Algoritmi e Strutture Dati

Alex Narder

October 19, 2022

Contents

1 Contenuti del corso

In questo corso si parlerà di:

- Algoritmi e le loro complessità; capire il comportamento asintotico
- Ricorrenze
- Grafi
- Alberi di copertura minimi
- Problema dei cammini minimi
- Algoritmi greedy

2 Introduzione

Un **algoritmo** per essere utile deve essere funzionale e veloce. Quindi deve essere **ottimizzato**, nel momento in cui il tempo di esecuzione è esponenziale allora l'algoritmo non è ottimizzato.

Ci sono alcuni problemi per cui non c'è la speranza di trovare algoritmi efficenti. L'obiettivo di questo corso è studiare algoritmi non troppo complicati, che quindi non saranno esponenziali. Quando vado a misurare la complessità di un algoritmo la misuro rispetto alla dimensione dell'input dato.
In alcune situazioni è molto utile capire di che tipo di input si parla, per analizzare il problema e trarre delle conclusioni.

Parlando di complessità si usa sempre il **worst case**, ovvero il caso peggiore, mentre tutti gli altri casi saranno **best case** e **avarage case**.

3 Numeri di Fibonacci

Problema:
Scrivere un algoritmo che restituisca in uscita i numeri di Fibonacci.

La sequenza di Fibonacci è un insieme di numeri, la sequenza è definita ricorsivamente;

$$F_n = \begin{cases} 1, \ se\ n = 1, 2 \\ F_{n-1} + F_{n-2}\ se\ n >= 3 \end{cases}$$

F_n rappresenta l'iesimo numero di Fibonacci.

4 Versioni Algoritmo di Fibonacci

4.1 Versione 1 Algoritmo di Fibonacci

La sequenza di Fibonacci è direttamente collegata alla sezione aurea.
Formula di **Binet**;

$x^2 = x + 1$

$x^2 - x - 1 = 0$
$ax^2 + bx + c = 0 \rightarrow$ ci sono 2 soluzioni $x_{1,2}$:
x1 = 1,618... che chiamiamo ϕ
x2 = 0,618... che chiamiamo $\hat{\phi}$

La formula di Binet dice che per ogni n maggiore o uguale a 1 risulta

$F_n = \frac{1}{\sqrt{5}} * (\phi^n - \hat{\phi}\ ^n)$

Dimostrare la formula di Binet: (induzione su n)

Base $\rightarrow$ n = 1,2

n = 1 $\rightarrow F_1 = \frac{1}{\sqrt{5}} * (\phi - \hat{\phi}\ )$

$\frac{1}{\sqrt{5}} * (\frac{1+\sqrt{5}}{2} - \frac{1-\sqrt{5}}{2)} = \mathbf{1}$

n = 2 $\rightarrow F_2 = 1/\sqrt{5} * (\phi^2 - \hat{\phi}^2)$

$\frac{1}{\sqrt{5}} * (\frac{1+\sqrt{5}}{2} - \frac{1-\sqrt{5}}{2}) = \mathbf{1}$

Ipotesi induttiva:

def. $\rightarrow F_n = F_{n-1} + F_{n-2}$

$F_{n-1} = \frac{1}{\sqrt{5}} * (\phi^{n-1} - \hat{\phi}\ ^{n-1}) \quad +$

$F_{n-2} = \frac{1}{\sqrt{5}} * (\phi^{n-2} - \hat{\phi}\ ^{n-2}) \quad =$

$\frac{1}{\sqrt{5}} * [(\phi^{n-1} + \phi^{n-2}) - (\hat{\phi}\ ^{n-1} + \hat{\phi}\ ^{n-2})]$

$$\begin{cases} \phi^n = \phi^{n-1} + \phi^{n-2} \\ \hat{\phi}\ ^n = \hat{\phi}\ ^{n-1} + \hat{\phi}\ ^{n-2} \end{cases}$$

è vera per la definizione di ϕ e $\hat{\phi}$

```
int Fib1(int n){
   return 1/sqrt(5) * (phi^n - phi^^n);
}
```

facendo i calcoli l'argoritmo sembra corretto, ma al numero n = 18 il risultato è errato.

