Algoritmi - Lezione 1

Ionut Zbirciog

6 October 2023

1 Algoritmi per risolvere Fibonacci

1.1 fib1 - Formula chiusa

Uso la formula chiusa:

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

Algorithm 1 fibonacci1

```
function FIBONACCI2(intero n) → intero
    phi = 1.618
    phiSegnato = −0.618
    return 0.447 ∗ phi^n − phiSegnato^n
end function
```

1.2 fib2 - Ricorsione

Algorithm 2 fibonacci2

```
function FIBONACCI2(intero n) → intero
    if n ≤ 2 then
        return 1
    else
        return FIBONACCI2(n − 1) + FIBONACCI2(n − 2)
    end if
end function
```

Costo:

$$T(n) = 2 + T(n-1) + T(n-2)$$
$$T(1) = T(2) = 1$$

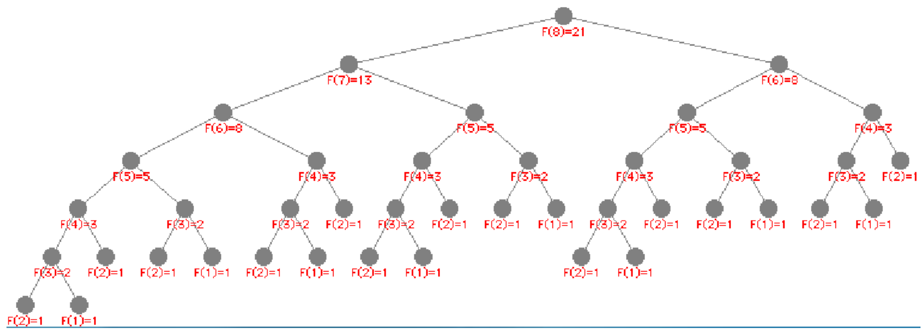

Figure 1: Albero della ricorsione di fibonacci2.

1.3 Albero della Ricorsione

- Utile per risolvere la relazione di ricorrenza.
- Nodi corrispondenti alle chiamate ricorsive.
- Figli di un nodo corrispondono alle chiamate.
- Il primo nodo è la radice.
- I nodi interni hanno etichetta 2.
- Le foglie hanno etichetta 1.

1.3.1 Calcolo di $T(n)$

Per calcolare $T(n)$:

- Contiamo il numero di foglie.
- Contiamo il numero di nodi interni.

1.3.2 Lemma 1

Il numero di foglie dell'albero della ricorsione di Fibonacci è pari a F_n.

1.3.3 Lemma 2

Il numero di nodi interni di un albero in cui ogni nodo interno ha due figli è pari al numero di foglie - 1. In totale, otteniamo:

$$T(n) = F_n + 2(F_{n-1}) = 3F_n - 2 = F_n = \phi^n$$

È lento perché continua a ricalcolare ripetutamente la soluzione dello stesso sottoproblema (crescita esponenziale).

Algorithm 3 fibonacci3

```
function FIBONACCI3(intero n) → intero
    Sia Fib un array di n interi
    Fib[1] = 1; Fib[2] = 1
    for i = 3 to n do
        Fib[i] = Fib[i − 1] + Fib[i − 2]
    end for
    return Fib[n]
end function
```

1.4 fib3 - Memorizzazione in Array

$$T(n) = n + n + 3 = 2n + 3$$

Algoritmo più veloce rispetto a fibonacci2 di 38 milioni di volte con crescita lineare. Unica pecca: l'utilizzo della memoria, fib3 occupa in memoria uno spazio proporzionale a n.

1.5 fib4 - Memorizzazione degli ultimi 2 valori

Algorithm 4 fibonacci4

```
function FIBONACCI4(intero n) → intero
    a = 1; b = 1
    for i = 3 to n do
        c = a + b
        a = b
        b = c
    end for
    return c
end function
```

$$T(n) = 4n + 2$$

E' più lento di fib3? Andiamo a vedere.

1.6 Notazione Asintotica

- Esprimere $T(n)$ in modo QUALITATIVO.
- Perdere un po' in PRECISIONE ma guadagnare SEMPLICITÀ.
- Di $T(n)$ vogliamo descrivere come cresce al crescere di n:
  - Ignoro costanti moltiplicative.
  - Ignoro termini di ordine inferiore.

1.7 Esempi di Notazione Asintotica

- $T(n) = 5n + 9 = O(n)$
- $T(n) = 6n^2 + 8n - 13 = O(n^2)$

1.8 fib5 - Matrici

- fib4 non è il miglior algoritmo possibile.
- È possibile dimostrare per induzione la seguente proprietà di matrici:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

- Useremo questa proprietà per progettare un algoritmo più efficiente.

Lemma:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

Algorithm 5 fibonacci5

function FIBONACCI5(int n) $\to$ int
  $M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$
  for $i = 1$ to $n - 1$ **do**
    $M = M \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$
  end for
  return $M[0][0]$
end function

Il tempo di esecuzione è ancora $O(n)$ ma possiamo usare le proprietà delle potenze e ottenere un risultato migliore.

1.9 Calcolo di Potenze

- Possiamo calcolare la n-esima potenza elevando al quadrato la $\left(\frac{n}{2}\right)$-esima potenza.
- Se n è dispari, eseguiamo una ulteriore moltiplicazione.
- Esempio:

$$3^2 = 9,\ 3^4 = (9)^2 = 81,\ 3^8 = (81)^2 = 6561$$

  Abbiamo eseguito solo 3 prodotti invece di 7.

1.10 fib6

Costo: $T(n) = O(\log_2 n)$, quindi è esponenzialmente più veloce di fibonacci3.

Algorithm 6 fibonacci6

```
function FIBONACCI6(int n) → int
    A = [[1, 1], [1, 0]]
    M = potenzaDiMatrice(A, n − 1)
    return M[0][0]
end function
function POTENZADIMATRICE(matrice A, int k) → matrice
    if k = 0 then
        return [[1, 0], [0, 1]]
    else
        M = potenzaDiMatrice(A, k/2)
        M = M · M
    end if
    if k dispari then
        M = M · A
    end if
    return M
end function
```

1.11 Quanta memoria usa un algoritmo?

- Algoritmo non ricorsivo: dipende dalla memoria ausiliaria utilizzata (variabili, array, strutture dati).
- Algoritmo ricorsivo: dipende dalla memoria ausiliaria utilizzata da ogni chiamata e dal numero di chiamate che sono contemporaneamente attive.

Nota: Una chiamata usa sempre almeno memoria costante.

1.12 Analisi Memoria Ausiliaria Fibonacci2

- Chiamate attive formano un cammino P radice-nodo.
- P ha al più n nodi.
- Ogni nodo/chiamata usa memoria costante.

Risultato: Spazio $O(n)$.

1.13 Analisi Memoria Ausiliaria Fibonacci6

Risultato: Spazio $O(\log n)$.

1.14 Riepilogo

	Tempo di Esecuzione	Occupazione di Memoria
fibonacci2	$O(\phi^n)$	$O(n)$
fibonacci3	$O(n)$	$O(n)$
fibonacci4	$O(n)$	$O(1)$
fibonacci5	$O(n)$	$O(1)$
fibonacci6	$O(\log_2 n)$	$O(\log_2 n)$