Programmazione dinamica: primo esempio

Esercizi

25 marzo 2022

Programma in breve: siamo a 1/3

- Metodologie di analisi di algoritmi
- Tecniche di progettazione di algoritmi:
  1. Divide et impera
  2. Programmazione dinamica
  3. Tecnica greedy
  4. Algoritmi esaustivi
- Grafi e principali algoritmi su grafi

Dynamic Programming Applications

Areas.

- Bioinformatics.
- Control theory.
- Information theory.
- Operations research.
- Computer science: theory, graphics, AI, systems, ….

Some famous dynamic programming algorithms.

- Viterbi for hidden Markov models.
- Unix diff for comparing two files.
- Smith-Waterman for sequence alignment.
- **Bellman-Ford for shortest path routing in networks**.
- Cocke-Kasami-Younger for parsing context free grammars.

Programmazione dinamica e Divide et Impera

Entrambe le tecniche dividono il problema in **sottoproblemi**: dalle soluzioni dei sottoproblemi è possibile risalire alla soluzione del problema di partenza.

Divide et Impera dà luogo in modo naturale ad algoritmi ricorsivi.
La **programmazione dinamica** può dar luogo ad algoritmi:

- ricorsivi con annotazione delle soluzioni su una tabella
- iterativi

Idea chiave:
Qualora si presentino **sottoproblemi ripetuti** calcolare la soluzione di ogni sottoproblema 1 sola volta ed annotarla su una **tabella**

Primo esempio

- Un primo semplice esempio di programmazione dinamica: il calcolo dei numeri di Fibonacci che contiene gli ingredienti essenziali della tecnica.

- Puoi consultare [DFI] e/o le slides.

Leonardo Fibonacci

Nel **1223** a Pisa, l'imperatore Federico II di Svevia, fu ben felice di assistere a un singolare **torneo tra abachisti e algoritmisti**, armati soltanto di carta, penna e pallottoliere.
In quella gara infatti si dimostrò che col metodo posizionale indiano appreso dagli arabi si poteva calcolare più velocemente di qualsiasi abaco. Leonardo Fibonacci contribuì alla diffusione.

Il test era il seguente:
"Quante coppie di conigli si ottengono in un anno (salvo i casi di morte) supponendo che ogni coppia dia alla luce un'altra coppia ogni mese e che le coppie più giovani siano in grado di riprodursi già al secondo mese di vita?".

Un pisano, Leonardo, detto Bigollo, conosciuto anche col nome paterno di "fillio Bonacci" o Fibonacci, vinse la gara. Figlio d'un borghese uso a trafficare nel Mediterraneo, Leonardo visse fin da piccolo nei paesi arabi e apprese i principi dell'algebra, il calcolo, dai maestri di Algeri, cui era stato affidato dal padre, esperto computista.

L’isola dei conigli

- Una coppia di conigli genera un’altra coppia di conigli ogni mese
- I conigli cominciano a riprodursi dal secondo mese (tranne al più la coppia iniziale)
- I conigli sono immortali (!)

Il problema

- Una coppia di conigli genera un'altra coppia di conigli ogni mese
- I conigli cominciano a riprodursi dal secondo mese, anche la coppia iniziale
- I conigli sono immortali

F_n : numero di coppie di conigli presenti al mese n

F_1=1 (la coppia iniziale)

F_2=1 (la coppia iniziale è ancora troppo giovane)

F_3=2 (nasce una prima coppia da quella iniziale)

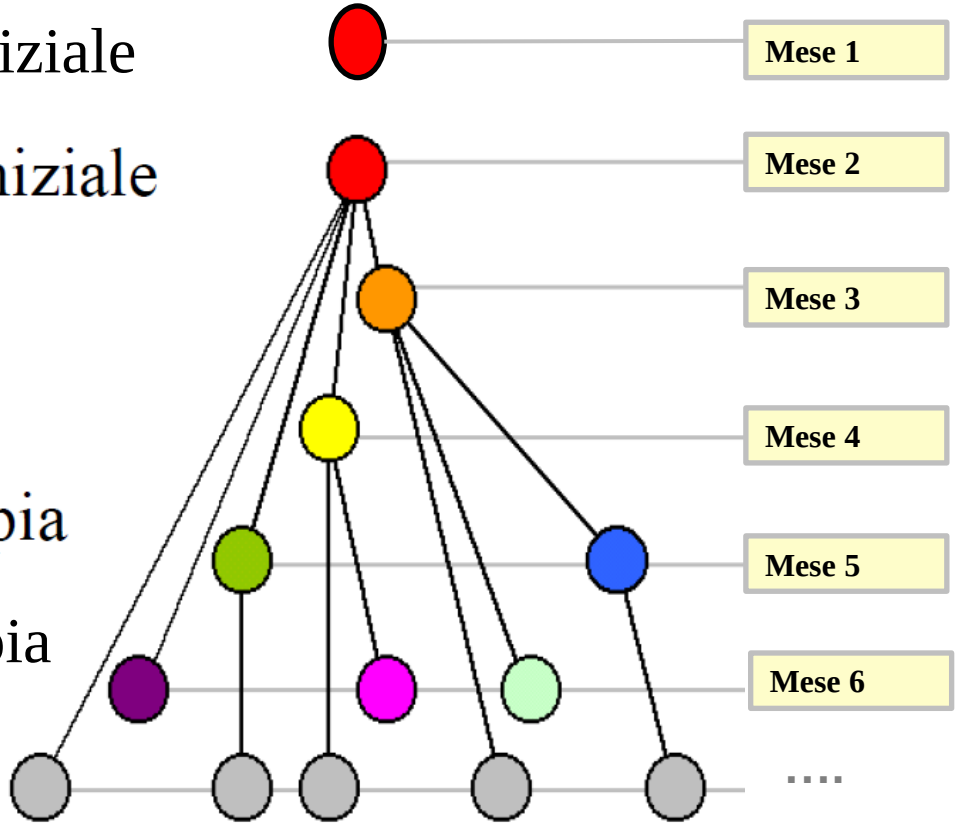
F_4=3 (nasce una seconda coppia da quella iniziale)

F_5=5 (nasce una terza coppia da quella iniziale e i nipotini!)

…

LA RIPRODUZIONE DEI CONIGLI

- 1° coppia iniziale
- 1° coppia iniziale
- 2° coppia
- 3° coppia
- 4° e 5° coppia
- 6°, 7° e 8° coppia
- ....



Le prime cinque generazioni di conigli secondo lo schema di Fibonacci. Ogni cerchio colorato rappresenta una coppia di conigli.

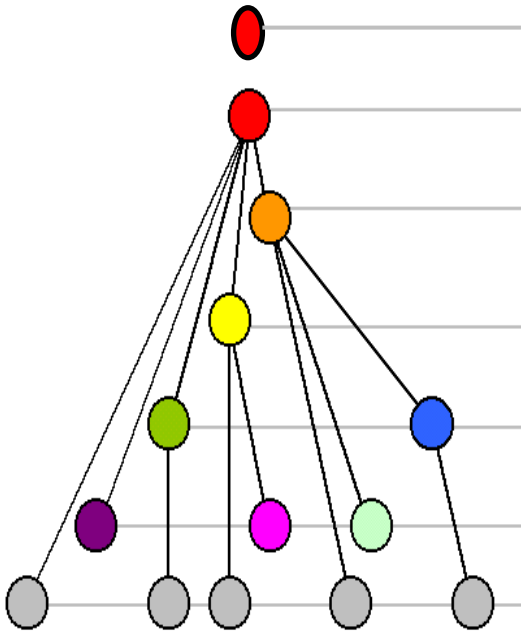
Conteggio dei conigli

Come calcolereste il numero di conigli presenti nel mese n?

Come risolse il problema Fibonacci nel 1223?

Ricorsivamente!
Espresse il problema in termini di sottoproblemi……

Il calcolo dei conigli presenti nel mese n può essere espresso rispetto al numero di conigli presenti nei due mesi precedenti



Un primo esempio: calcolo dei numeri di Fibonacci

Sequenza dei numeri di Fibonacci:

1, 1, 2, 3, 5, 8, 13, ….. ?

1, 1, 2, 3, 5, 8, 13, **21**, …

In generale il prossimo numero è la somma degli ultimi due. Come si formalizza?

Indicando con F_n (oppure $F(n)$) l' n-esimo numero,

abbiamo la seguente relazione di ricorrenza:

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

NOTA

Alcuni partono da

$F_0=1$, $F_1=1$

I numeri di Fibonacci

1, 1, 2, 3, 5, 8, 13, 21, …

I numeri di Fibonacci sono legati alla “sezione aurea” Φ.

$$\lim_{n\to\infty} \frac{F_n}{F_{n-1}} = \phi$$

$$\phi = \frac{1+\sqrt{5}}{2} = 1,6180339887...$$

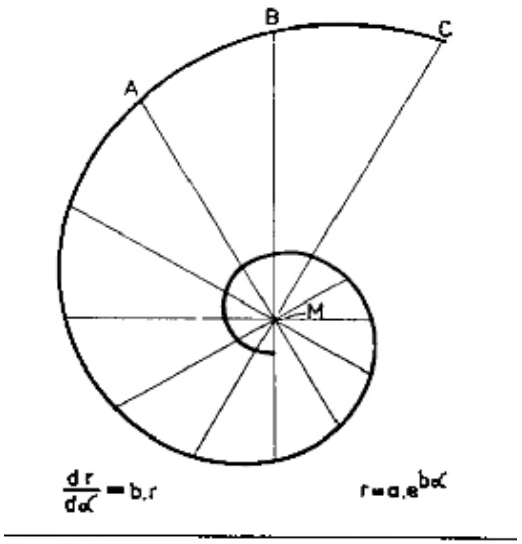
I numeri di Fibonacci

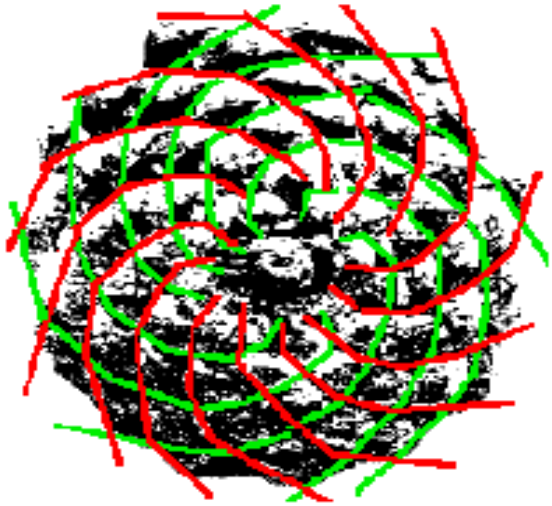
1, 1, 2, 3, 5, 8, 13, 21, …

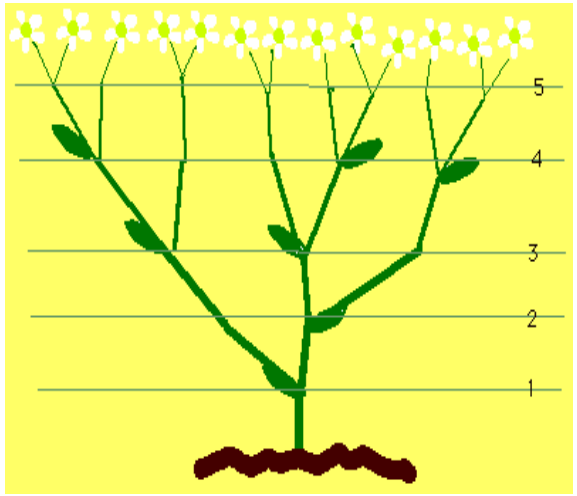
I numeri di Fibonacci godono di una gamma stupefacente di proprietà, si incontrano nei modelli matematici di svariati fenomeni e sono utilizzabili per molti procedimenti computazionali; essi inoltre posseggono varie generalizzazioni interessanti. A questi argomenti viene espressamente dedicato un periodico scientifico, *The Fibonacci Quarterly*. E non solo…..

Fermata Vanvitelli della Metropolitana di Napoli

Sezione aurea







La spirale logaritmica
Una conchiglia
Una pigna
Disposizione dei petali
Crescita di una pianta
Il Partenone
…….

Come calcolare l' n-esimo numero di Fibonacci?

Un primo approccio numerico.

Possiamo usare una funzione matematica che calcoli direttamente i numeri di Fibonacci.

Infatti si può dimostrare che:

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

$$\begin{aligned} \phi &= \frac{1+\sqrt{5}}{2} \approx +1.618 \\ \hat{\phi} &= \frac{1-\sqrt{5}}{2} \approx -0.618 \end{aligned}$$

e quindi…

Un primo algoritmo numerico

```
Fibonacci1(n)

    return 1/√5 (Φ^n - Φ^^n)
```

Però:
Problemi di accuratezza sul numero di cifre decimali;
il risultato non sarebbe preciso (i numeri di Fibonacci sono interi).

Un secondo algoritmo ricorsivo

Secondo approccio: utilizzare la definizione ricorsiva, che lavora solo su interi

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

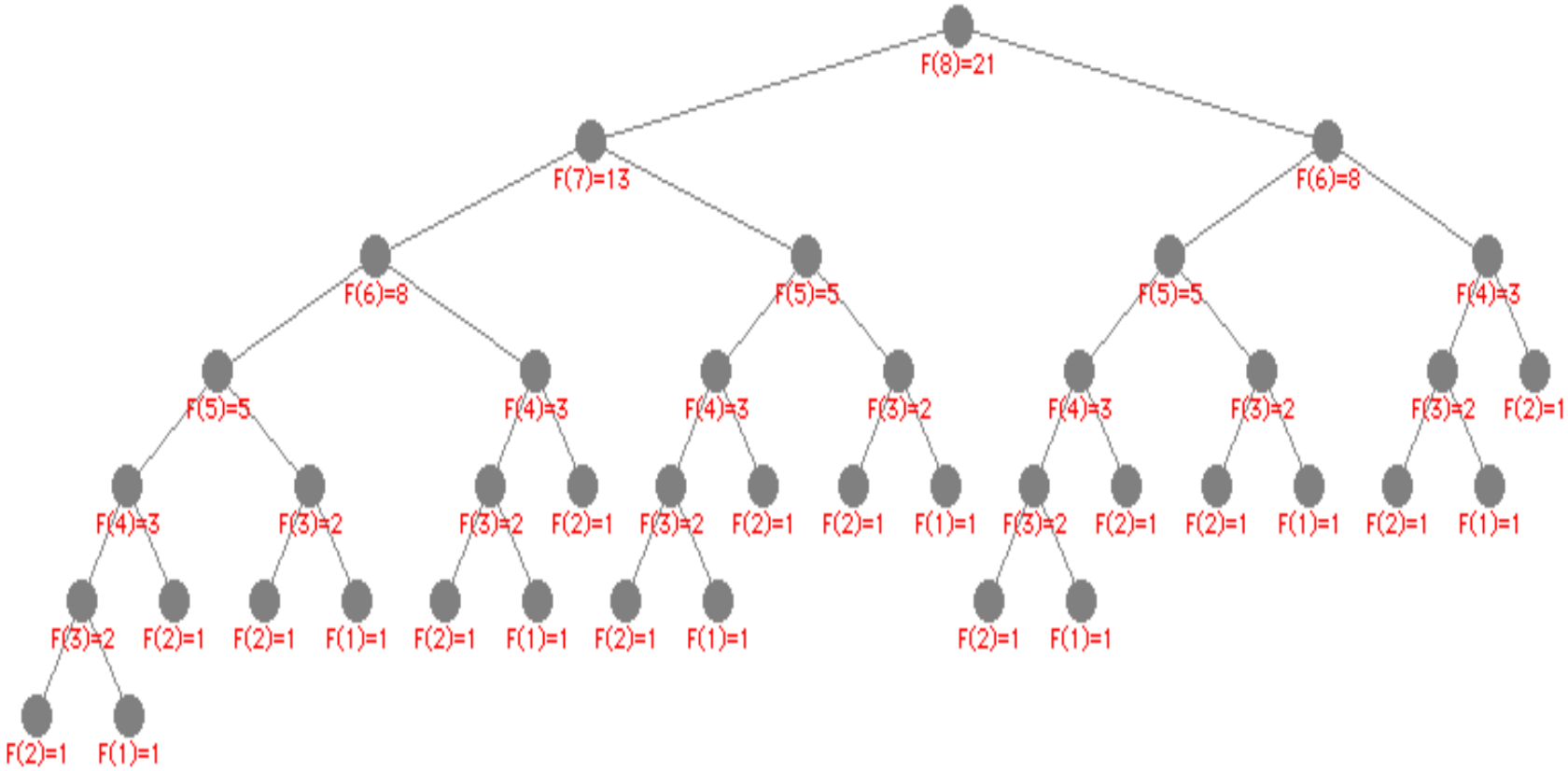
```
Fibonacci2(n)

If n≤2 then Return 1
Else Return Fibonacci2(n-1)+ Fibonacci2(n-2)
```

Un esempio

```
Fibonacci2(n)

If n≤2 then Return 1
Else Return Fibonacci2(n-1)+ Fibonacci2(n-2)
```



Analisi del tempo di esecuzione

$$T(n) = T(n-1) + T(n-2) + \Theta(1) \qquad n \geq 3$$
$$T(n) = \Theta(1) \qquad n = 1, 2$$

$$T(n) = O(2^n)$$

Dimostrazione per induzione.

Sia c>0 tale che $T(n) \leq T(n-1) + T(n-2) + c$ e $T(1), T(2) \leq c$.

Tesi: $T(n) \leq c\, 2^n$ per ogni $n \geq 1$

Base: $T(1) \leq c \leq c\, 2^1 = 2\, c$; $T(2) \leq c \leq c\, 2^2 = 4\, c$

Ipotesi induttiva: $T(n-1) \leq c\, 2^{n-1}$ e $T(n-2) \leq c\, 2^{n-2}$

Passo induttivo: $T(n) \leq c\, 2^{n-1} + c\, 2^{n-2} + c \leq c\, 2^{n-1} + c\, 2^{n-2} + c\, 2^{n-2} =$

$= c\, 2^{n-1} + 2 \cdot c\, 2^{n-2} = c\, 2^{n-1} + c\, 2^{n-1} = c\, 2^n$

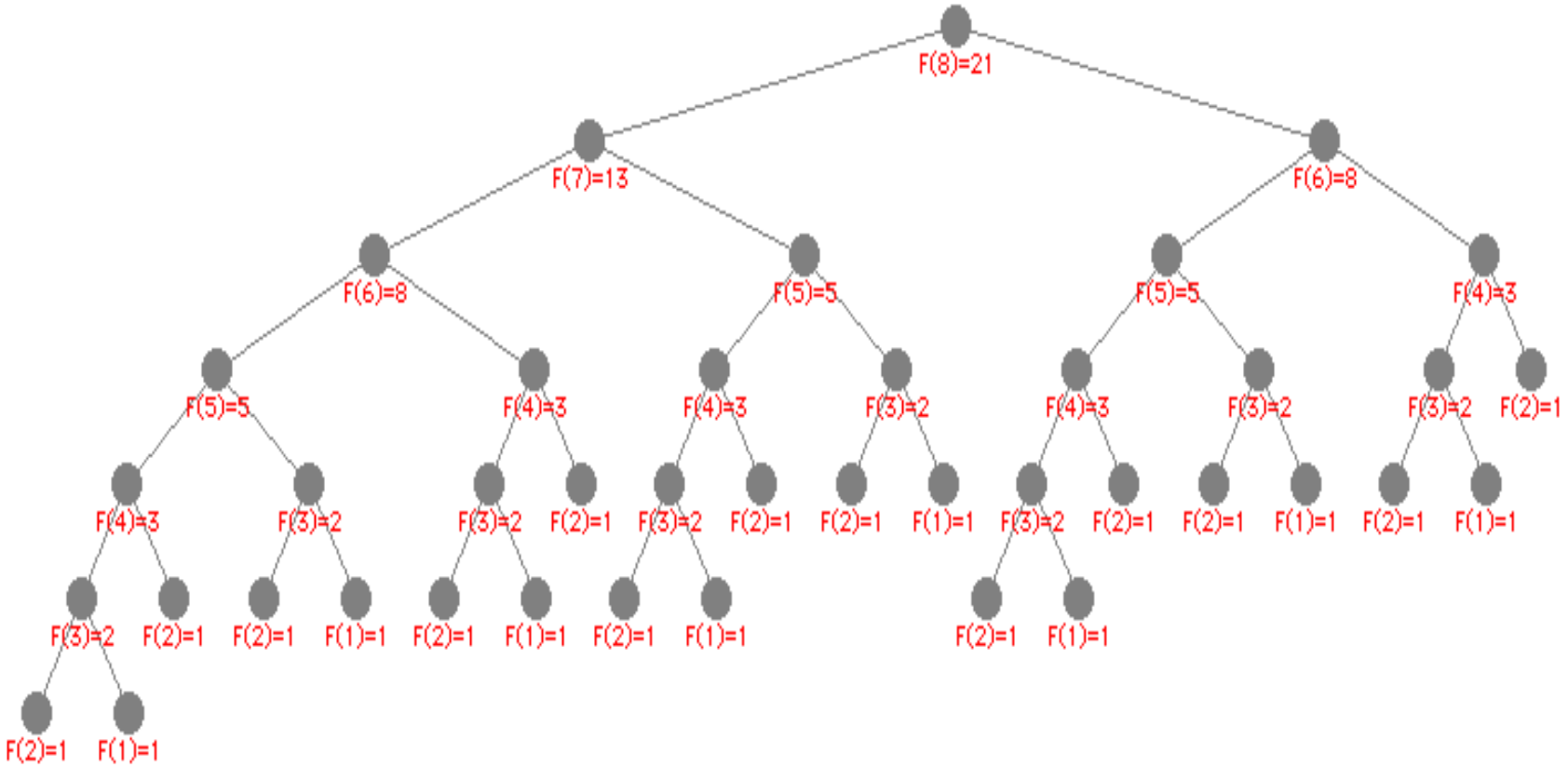
Nota: in realtà questa è solo una limitazione superiore, ma si può dimostrare che: T(n) è proporzionale ad $F(n) \approx \Phi^n$ (vedi esercizio alla fine)

Si può fare di meglio?

Difetti dell'algoritmo ricorsivo

Perché l'algoritmo **`Fibonacci2`** è lento?

Perché continua a ricalcolare ripetutamente la soluzione dello stesso sottoproblema.



Migliorare l'algoritmo ricorsivo (versione con annotazione)

Perché non memorizzare le soluzioni dei sottoproblemi via via calcolate (anziché perdere il lavoro già fatto)?

Basta un array F[1..n] per memorizzare tutti valori già calcolati.

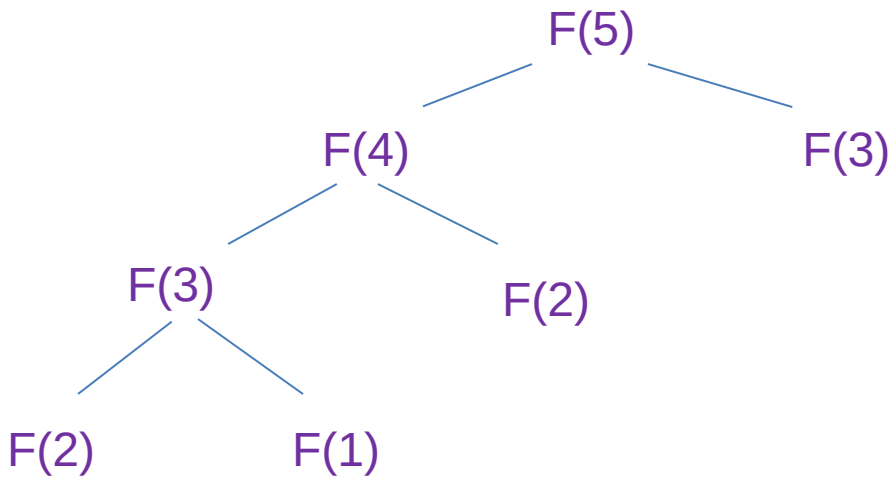
L'array F inizialmente è vuoto.

```
Fibonacci3-memo(j)

If j≤2 then F[j]=1
            Return F[j]
Else if F[j] non è vuoto then Return F[j]
Else Define F[j]= Fibonacci3-memo(j-1)+
                  Fibonacci3-memo(j-2)
      Return F[j]
Endif
```

Esempio j=5

```
Fibonacci3-memo(j)
If j≤2 then F[j]=1
              Return F[j]
Else if F[j] non è vuoto then Return F[j]
Else Define F[j]= Fibonacci3-memo(j-1)+
                              Fibonacci3-memo(j-2)
         Return F[j]
Endif
```

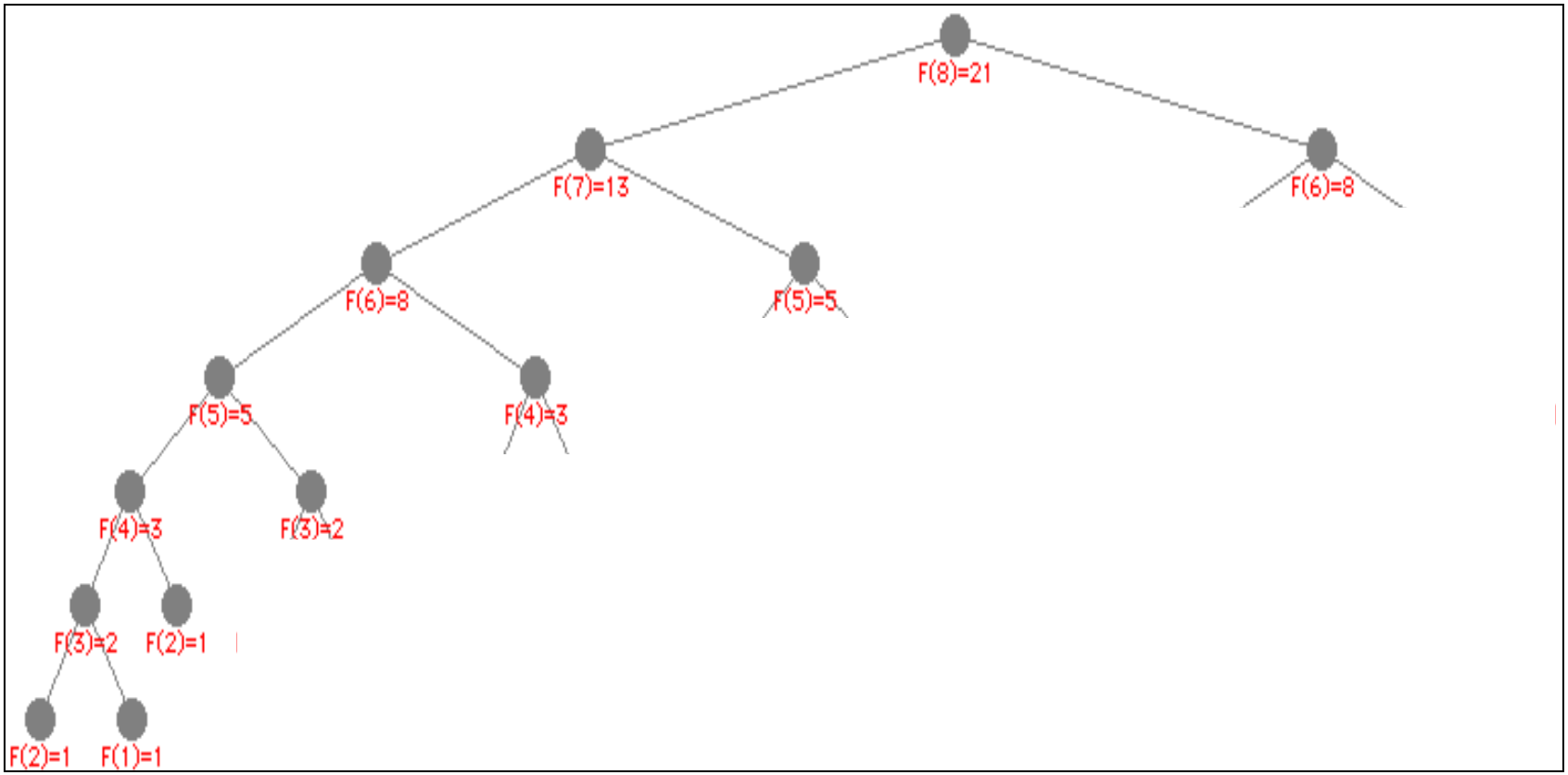


1	1	2	3	5
1	1	2	3	
1	1	2		
1	1			
	1			

Vantaggi dell'algoritmo di programmazione dinamica

Perché l'algoritmo **`Fibonacci2`** è lento?

Perché continua a ricalcolare ripetutamente la soluzione dello stesso sottoproblema. Invece **`Fibonacci3-memo`**

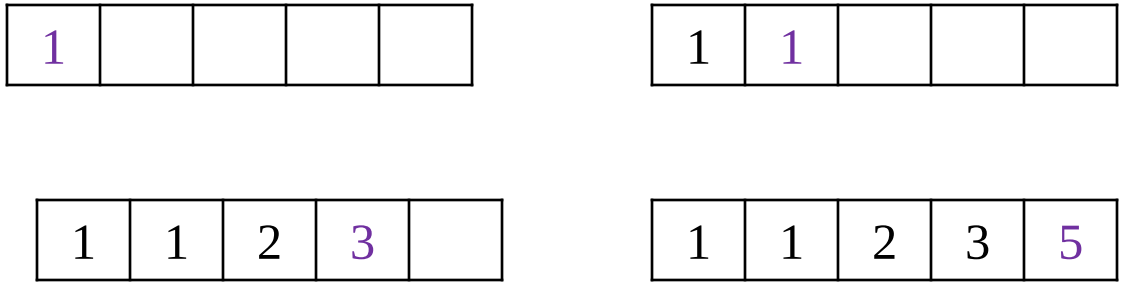


Migliorare l'algoritmo ricorsivo (versione iterativa)

```
Fibonacci3-iter(n)

F[1]=1
F[2]=1
For i=3,…,n
     F[i]= F[i-1]+F[i-2]
Endfor
Return F[n]
```

Esempio: n=5



Tempo di esecuzione

Il tempo di esecuzione di **`Fibonacci3-iter`** è $\Theta(n)$

Anche il tempo di esecuzione di **`Fibonacci3-memo`** è $\Theta(n)$: ogni differente chiamata ricorsiva è eseguita solo una volta, richiede tempo costante, e ci sono O(n) diverse chiamate a **`Fibonacci3-memo`**.

Confronto prestazioni (2004…)

L'algoritmo `Fibonacci3-iter` e `Fibonacci3-memo` impiegano tempo proporzionale a n invece di esponenziale in n come `Fibonacci2.`

Tempo effettivo richiesto da implementazioni in C dei due seguenti algoritmi su piattaforme diverse:

	`fibonacci2`(58)	`fibonacci3`(58)
Pentium IV 1700MHz	15820 sec. ($\simeq$ 4 ore)	0.7 milionesimi di secondo
Pentium III 450MHz	43518 sec. ($\simeq$ 12 ore)	2.4 milionesimi di secondo
PowerPC G4 500MHz	58321 sec. ($\simeq$ 16 ore)	2.8 milionesimi di secondo

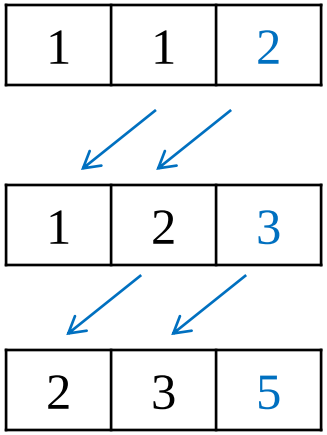
Analisi dello spazio necessario

Lo spazio di memoria necessario per eseguire **`Fibonacci3-iter`** e **`Fibonacci3-memo`** è

$$S(n) = \Theta(n)$$

perché entrambi gli algoritmi usano un array con n celle.

Ma in realtà possono bastare 3 celle (per qualsiasi n):
per il calcolo iterativo di F[j] servono solo i valori nelle 2 celle precedenti



Programmazione dinamica: caratteristiche

`Fibonacci3-memo` e **`Fibonacci3-iter`** sono algoritmi di programmazione dinamica: perché?

1. La soluzione al problema originale si può ottenere da soluzioni a sottoproblemi
2. Esiste una relazione di ricorrenza per la funzione che dà il valore ottimale ad un sottoproblema
3. Le soluzioni ai sottoproblemi sono calcolate una sola volta e via via memorizzate in una tabella

Due implementazioni possibili:

- Con annotazione (*memoized*) o *top-down*
- Iterativa o *bottom-up*

Programmazione dinamica vs Divide et Impera

Entrambe le tecniche dividono il problema in sottoproblemi: dalle soluzioni dei sottoproblemi è possibile risalire alla soluzione del problema di partenza

Dobbiamo allora considerare la tecnica Divide et Impera superata?

NO: La programmazione dinamica risulta più efficiente quando:

- Ci sono dei sottoproblemi ripetuti
- Ci sono solo un numero polinomiale di sottoproblemi (da potere memorizzare in una tabella)

Per esempio: nel MergeSort non ci sono sottoproblemi ripetuti.

Riepilogo, riferimenti, esercizi

- Un primo semplice esempio di programmazione dinamica: il calcolo dei numeri di Fibonacci che contiene gli ingredienti essenziali della tecnica. Puoi consultare [DFI] e/o le slides.

- Esercizi che potete svolgere: nelle prossime slides

Esercizio (analisi Fibonacci2)

Nelle slides precedenti è dimostrato che il tempo di esecuzione T(n) dell'algoritmo ricorsivo Fibonacci2(n) è $T(n)=O(2^n)$.

Questa in realtà è soltanto una limitazione superiore. Dimostrare che:

1. $T(n)= \Omega(l(n))$, dove l(n) è il numero di foglie dell'albero della ricorsione per Fibonacci2(n) (mostrato nelle slides precedenti)
2. Per ogni $n \geq 1$, $l(n)=F(n)$ (il numero di foglie è esattamente uguale all'n-esimo numero di Fibonacci), usando l'induzione strutturale.

Esercizio (Fibonacci con 2 celle)

Fornire una variante dell'algoritmo **`Fibonacci3-iter(n)`**, mostrato nelle slide precedenti, che utilizzi soltanto 2 celle di memoria (anziché n).

Appello 9 luglio 2015

(svolto in parte – da completare sulla piattaforma)

Quesito 2 (24 punti)

Da quando ti sei registrato su Facebook ad oggi, i tuoi amici sono aumentati in maniera vertiginosa. Il primo anno avevi solo 10 amici; il secondo 35; il terzo 100 e nessuno ti elimina mai dagli amici. Hai poi notato che ogni tuo amico, 3 anni dopo averti dato la sua amicizia, ti porta un nuovo amico (spesso è un collega di università/lavoro, fidanzato/a, fratello/a, cugino/a). E tutti i tuoi nuovi amici si aggiungono sempre e solo in questo modo.
Sapresti calcolare quanti diventeranno i tuoi amici nei prossimi anni?

a) Descrivere un algoritmo efficiente per il calcolo del numero dei tuoi amici dopo *n* anni dalla tua registrazione su Facebook, supponendo che aumentino sempre rispettando la regola sopra descritta. E'necessario analizzare la complessità di tempo e di spazio dell'algoritmo proposto.
b) Valutare la crescita del numero di amici rispetto ad *n* (in notazione asintotica).

Appello 9 febbraio 2011

I numeri di Tribonacci sono cosi' definiti:

$R(0) = 0$

$R(1) = 0$

$R(2) = 1$

$R(n) = R(n-1) + R(n-2) + R(n-3)$ se $n \geq 3$.

a) Scrivere lo pseudocodice di un algoritmo di programmazione dinamica per il calcolo dell'n-esimo numero di Tribonacci $R(n)$.

b) Analizzare la complessita' di tempo e di spazio dell'algoritmo proposto.

c) E' possibile realizzare l'algoritmo con spazio $O(1)$? Giustificare la risposta.

Appello 12 settembre 2016

Quesito 1 (24 punti) (*Cappanacci*)

La sequenza dei numeri di Fibonacci k-generalizzati, per un intero k, è definita come segue

$F_{n,k} = 0$ per $n = 0, 1, \ldots, k-2$

$F_{k-1,k} = 1$

$F_{n,k} = F_{n-1,k} + F_{n-2,k} + \ldots + F_{n-k,k}$ per ogni $n \geq k$.

Descrivere ed analizzare un algoritmo di programmazione dinamica che dati due interi, n e k, calcola il numero $F_{n,k}$.

Esempio. Per k=3, i primi numeri di Fibonacci 3-generalizzati sono:

$F_{0,3} = F_{1,3} = 0$, $F_{2,3} = 1$, $F_{3,3} = 1$, $F_{4,3} = 2$, $F_{5,3} = 4$, …

Partition (svolto)

Una chiamata alla procedura `PARTITION` (come studiata) su [6, 1, 3, 9, 8] restituisce

A. 2 B. 3 C. 6

D. Nessuna delle risposte precedenti

Merge

Quanti confronti effettua l'algoritmo `MERGE` per la fusione dei due array ordinati [1,5,6,7] e [2,3,4]?

A. 4 B . 6 C. 12

D. Nessuna delle risposte precedenti

```
i = 1, j = 1
while (both lists are nonempty) {
   if (a_i ≤ b_j) append a_i to output list and increment i
   else(a_i > b_j)append b_j to output list and increment j
}
append remainder of nonempty list to output list
```

Tempo di esecuzione 4

Il tempo di esecuzione del seguente frammento di pseudocodice è

```
for i=1 to n/2
   PARTITION(A,i,n)
```

A. $\Theta(\log n)$
B. $\Theta(n \log n)$
C. $\Theta(n)$

D. Nessuna delle risposte precedenti

Dalla piattaforma

- (Relazioni di ricorrenza 2)

Nella risoluzione della relazione di ricorrenza

T(n) = 2 T(n /2) + n, con T(1) = c

col metodo di iterazione, qual è il valore di T(n) alla i-esima iterazione?

- (Soluzione relazione di ricorrenza 3)

La soluzione della relazione di ricorrenza

T(n) = 2T(n/2) + c, con T(1) = c, è: …

- (Soluzione relazione di ricorrenza 4)

La soluzione della relazione di ricorrenza

T(n) = 4T(n/2) + n, con T(1) = 1, è: …

Altri esempi (vedi piattaforma e-learning)

Sia $T(1) = 1$. Valutate

- $T(n) = 2T(n/2) + n^3$
- $T(n) = T(9n/10) + n$
- $T(n) = 16T(n/4) + n^2$
- $T(n) = 7T(n/3) + n^2$
- $T(n) = 7T(n/2) + n^2$
- $T(n) = 2T(n/3) + \sqrt{n}$
- $T(n) = T(n-1) + n$
- $T(n) = T(\sqrt{n}) + 1$