4.2 Versione 2 Algoritmo di Fibonacci

```
int Fib2(int n){
   if (n<=2) {
      return 1;
   } else return Fib2(n-1) + Fib2(n-2);
}
```

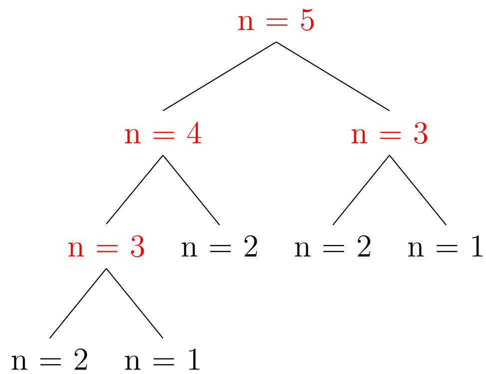
n	T(n)
1	1
2	1
3	2+1+1 = 4
4	2+4+1 = 7

n è il numero, T(n) invece è il numero della istruzioni necessarie prima di ricevere un output.

$$\mathrm{T(n)} = \begin{cases} 1 \; se \; n = 1, 2 \\ 2 + T(n-1) + T(n-2) \; se \; n >= 3 \end{cases}$$

$T(n) = 2 + T(n-1) + T(n-2) \rightarrow$ quando $n >= 3$

Albero delle ricorsioni di n = 5:



La complessità dei pallini in **rosso** è 2 mentre quella degli altri è 1. Sommando tutte le complessità otteniamo il numero di istruzioni necessarie per eseguire, in questo caso 13.

In **rosso** ci sono i **nodi interni**
Mentre in **nero** ci sono i **nodi foglia**

$T(5) = 4*2+5*1 \rightarrow$ è dato da un numero di nodi interi (i) moltiplicati per 2 + il numero di foglie (f)

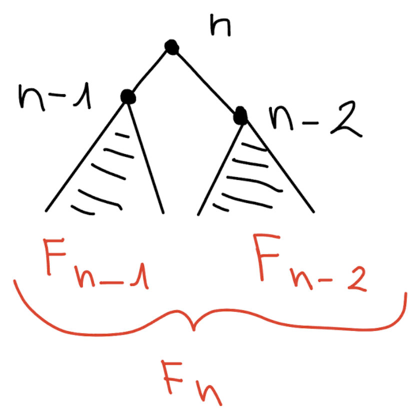
$\rightarrow T(n) = 2*i(Tn) + 1*f(Tn)$

4.2.1 Proposizione 1

Dato un generico albero di ricorsione, siamo in grado di arrivare a una formula che ci dica esattamente quanti nodi interni e foglia ci sono?

→ Sia T(n) l'albero di ricorsione relativo alla chiamata di Fib2(n),
allora: f(Tn) = F(n) → infatti il numero delle foglie è l'esimo numero di Fibonacci.

$$\begin{cases} n = 1, 2 \\ n \geq 3 \end{cases}$$



4.2.2 Proposizione 2

Sia T un albero dove i nodi interni hanno esattamente 2 figli, allora:
Il numero di nodi interni è sempre uguale a f(T) - 1:

$$i(T) = f(t) - 1$$

Da dimostrare.
Quindi con la proposizione uno e la proposizione 2 otteniamo:
T(n) = 2(F(n) - 1) + F(n) = 3F(n) - 2

4.2.3 Proposizione 3

Per ogni $n >= 6$ abbiamo che $F(n) >= 2^{n/2}$

Da dimostrare

Base: n = 6,7

I.P.: n >= 8

$$\begin{aligned} F_n &= F_{n-1} + F_{n-2} \\ &>= 2^{(n-1)/2} + 2^{(n-2)/2} \\ &. \\ &. \\ &. \\ &>= 2^{n/2}(\frac{1}{\sqrt{2}} + \frac{1}{2}) >= 2^{n/2} \end{aligned}$$

4.3 Versione 3 Algoritmo di Fibonacci

```
int Fib3(int n){

}
```

Complessità? T(n) è più o meno uguale a n

4.4 Versione 4 Algoritmo di Fibonacci

```
int Fib4(int n){
   int a = 1, b = 2;
   for (int i = 3 ; i <= n ; i++){
      c = a+b;
      a = b;
      b = c;
   }
   return b;
}
```

4.5 In sostanza

tabella:

	corretto?	**complessità temporale**	**complessità spaziale**
Fib1	no	costante	costante
Fib2	si	esponenziale	lineare
Fib3	si	lineare	lineare
Fib4	si	lineare	lneare

L'algoritmo Fib4 è il più efficente.

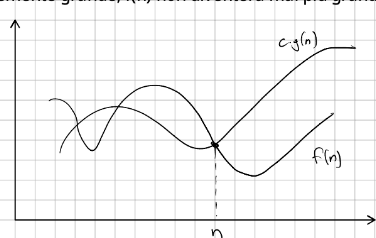
5 Classi asintotiche

5.1 Prima classe

$O(g(n)) = \{f(n) \mid \exists\, c > 0$
$\exists\; un\; n_0 \in N \mid \forall n >= n_0 :$
$f(n) <= c * g(n)\}$

Data una funzione g(n), la notazione O(g(n)) indica l'insieme di tutte le funzioni f(n) che soddisfano la proprietà che, data una costante positiva e un n_0 appartenente all'insieme dei numeri naturali tale che n >= n_0, f(n) <= c g(n).

