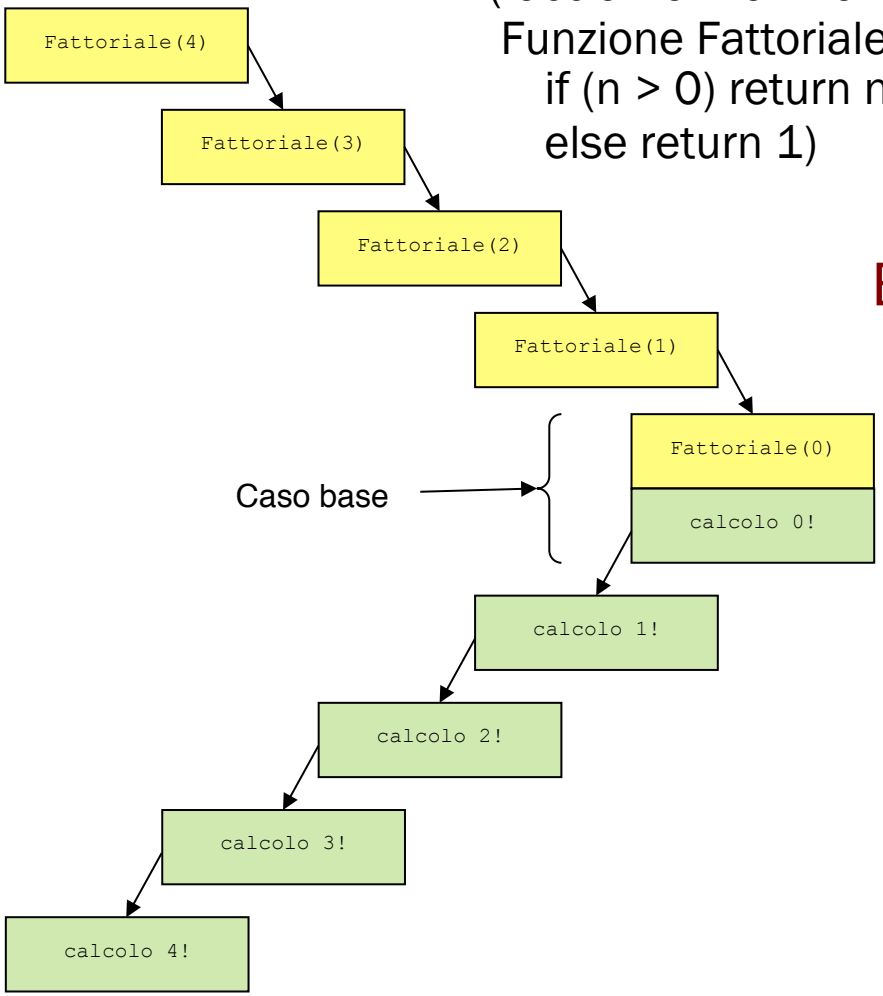
Corso di Introduzione agli algoritmi
Prof.ssa Tiziana Calamoneri

La ricorsione

La ricorsione (1)

Come si sviluppa il procedimento di calcolo effettivo:

(facciamo riferimento a:
Funzione Fattoriale_Ric (n: intero)
if (n > 0) return n*Fattoriale_Ric (n – 1)
else return 1)



Esempio: Fattoriale_Ric(4)

La ricorsione (2)

Ogni funzione, sia essa ricorsiva o no, richiede per la sua esecuzione una certa quantità di memoria RAM, per:

- caricare in memoria il codice della funzione;
- passare i parametri alla funzione;
- memorizzare i valori delle variabili locali della funzione.

Quindi le funzioni ricorsive in generale hanno maggiori esigenze, in termini di memoria, delle funzioni non ricorsive (dette anche funzioni iterative).

Per inciso, questa è la ragione per la quale una funzione ricorsiva mal progettata, ad es. nella quale la condizione di terminazione non si incontra mai, esaurisce con estrema rapidità la memoria disponibile con la conseguente inevitabile terminazione forzata dell'esecuzione.

La ricorsione (3)

Va detto che qualsiasi problema risolvibile con un algoritmo ricorsivo può essere risolto anche con un algoritmo iterativo.

Ma allora, in quali situazioni è consigliabile utilizzare un algoritmo ricorsivo anziché uno iterativo? Queste considerazioni possono essere d'aiuto nella decisione:

- è conveniente utilizzare la ricorsione quando essa permette di formulare la soluzione del problema in un modo che è chiaro ed aderente alla natura del problema stesso, mentre la soluzione iterativa è molto più complicata o addirittura non evidente;
- non è conveniente utilizzare la ricorsione quando esiste una soluzione iterativa altrettanto semplice e chiara;
- non è conveniente utilizzare la ricorsione quando l'efficienza è un requisito primario.

