$$

… proseguo finché $k = \log n$ …

$$\ldots = 2^{\log n}\Theta(1) + \sum_{i=0}^{\log n - 1} 2^i \Theta(1) = n \cdot \Theta(1) + (n-1) \cdot \Theta(1) = \Theta(n)$$

Dove ho usato: $\sum_{i=0}^{x} 2^i = 2^{x+1} - 1$

Metodo iterativo

A volte le cose sono un po' più complicate….

Esempio: Equazione relativa al calcolo dell'ennesimo numero di Fibonacci:

$$T(n) = T(n-1) + T(n-2) + \Theta(1)$$
$$T(0) = T(1) = \Theta(1)$$

A partire da $T(n) = T(n-1) + T(n-2) + \Theta(1)$ abbiamo:

- $T(n) \leq 2 \cdot T(n-1) + \Theta(1)$ questa disuguaglianza ci permetterà di derivare un limite; superiore O.
- $T(n) \geq 2 \cdot T(n-2) + \Theta(1)$ questa disuguaglianza ci permetterà di derivare un limite inferiore Ω.

Metodo iterativo

segue esempio: Equazione relativa al calcolo dell'ennesimo numero di Fibonacci:

$$T(n) \leq 2 \cdot T(n-1) + \Theta(1)$$
$$T(0) = T(1) = \Theta(1)$$

$$\begin{aligned} T(n) &\leq 2T(n-1) + \Theta(1) \quad \leq 2(2T(n-2) + \Theta(1)) + \Theta(1) \\ &\leq 2^2 T(n-2) + 2^1\Theta(1) \quad \leq 2^2(2T(n-3) + \Theta(1)) + 2^1\Theta(1) \\ &\leq 2^3 T(n-3) + 2^2\Theta(1) + 2^1\Theta(1) + \Theta(1) \\ &\cdots\cdots \\ &\leq 2^k T(n-k) + \sum_{i=0}^{k-1} 2^i \Theta(1) \end{aligned}$$

il procedimento si ferma quando $n - k = 1$ vale a dire k=n-1 e si ottiene:

$$\begin{aligned} &\leq 2^{n-1}\Theta(1) + \sum_{i=0}^{n-2} 2^i \Theta(1) \\ &= \Theta(2^n) + (2^{n-1} - 1)\Theta(1) = \Theta(2^n) \end{aligned}$$

E dunque troviamo $T(n) = O(2^n)$

Metodo iterativo

segue esempio: Equazione relativa al calcolo dell'ennesimo numero di Fibonacci:

$$T(n) \geq 2 \cdot T(n-2) + \Theta(1)$$
$$T(0) = T(1) = \Theta(1)$$

$$\begin{aligned} T(n) &\geq 2T(n-2) + \Theta(1) \geq 2(2T(n-4) + \Theta(1)) + \Theta(1) \\ &\geq 2^2 T(n-4) + 2^1\Theta(n) \geq 2^2(2T(n-6) + \Theta(1)) + 2^1\Theta(1) \\ &\geq 2^3 T(n-6) + 2^2\Theta(1) + 2^1\Theta(1) + \Theta(1) \\ &\cdots\cdots \\ &\geq 2^k T(n-2k) + \sum_{i=0}^{k-1} 2^i \Theta(1) \end{aligned}$$

il procedimento si ferma quando $n - 2k = 0$ se n è pari e per $n - 2k = 1$ se n è dispari. il comportamento asintotico non cambia assumiamo n pari e otteniamo:

$$\geq 2^{\frac{n}{2}}\Theta(1) + \sum_{i=0}^{\frac{n}{2}-1} 2^i \Theta(1) = \Theta(2^{\frac{n}{2}}) + (2^{\frac{n}{2}} - 1)\Theta(1) = \Theta\left(2^{\frac{n}{2}}\right)$$

E dunque troviamo $T(n) = \Omega\left(2^{\frac{n}{2}}\right)$

Metodo iterativo

segue esempio: Equazione relativa al calcolo dell'ennesimo numero di Fibonacci:

$$T(n) = T(n-1) + T(n-2) + \Theta(1)$$
$$T(0) = T(1) = \Theta(1)$$

- Abbiamo quindi scoperto che $T(n) = O(2^n)$ e $T(n) = \Omega\left(2^{\frac{n}{2}}\right)$
- Anche se non abbiamo trovato la crescita asintotica precisa (vale a dire il Θ) di $T(n)$ possiamo comunque concludere che il calcolo dei numeri di Fibonacci con la tecnica ricorsiva richiede un tempo esponenziale in n visto che esistono due costanti k_1 e k_2 tali che:

$$k_1 \cdot 2^{\frac{n}{2}} \leq T(n) \leq k_2 \cdot 2^n$$

Metodo di sostituzione

Idea:

- si ipotizza una soluzione per l'equazione di ricorrenza data;
- si verifica se essa "funziona".

Difficoltà:

- si deve trovare la funzione più vicina alla vera soluzione, perché tutte le funzioni più grandi (se stiamo cercando *O*) o più piccole (se stiamo cercando Ω) funzionano.
- 'indovinare' la giusta funzione richiede una grande intuizione

Metodo di sostituzione

Esempio: Equazione relativa alla ricerca sequenziale ricorsiva

- $T(n) = T(n-1) + \Theta(1)$
- $T(1) = \Theta(1)$

- Nell'applicare questo metodo, è necessario eliminare la notazione asintotica dall'equazione di ricorrenza, così che le costanti non rimangano "nascoste" nella notazione, conducendo a risultati errati

  - $T(n) = T(n-1) + b$
  - $T(1) = a$

    per due costanti *a* e *b*.

Metodo di sostituzione

segue Esempio: Equazione relativa alla ricerca sequenziale ricorsiva

- $T(n) = T(n-1) + b$
- $T(1) = a$

- Ipotizziamo la soluzione $T(n) = O(n)$, *ossia* $T(n) \leq c \cdot n$ dove *c* è una costante che va ancora determinata.
- N.B. non si può sperare in una soluzione esatta, ma possiamo solo maggiorare.
- Sostituiamo nel caso base, dovrebbe aversi $T(1) = a \leq c$ e la disuguaglianza è soddisfatta se $c \geq a$.
- Sostituiamo nella formulazione ricorsiva dell'equazione di ricorrenza, dovrebbe aversi: $T(n) \leq c(n-1) + b = c \cdot n - c + b \leq c \cdot n$ dove l'ultima diseguaglianza risulta vera se $-c + b \leq 0$ vale a dire $c \geq b$.
- Deduciamo dunque che *T(n)* è effettivamente un *O(n)* poiché è possibile trovare in valore *c* per il quale si ha $T(n) \leq c \cdot n$ (basta prendere *c* ≥ *max(a,b)*).

Metodo di sostituzione

segue Esempio: Equazione relativa alla ricerca sequenziale ricorsiva

- $T(n) = T(n-1) + b$
- $T(1) = a$

  per due costanti *c* e *d* fissate.

- Che cosa sarebbe successo se avessimo ipotizzato $T(n) \leq c \cdot n^2$?
- Anche questa soluzione è corretta per tutti i valori di *c* tali che $c \geq max(a, b)$.
- A noi interessa stimare *T(n)* asintoticamente tramite la funzione più piccola, e questo è spesso un obiettivo difficile.

Metodo di sostituzione

segue Esempio: Equazione relativa alla ricerca sequenziale ricorsiva

- $T(n) = T(n-1) + b$
- $T(1) = a$

per due costanti *a* e *b*.

- Per rendere il nostro risultato stretto, ipotizziamo ora la soluzione $T(n) = \Omega(n)$, *ossia* $T(n) \geq c \cdot n$ dove *c* è una costante che va ancora determinata.
- In modo assolutamente analogo al caso *O grande,* sostituiamo nel caso base, dovrebbe avevi $T(1) = a \geq c$ che è soddisfatta per $c \leq a$.
- Sostituiamo nella formulazione ricorsiva dell'equazione di ricorrenza ottenendo $T(n) \geq c(n-1) + b = c \cdot n - c + b \geq c \cdot n$ dove l'ultima diseguaglianza risulta vera se $-c + b \geq 0$ vale a dire $c \leq b$.
- Deduciamo dunque che *T(n)* è effettivamente un $\Omega(n)$ poiché è possibile trovare un valore *c per il quale si ha* $T(n) \leq c \cdot n$ *(basta prendere c* $\leq$ *min(a,b))*.

Metodo di sostituzione

segue Esempio: Equazione relativa alla ricerca sequenziale ricorsiva

- $T(n) = T(n-1) + b$
- $T(1) = a$

per due costanti *a* e *b*.

Dalle due soluzioni: $T(n) = O(n)$ e $T(n) = \Omega(n)$, si ottiene ovviamente che $T(n) = \Theta(n)$.

Metodo di sostituzione

Esempio.

- $T(n) = 2 \cdot T(n/2) + \Theta(1)$
- $T(1) = \Theta(1)$

Dobbiamo eliminare per prima cosa la notazione asintotica:

- $T(n) = 2 \cdot T(n/2) + b$
- $T(1) = a$ per due costanti positive a e b

Ipotizziamo la soluzione $T(n) = O(n)$ vale a dire $T(n) \leq c \cdot n$ per qualche c **da determinare** e sostituiamo:

- nel caso base: a ≤ *c*
- nel caso generale: $T(n) \leq 2 \cdot c \cdot \frac{n}{2} + b = c \cdot n + b$ **che non è MAI** $\leq c \cdot n \ldots$

Vuol dire che l'ipotesi è errata? non sempre:

Metodo di sostituzione

segue Esempio.

- T(n)=2T(n/2)+*b*
- T(1)=*a*

Ipotizziamo la nuova soluzione $T(n) \leq c \cdot n - d$ per qualche *c* ed *d* **da determinare** e sostituiamo:

- nel caso base: $a \leq c - d$
- nel caso generale:

$$T(n) \leq 2\left(c\frac{n}{2} - d\right) + b = c \cdot n - 2d + b \leq c \cdot n - d$$

che è verificato per $-b + d \leq 0$ *vale a dire* $d \leq b$.

- *Non è difficile trovare valori di c e d per cui valga* $a \leq c - d$ *e* $d \leq b$ *(basta ad esempio prendere d=b e c=a+b)*

Da qui deduciamo che *T(n)=O(n).*

PER ESERCIZIO: Poi dobbiamo dimostrare anche che

Metodo dell'albero

Idea:

- rappresentare graficamente lo sviluppo del costo computazionale dell'algoritmo, in modo da poterlo valutare con una certa facilità

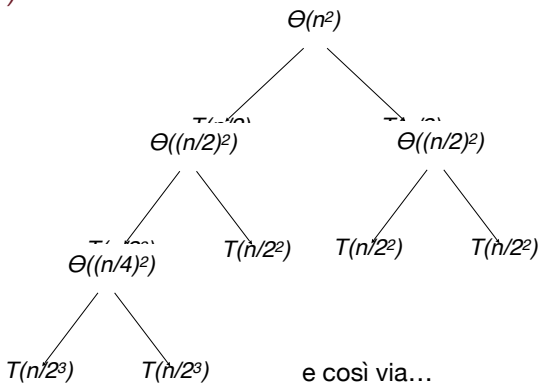
Difficoltà:

- come il metodo iterativo

Metodo dell'albero

Esempio:

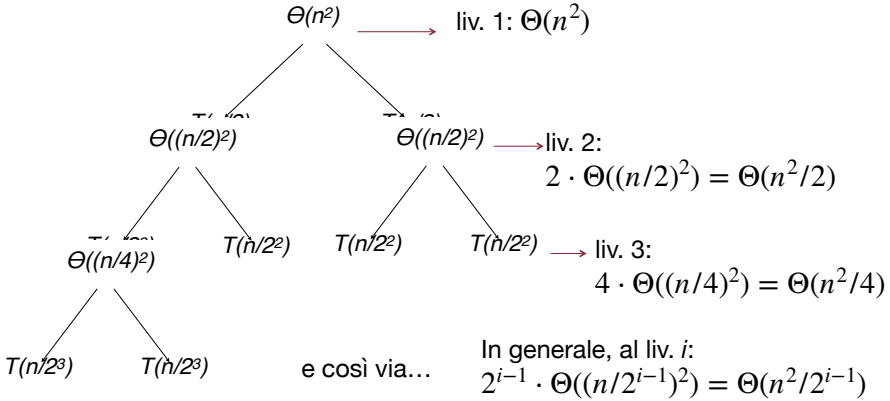
- $T(n) = 2T(n/2) + \Theta(n^2)$
- $T(1) = \Theta(1)$



Metodo dell'albero

segue Esempio:

Una volta completato l'albero, il costo computazionale è dato dalla somma dei contributi di tutti i livelli (cioè le "righe" in cui sono disposti i nodi) di cui è costituito l'albero.



Metodo dell'albero

segue Esempio:

In questo caso il numero di livelli dell'albero ha un valore tale che $\frac{n}{2^{i-1}} = 1,$

ossia $i - 1 = \log n$

da cui $i = \log n + 1.$

Sommiamo i contributi di tutti i livelli:

$$\sum_{i=1}^{\log n} \Theta\left(\frac{n^2}{2^{i-1}}\right) = n^2 \sum_{j=0}^{\log n - 1} \Theta\left(\frac{1}{2^j}\right) = \Theta(n^2)$$

Metodo principale

Idea:

- avere una "ricetta" meccanica per risolvere un'equazione di ricorrenza

Difficoltà:

- funziona solo quando l'equazione è della forma $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ con $T(1) = \Theta(1)$

Metodo principale

Enunciato del teorema principale:

Dati $a \geq 1$, $b > 1$, una funzione asintoticamente positiva *f(n)* ed un'equazione di ricorrenza della forma:

$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ e $T(1) = \Theta(1)$ vale che:

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

Metodo principale

Il teorema principale ci dice che in ciascuno dei tre casi vengono confrontati fra loro *f(n)* e $n^{log_b a}$ ed il costo computazionale è governato dal maggiore dei due:

- se (caso 1) il più grande dei due è $n^{log_b a}$, allora il costo è $\Theta(n^{\log_b a})$;
- se (caso 2) sono uguali, allora si moltiplica *f(n)* per un fattore logaritmico.
- se (caso 3) il più grande dei due è *f(n)*, allora il costo è $\Theta(f(n))$;

Metodo principale

Si noti che "più grande" e "più piccolo" in questo contesto significa ***polinomialmente*** più grande (o più piccolo), data la posizione all'esponente di ϵ. In altre parole, *f(n)* deve essere asintoticamente più grande (o più piccola) rispetto a $n^{log_b a}$ di un fattore n^{ϵ} per qualche $\epsilon > 0$.

In effetti fra i casi 1 e 2 vi è un intervallo in cui *f(n)* è più piccola di $n^{log_b a}$, ma non polinomialmente. Analogamente, fra i casi 2 e 3 vi è un intervallo in cui *f(n)* è più grande di $n^{log_b a}$, ma non polinomialmente.

Metodo principale – caso 1
Esempio.

$T(n) = 9T(n/3) + \Theta(n)$ e $T(1) = \Theta(1)$

- $a = 9,\ b = 3,\ f(n) = \Theta(n),$

  $n^{\log_b a} = n^{log_3 9} = n^2$

Poiché $f(n) = \Theta\left(n^{\log_3 9 - \epsilon}\right)$ con $\epsilon = 1$, siamo nel **caso 1**, per cui $T(n) = \Theta(n^{log_b a}) = \Theta(n^2)$.

Metodo principale – caso 2
Esempio.

$T(n) = T\left(\frac{2n}{3}\right) + \Theta(1)$ e $T(1) = \Theta(1)$

- $a = 1,\ b = \frac{3}{2},\ f(n) = \Theta(1)$

  $n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = n^0 = 1$

Poiché $f(n) = \Theta(n^{\log_b a)} = \Theta(n^{log_b a})$ siamo nel **caso 2**, per cui $T(n) = \Theta(\ n^{\log_b a} \log\ n\ ) = \Theta(\log\ n)$.

Metodo principale – caso 3
Esempio.

$T(n) = 3 \cdot T\left(\frac{n}{4}\right) + \Theta(n \log\ n)$ e $T(1) = \Theta(1)$

- $a = 3,\ b = 4,\ f(n) = \Theta(nlog\, n)$
- $n^{log_b a} = n^{log_4 3} \approx n^{0,7}$

Poiché $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ con, ad esempio, $\epsilon = 0.2$ siamo nel **caso 3** se possiamo dimostrare che $3\frac{n}{4} \log \frac{n}{4} \leq c \cdot n \log\ n$, per qualche $c < 1$ ed n abbastanza grande.

Ponendo $c = \frac{3}{4}$ otteniamo: $\quad 3\frac{n}{4} \log \frac{n}{4} \leq \frac{3}{4} n \log\ n$

che è vera, quindi $T(n) = \Theta(n \log n)$.

Metodo principale
Esempio.

$T(n) = 2T(n/2) + \Theta(n \log\ n)$ e $T(1) = \Theta(1)$

- $a = 2,\ b = 2,\ f(n) = \Theta(n \log\ n)$

  $n^{log_b a} = n^{log_2 2} = n$

Ora, $f(n) = \Theta(n \log\ n)$ è asintoticamente più grande di $n^{log_b a} = n$, ma non polinomialmente più grande. Infatti, $\log\ n$ è asintoticamente minore di n^{ϵ} per qualunque valore di $\epsilon > 0$. Di conseguenza non possiamo applicare il metodo del teorema principale.

Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizi

Calcolare la soluzione delle seguenti equazioni di ricorrenza con tutti e quattro i metodi ove possibile:

- $T(n)=2T(n/2)+\Theta(n)$ $T(1)=\Theta(1)$
- $T(n)=3T(n/2)+\Theta(n)$ $T(1)=\Theta(1)$
- $T(n)=3T(n/4)+\Theta(n)$ $T(1)=\Theta(1)$
- $T(n)=2T(n/2)+\Theta(n)$ $T(1)=\Theta(1)$
- $T(n)=4T(n/2)+\Theta(n)$ $T(1)=\Theta(1)$
- $T(n)=2T(n/2)+\Theta(n^3)$ $T(1)=\Theta(1)$
- $T(n)=16T(n/4)+\Theta(n^2)$ $T(1)=\Theta(1)$
- $T(n)=T(n-1)+\Theta(n)$ $T(1)=\Theta(1)$
- $T(n)=3T(n/2)+\Theta(n \log n)$ $T(1)=\Theta(1)$