Quando n diventa sufficientemente grande, f(n) non diventerà mai più grande di c g(n).



Esempio: $\frac{1}{2}n^2 - 3n = O(n^2)$

$\exists c > 0,\ \exists n_0 \in \mathbb{N} \ni \forall n \geq n_0 \Rightarrow$

$\Rightarrow \frac{1}{2}n^2 - 3n \leq c \cdot n^2$

$\Leftrightarrow \frac{1}{2}n - 3 \leq c \cdot n$

$\Leftrightarrow \frac{1}{2}n - cn \leq 3$

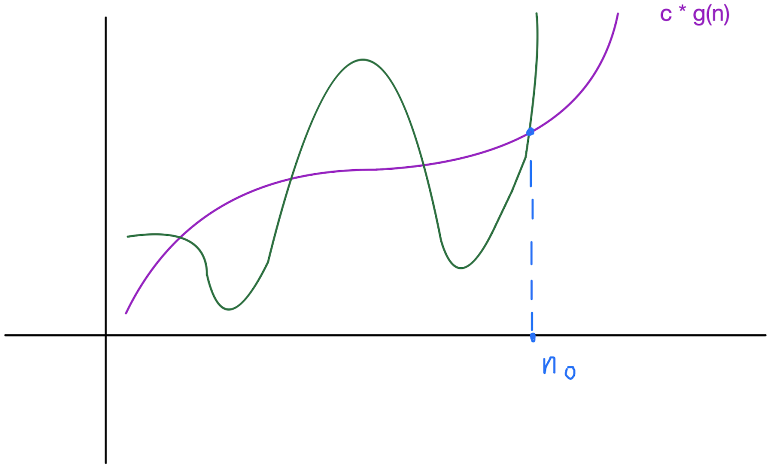
$\Leftrightarrow \underbrace{\left(\frac{1}{2} - c\right)}_{\leq 0} n \leq 3$

$\leq 0 \Rightarrow \quad c \geq \frac{1}{2} > 0 \qquad \forall n \geq 1$

Quindi: $\boxed{c = \frac{1}{2}}$ $\boxed{n_0 = 1}$

5.2 Seconda classe

$\Omega(g(n)) = \{f(n) \mid \exists\, c > 0$
$\exists\ un\ n_0 \in N \mid \forall n >= n_0 :$
$f(n) <= c * g(n)\}$



$\frac{1}{2}n^2 - 3n = \Omega(n^2)$

$$\begin{aligned} \exists c > 0 \quad & \exists n_0 \in N \mid \forall n >= n_0 : \\ & c * n^2 <= \frac{1}{2}n^2 - 3n \\ & c * n <= \frac{1}{2}n - 3 \\ & n(\frac{1}{2} - c) >= 3 \end{aligned}$$

Supponendo che $\frac{1}{2} - c > 0$ $[c < \frac{1}{2}]$
si ha:

$$n >= \frac{3}{\frac{1}{2} - c} = \frac{6}{1 - 2c}$$

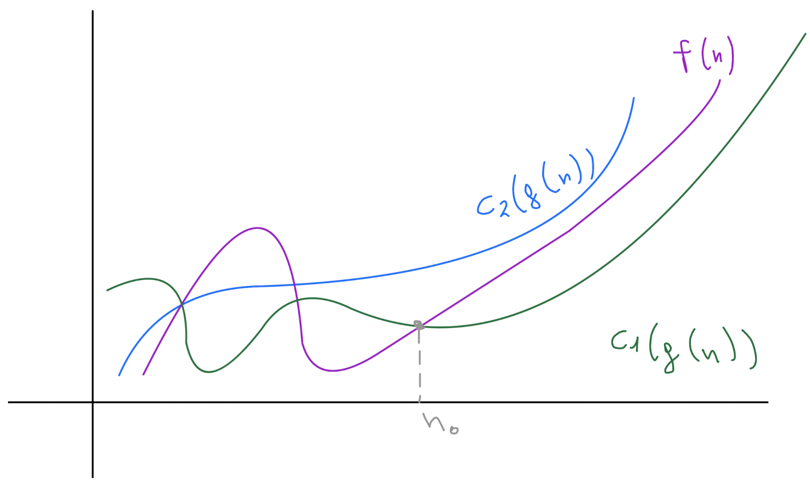
$\rightarrow$ scegliamo $c = \frac{1}{14}$