La ricorsione (4)

Esempio: calcolo dell'*n*-esimo numero di Fibonacci

La definizione dei numeri di Fibonacci è la seguente:

- *F(0) = 0*
- *F(1) = 1*
- *F(i) = F(i – 1) + F(i – 2) se i > 1*

Leonardo Fibonacci
1170-1250

La ricorsione (5)

Esempio: calcolo dell'*n*-esimo numero di Fibonacci (segue)

Buona notizia:

Esiste una formula chiusa per il calcolo del valore di *F(n):*

$$F(n) = \frac{1}{\sqrt{5}}(\Phi^n - \hat{\Phi}^n)$$

dove $\Phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ $\hat{\Phi} = \frac{1-\sqrt{5}}{2} \approx -0.618$

Cattiva notizia:

La valutazione di tale formula introduce errori numerici dovuti ai calcoli in virgola mobile.

La ricorsione (6)

segue Esempio: calcolo dell'*n*-esimo numero di Fibonacci

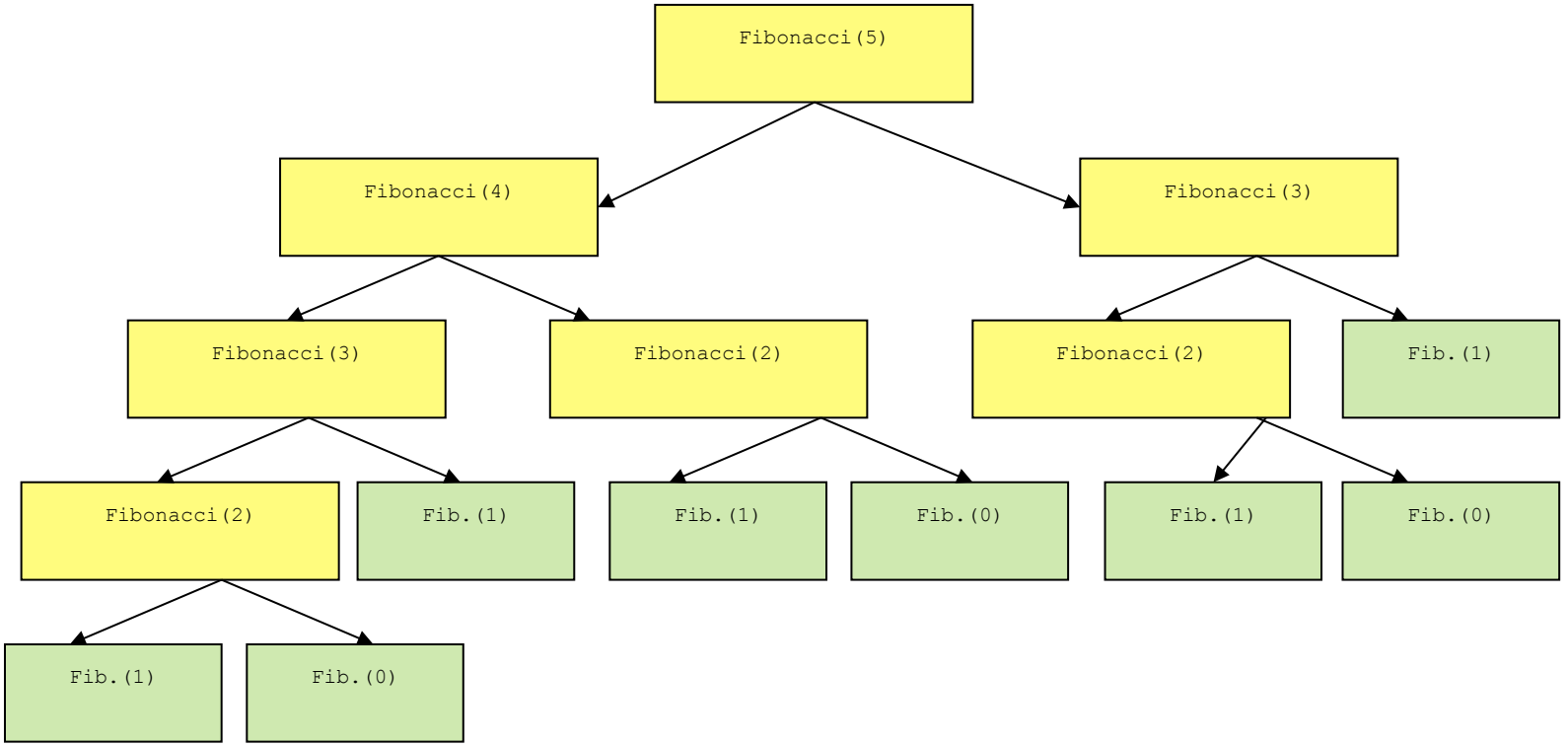
Allora calcoliamo seguendo la definizione:

```
Funzione Fibonacci (n: intero non negativo)
    if ((n = 0) oppure (n = 1))
            return n
    else
            return (Fibonacci(n – 1) + (Fibonacci(n – 2))
```

La ricorsione (7)

segue Esempio:
Fibonacci(5)

```
Funzione Fibonacci (n: intero non negativo)
    if ((n = 0) oppure (n = 1))
            return n
    else
            return (Fibonacci(n – 1) + (Fibonacci(n – 2))
```

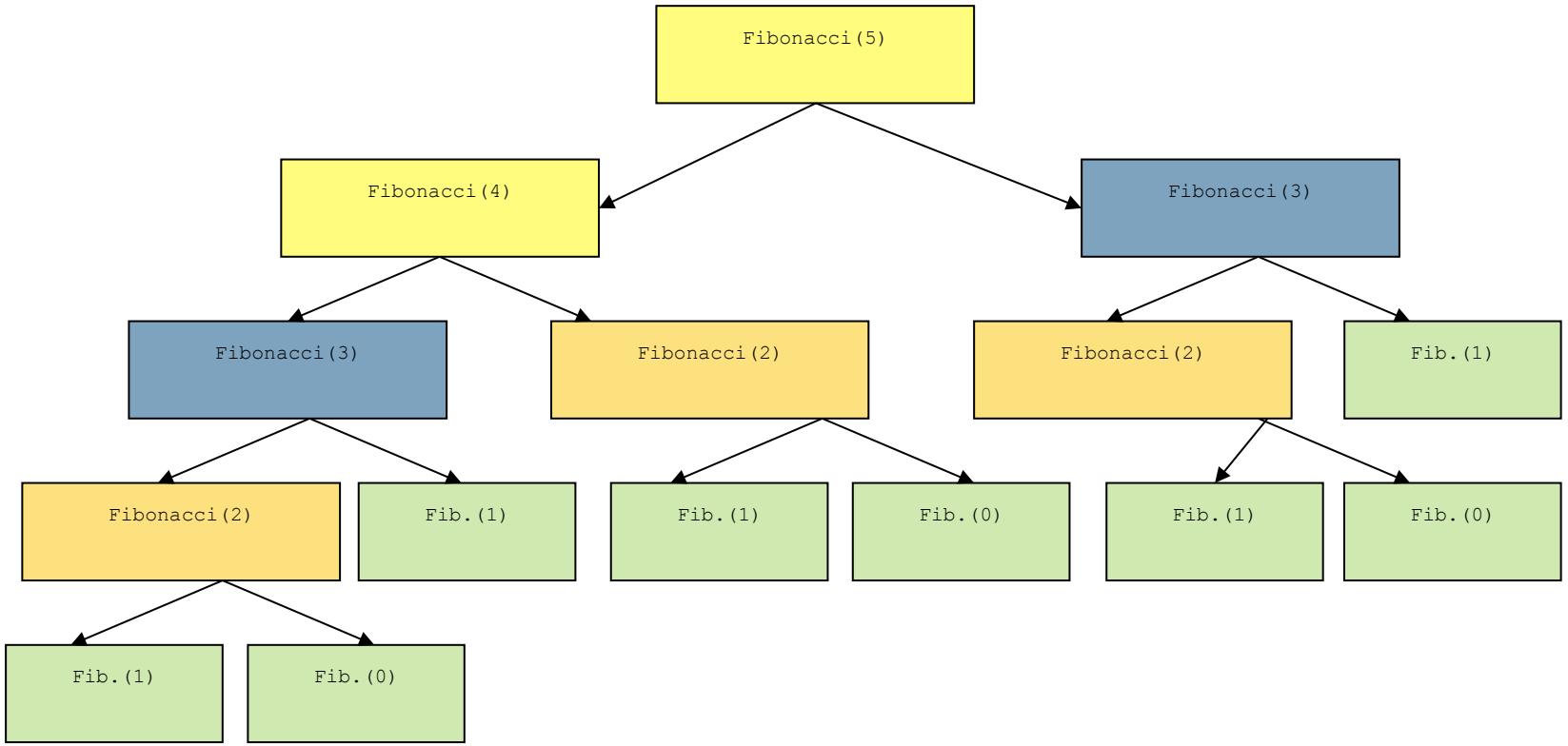


La ricorsione (8)

segue Esempio: calcolo dell'*n*-esimo numero di Fibonacci

Due osservazioni:

1. uno stesso calcolo viene ripetuto molte volte.



La ricorsione (9)

segue Esempio: calcolo dell'*n*-esimo numero di Fibonacci

Due osservazioni (segue):

2. L'occupazione dello spazio di memoria, legata alle chiamate di funzione, è molto alta.

Il numero totale delle chiamate effettuate dall'inizio alla fine dell'elaborazione cresce molto velocemente con *n*.

Inoltre, quando si arriva a ciascun caso base vi è una catena di chiamate "aperte" lungo il percorso che va dalla chiamata iniziale al caso base in questione.

La ricorsione (10)

segue Esempio: calcolo dell'*n*-esimo numero di Fibonacci

Osserviamo che il dispendio di tempo richiesto dalla funzione ricorsiva per il calcolo del numero di Fibonacci può essere evitato usando una versione iterativa:

```
Funzione Fibonacci_iterativa_1 (n: intero non negativo)
    Sia Fib[1..n] un vettore di n interi                    Θ(1)
    Fib[1] ←1                                               Θ(1)
    Fib[2] ←1                                               Θ(1)+(n-2) volte
    for i=3 to n do                                           Θ(1)
          Fib[i] ←Fib[i-1]+Fib[i-2]                         Θ(1)
    return Fib[n]
```

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

spazio occupato: $\Theta(n)$

segue Esempio: calcolo dell'*n*-esimo numero di Fibonacci

Possiamo ottimizzare l'utilizzo di memoria osservando che il calcolo di *F(n)* dipende solo dai due valori precedenti:

```
Funzione Fibonacci_iterativa (n: intero non negativo)
    if ((n = 0)oppure (n = 1))
            return n
    else
            fib_prec_prec ← 0
            fib_prec ← 1
            for (i = 2 to n)
                    fib ← fib_prec + fib_prec_prec
                    fib_prec_prec ← fib_prec
                    fib_prec ← fib;
            return fib
```

La ricorsione (12)

segue Esempio

```
Funzione Fibonacci_iterativa (n: intero non negativo)
    if ((n = 0) oppure (n = 1)) return n
    else
        fib_prec_prec ← 0
        fib_prec ← 1
        for (i = 2 to n)
            fib ← fib_prec + fib_prec_prec
            fib_prec_prec ← fib_prec
            fib_prec ← fib;
        return fib
```

Infatti, questa funzione ha un costo computazionale di $\Theta(n)$, e banalmente, utilizza spazio costante.

La ricorsione (13)

segue Esempio: calcolo dell'*n*-esimo numero di Fibonacci

Siamo in grado di calcolare il costo computazionale della funzione ricorsiva data?

```
Funzione Fibonacci (n: intero non negativo)
    if ((n = 0) oppure (n = 1))
            return n
    else
            return (Fibonacci(n – 1) + (Fibonacci(n – 2))
```

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

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

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

E' simile alla formula che definisce i numeri di Fibonacci

esponenziale, ma come lo dimostriamo?

Esercizi (1)

- Progettare una funzione ricorsiva che, dati due interi *k* ed *n*, *0≤k≤n*, calcoli il valore del coefficiente binomiale *n* su *k* utilizzando la seguente relazione:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \quad \text{con} \quad \binom{n}{0} = \binom{n}{n} = 1$$

- Progettare una funzione ricorsiva che, dato un intero *n*, risolva il problema delle Torri di Hanoi:
  - sono dati 3 pali p_1, p_2 e p_3, ed *n* dischi di raggio via via decrescente che, all'inizio sono infilati in quest'ordine in p_1;
  - ogni disco può essere spostato da un palo ad un altro senza che un disco di diametro maggiore sia posto sopra un disco di diametro minore
  - l'obiettivo consiste nello spostare tutti gli *n* dischi dal primo al terzo palo, facendoli eventualmente passare per il secondo.

Esercizi (2)

- Progettare una funzione ricorsiva che, dato un vettore *A* di *n* interi, lo stampi al contrario (cioè: A[n] A[n-1]…A[2]A[1])
- Progettare una funzione ricorsiva che calcoli la somma di *n* valori inseriti da tastiera (un valore inserito per ogni chiamata ricorsiva)
- Progettare un algoritmo ricorsivo che, dati due interi *x* e *y*, x>y>0, ne calcoli il massimo comun divisore utilizzando il seguente procedimento (di Euclide):
  - se *y=0* allora *MCD(x,y)=x*
  - altrimenti *MCD(x,y)=MCD(y,x%y)*

  dove *x%y* rappresenta il resto della divisione tra *x* ed *y*

Esercizi (3)

Progettare due funzioni, una ricorsiva ed una iterativa, per risolvere i seguenti problemi:

- dati in input due interi *n* e *k,* calcolare la potenza *k*-esima di *n*
- dati *n* interi memorizzati in un vettore, calcolarne la somma
- dato un vettore di *n* caratteri, verificare se esso è palindromo
- dato un vettore di *n* elementi, trovare l'elemento minimo.