si ottiene: $n >= \frac{6}{1 - \frac{1}{7}} = 7$

5.3 Terza classe

$$\Theta(g(n)) = \{f(n) \mid \exists c_1 > 0,\ \exists c_2 > 0$$
$$\exists n_0 \in N \mid \forall n >= n_0 :$$
$$c_1\ g(n) <= f(n) <= c_2\ g(n)\}$$

notazione: $f(n) = \Theta(g(n))$



<u>proprietà:</u>

$$f(n) = \Theta(g(n)) \longleftrightarrow f(n) = O(g(n))$$
$$f(n) = \Omega(g(n))$$

quindi: $\frac{1}{2}n^2 - 3n = \theta(n^2)$

$\sqrt{n+10} = \theta(\sqrt{n})$

$$\exists c_1, c_2 > 0 \quad \exists n_0 \in N \mid \forall >= n_0 \ :$$
$$c_1\sqrt{n} <= \sqrt{n+10} <= c_2\sqrt{n} \longleftrightarrow$$
$$c_1^2 n <= n + 10 <= c_2^2 n$$

1) $c_1^2 n <= n + 10 \longleftrightarrow$

$(c_1^2 - 1)n <= 10$

se $c_1^2 - 1 <= 0 \quad (c_1^2 <= 1, c_1 <= 1)$

diventa vera $\forall n >= 1 \qquad \rightarrow c_1 = 1$

2) $n + 10 <= c_2^2 n \longleftrightarrow$

$n(c_2^2 - 1) >= 10$

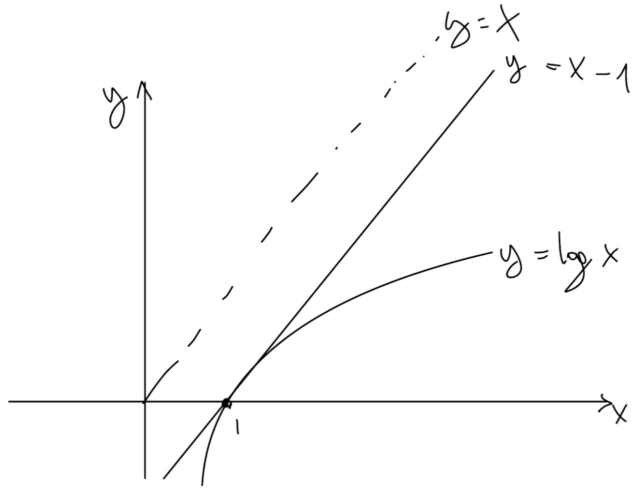
se $c_2 > 1$:

$n >= \frac{10}{c_2^2 - 1}$

poniamo $c_2 = \sqrt{2}(> 1)$
si ha $n >= 10$
quindi:
$c_1 = 1$
$c_2 = \sqrt{2}$
$n_0 = 10$

5.3.1 esempi:

1) $\log n = O(n)$



2) $n \log n = O(n^2)$
$\forall n >= 1 : \log n <= n \longleftrightarrow$
$n \log n <= n^2$

c = 1 , $n_0 = 1$

3) n! $= O(n^n)$

infatti:

$$n! = 1 * 2 * 3 * ...n$$
$$<= n * n * n * ...n$$
$$= n^n$$

5.4 Quarta classe

$o(g(n)) = \{f(n) \mid \forall c > 0, \exists n \in N \text{ tale che } \forall n >= n_0 : f(n) < c * g(n)\}$

osservazione:
$o(g(n)) \subset O(g(n))$
quindi:
$f(n) = o(g(n)) \rightarrow f(n) = O(g(n))$

proprietà:

$f(n) = o(g(n)) \longleftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

esempio d'uso:

$\log(n) = O(\sqrt{n})$

qui per semplicità assumiamo che la base del log è e

$$\lim_{n\to\infty} \frac{\log(n)}{\sqrt{n}} = \left[\frac{\infty}{\infty}\right] \overset{H}{=} \lim_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2} n^{-1/2}} =$$

$$= \lim_{n\to\infty} \frac{2\sqrt{n}}{n} = \lim_{n\to\infty}$$

5.5 Quinta classe

$w(g(n)) = \{f(n) \mid \forall c > 0, \exists n_0 \in N$ tale che $\forall n >= n_0 : c * g(n) < f(n)\}$

osservazione:
$w(g(n)) \subset \Omega(g(n))$
$f(n) = w(g(n)) \rightarrow f(n) = \Omega(g(n))$

proprietà:

$f(n) = w(g(n)) \longleftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty$