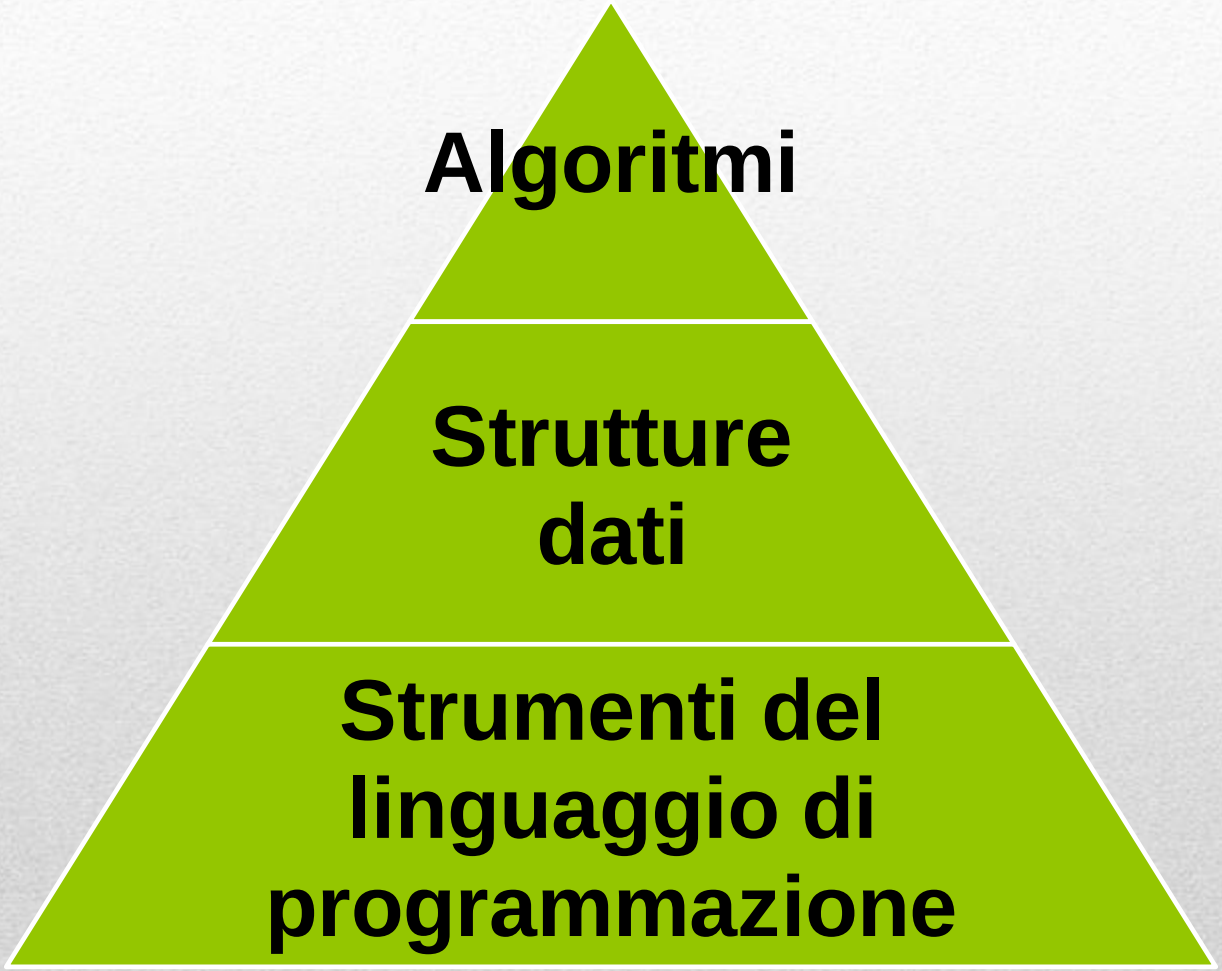
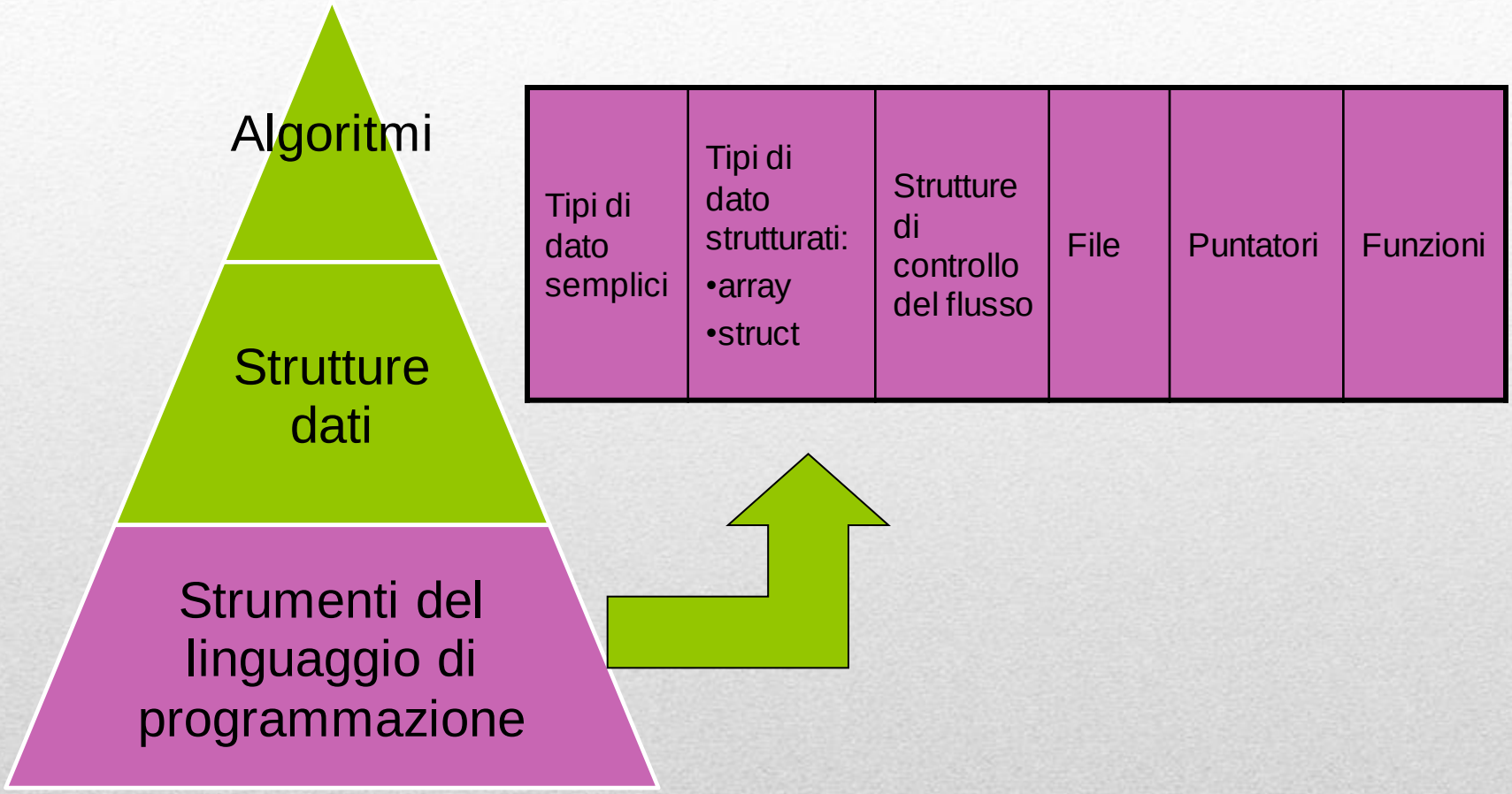
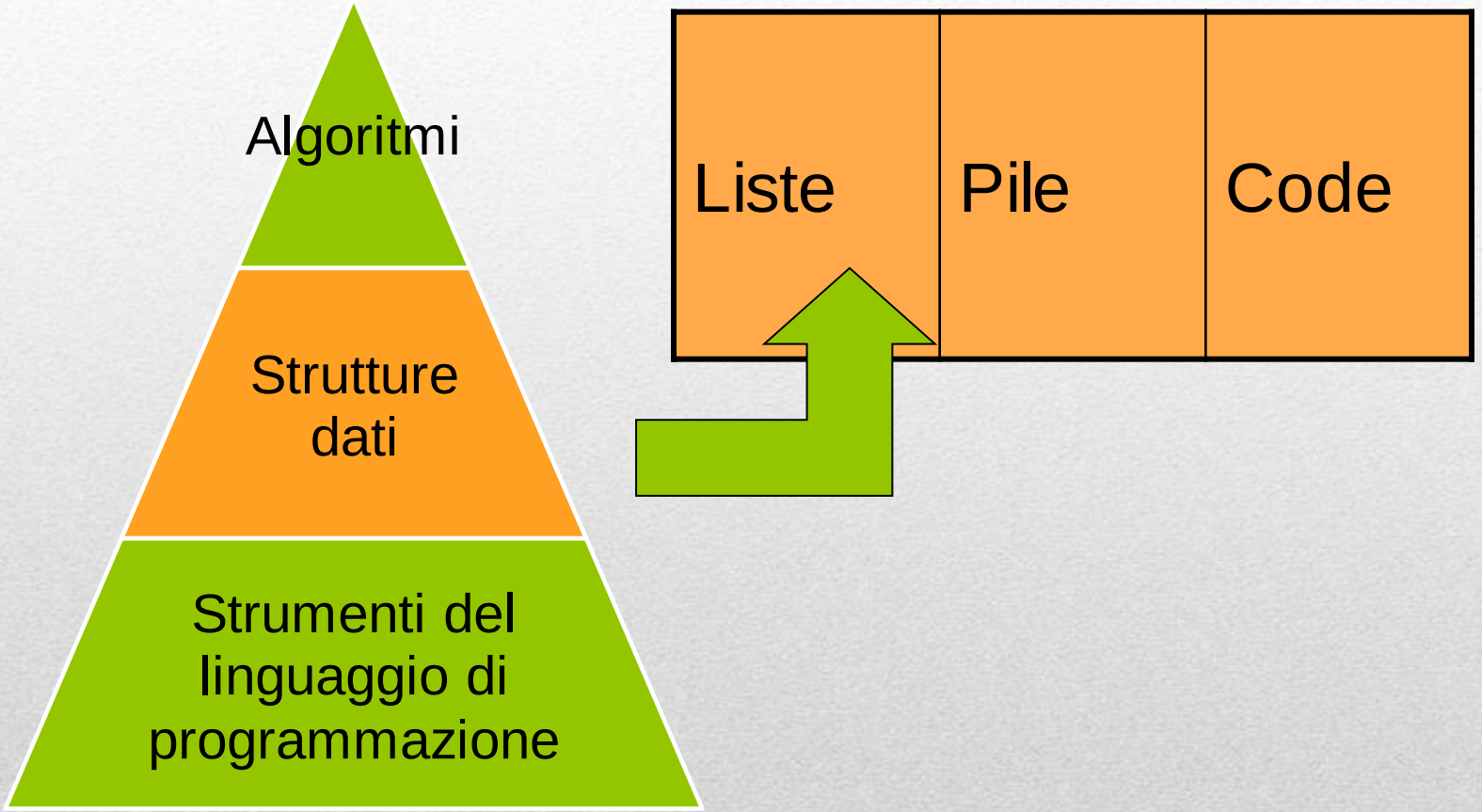
COSA IMPAREREMO

Argomenti corso



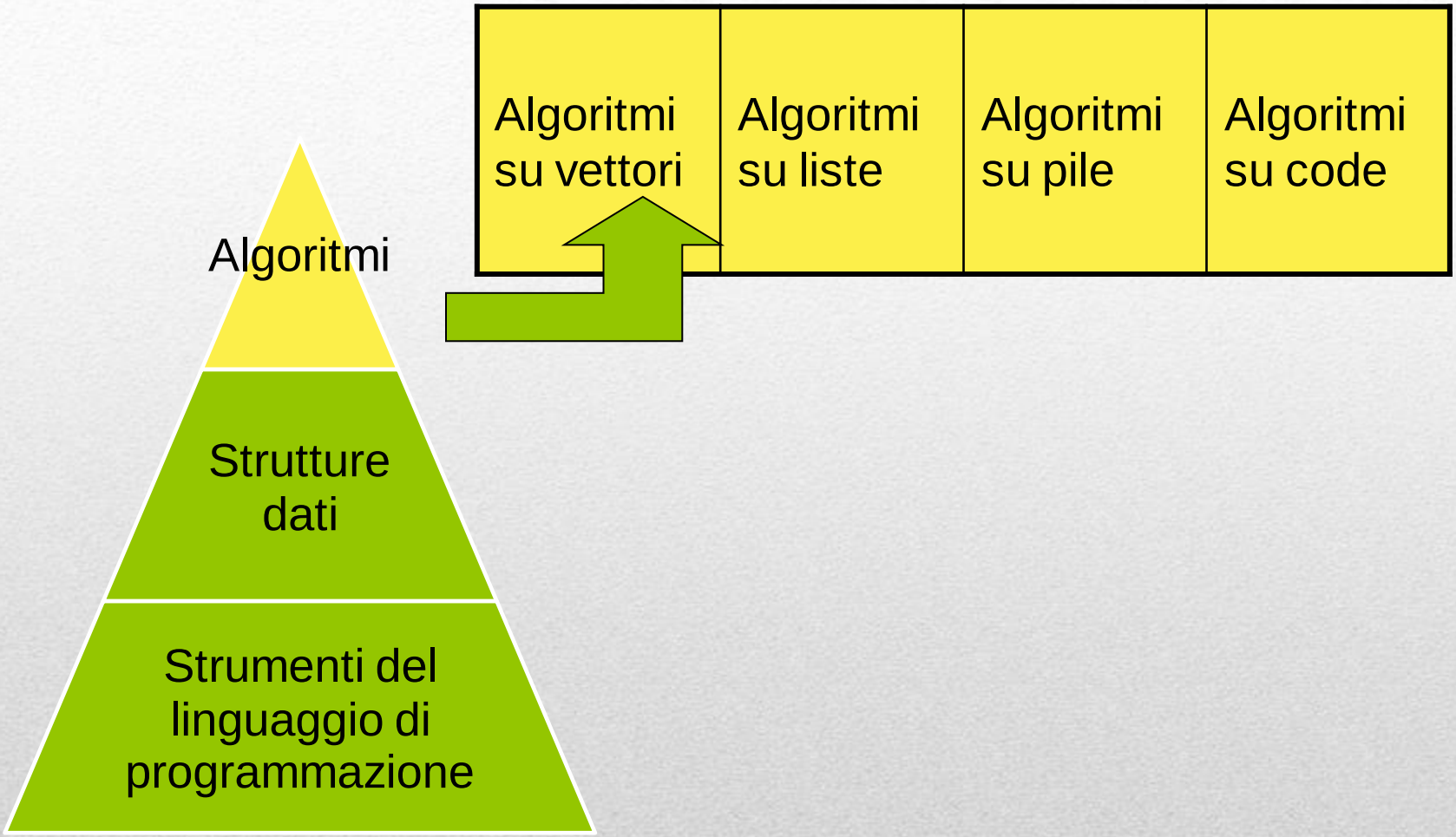
Argomenti corso



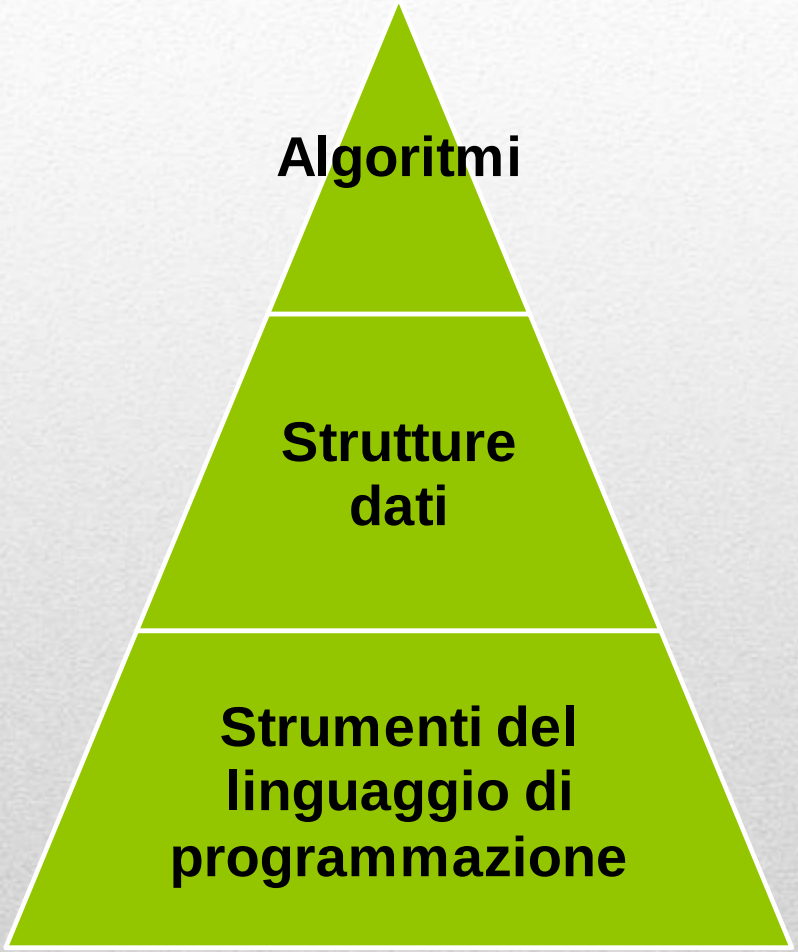


Argomenti corso

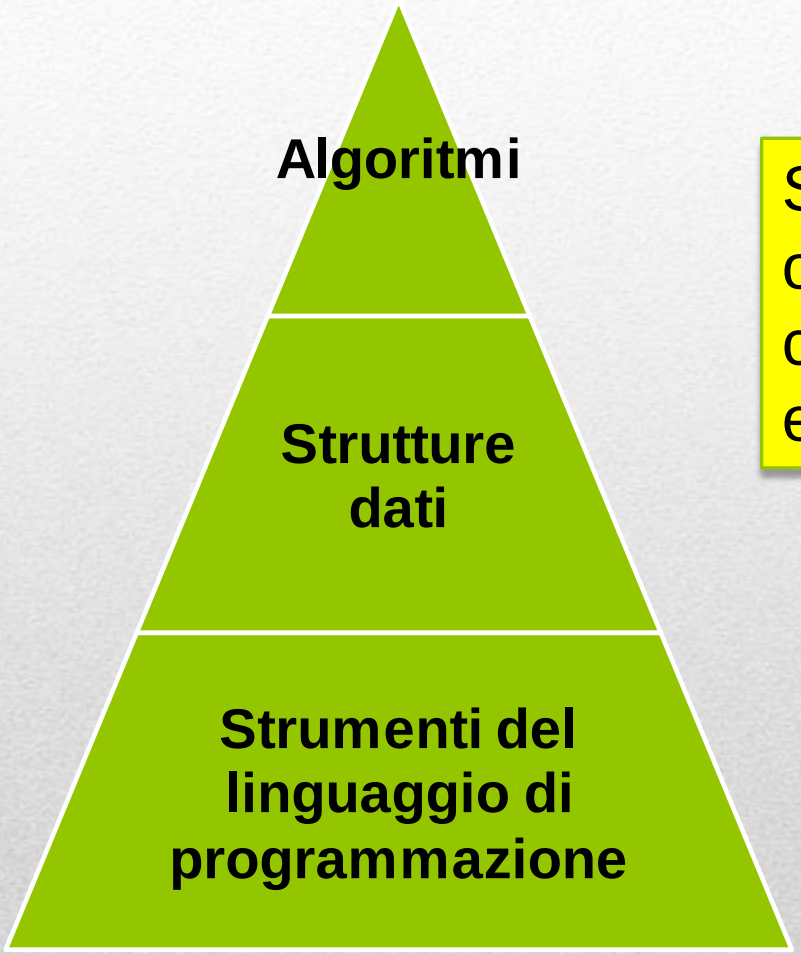
Argomenti corso



Argomenti corso – cosa succede se ...?



Argomenti corso – cosa impariamo?

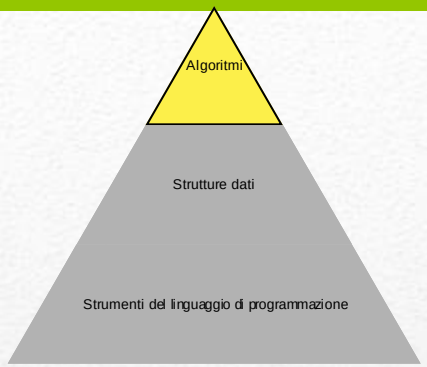


Scelte CONSAPEVOLI operate considerando tutti i casi possibili e valutando efficacia ed efficienza

PROGRAMMARE == IMPLEMENTARE?

«Algoritmi e strutture dati»
Camil Demetrescu, Irene Finocchi, Giuseppe F. Italiano, McGraw Hill
Capitolo 1

L'isola dei conigli



Leonardo da Pisa si interessò di molte cose, tra cui il seguente problema di dinamica delle popolazioni:

Quanto velocemente si espanderebbe una popolazione di conigli sotto appropriate condizioni?

In particolare, partendo da una coppia di conigli in un'isola deserta, quante coppie si avrebbero nell'anno n?

Le regole di riproduzione

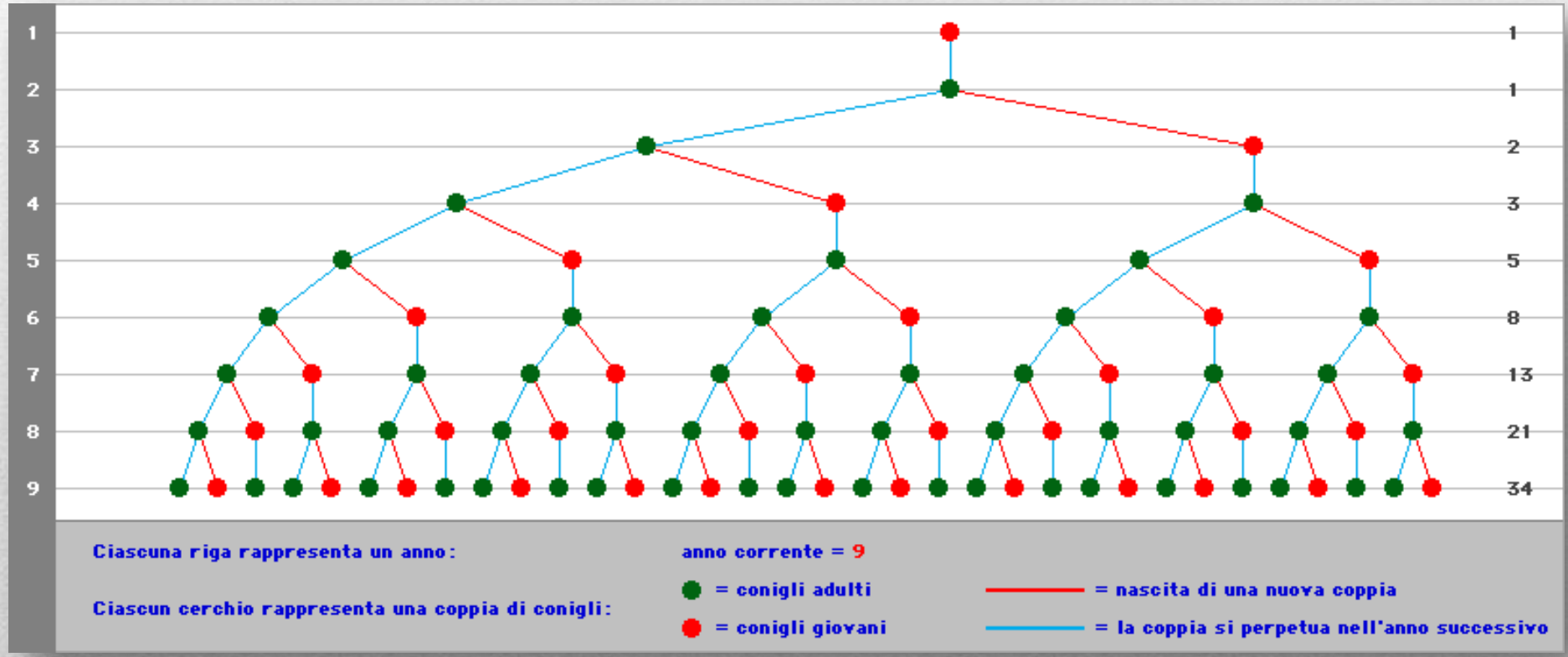
Una coppia di conigli genera due coniglietti ogni anno

I conigli cominciano a riprodursi soltanto al secondo anno dopo la loro nascita

I conigli sono immortali

L'albero dei conigli

La riproduzione dei conigli può essere descritta in un albero come segue:



La regola di espansione

Nell'anno n, ci sono tutte le coppie dell'anno precedente, e una nuova coppia di conigli per ogni coppia presente due anni prima.

Indicando con F_n il numero di coppie dell'anno n, abbiamo la seguente relazione di ricorrenza:

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

Il problema

Come calcoliamo F_n?

Un approccio numerico

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

Si può dimostrare che:

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

dove:

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

Algoritmo fibonacci1

algoritmo `fibonacci1`$(intero\ n) \rightarrow intero$

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

Correttezza!

Qual è l'accuratezza su ϕ $\hat{\phi}$ ottenere un risultato corretto?

Ad esempio, con 3 cifre decimali:

$$\phi \approx 1.618 \text{ e } \hat{\phi} \approx -0.618$$

n	`fibonacci1`(n)	arrotondamento	F_n
3	1.99992	2	2
16	986.698	987	987
18	2583.1	2583	2584

Algoritmo fibonacci2

Poiché fibonacci1 non è corretto, un approccio alternativo consiste nell'utilizzare direttamente la definizione ricorsiva:

algoritmo `fibonacci2`*(intero n)* $\rightarrow$ *intero*
  if (n$\leq$2) **then return** 1
  else return `fibonacci2`(n-1) +
    `fibonacci2`(n-2)

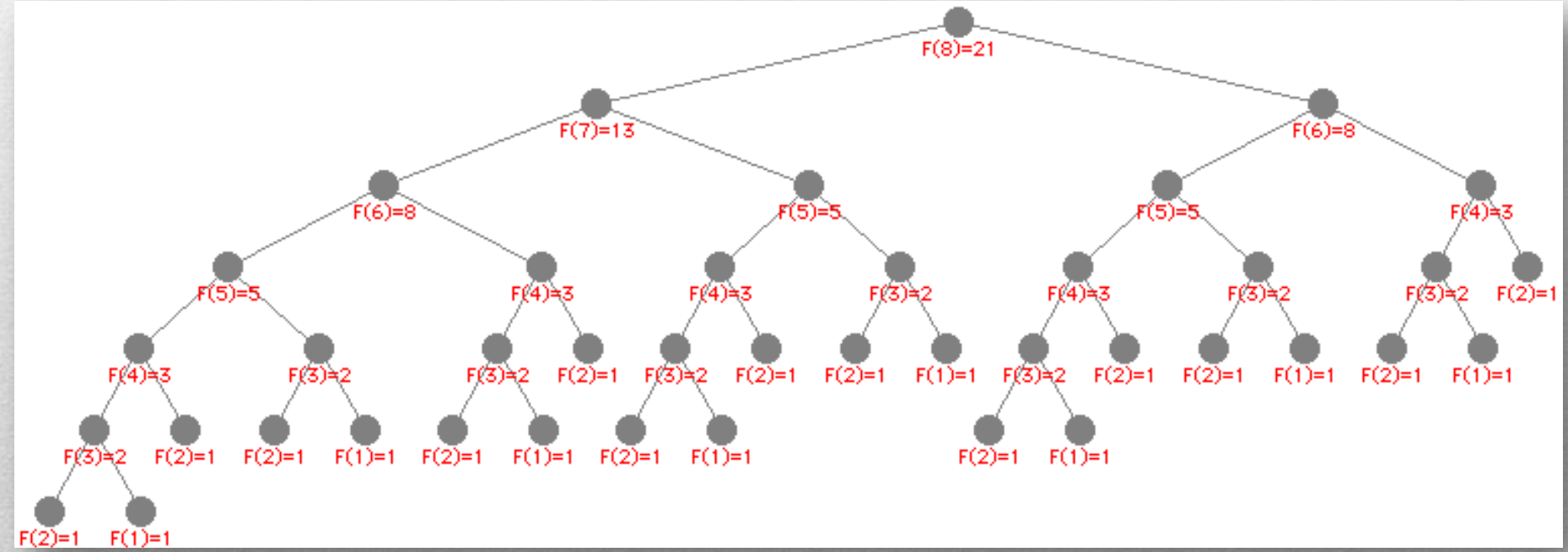
Opera solo con numeri interi.

Albero della ricorsione

Utile per risolvere la relazione di ricorrenza.

Nodi corrispondenti alle chiamate ricorsive.

Figli di un nodo corrispondenti alle sottochiamate.



Albero della ricorsione

Dall'albero della ricorsione e sfruttando alcuni teoremi sul numero di foglie e sul numero di nodi interni contenuti in un albero binario, si evince che l'algoritmo *fibonacci2* è **MOLTO** lento: il numero di linee di codice mandate in esecuzione a fronte di una generica chiamata alla funzione *fibonacci2(n)* cresce infatti ….

!!!! come i conigli di Fibonacci !!!!

Ad esempio:

- per n=8 vengono mandate in esecuzione 61 linee di codice,
- per n=45 vengono mandate in esecuzione 3.404.709.508 linee di codice.

Algoritmo fibonacci3

Perché l'algoritmo *fibonacci2* è lento? Perché continua a ricalcolare ripetutamente la soluzione dello stesso sottoproblema.

: memorizzare allora in un array le soluzioni dei sottoproblemi

algoritmo `fibonacci3`*(intero n)* → *intero*

sia Fib un array di n interi

Fib[1] ← *Fib[2]* ← *1*

for i = 3 **to** n **do**

Fib[i] ← *Fib[i-1] + Fib[i-2]*

return *Fib[n]*

Calcolo del tempo di esecuzione

L'algoritmo *fibonacci3* impiega un tempo proporzionale a *n* invece che *esponenziale in n* come *fibonacci2*.

Ad esempio:

- per **n=45** vengono mandate in esecuzione 90 linee di codice, risultando così 38 milioni di volte più veloce di *fibonacci2*!!!!
- per **n=58** *fibonacci3* è circa 15 miliardi di volte più veloce di *fibonacci2!!!*

Tempo effettivo richiesto da implementazioni in C dei due 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

Occupazione di memoria

Il tempo di esecuzione non è la sola risorsa di calcolo che ci interessa. Anche la **quantità di memoria** necessaria può essere cruciale.

Se abbiamo un algoritmo lento (non troppo), dovremo solo attendere più a lungo per ottenere il risultato.

Ma se un algoritmo richiede più spazio di quello a disposizione, non otterremo mai la soluzione, indipendentemente dal tempo di attesa.

Algoritmo fibonacci4

fibonacci3 usa un array di dimensione n.

In realtà non ci serve mantenere tutti i valori di Fn precedenti, ma solo gli ultimi due, riducendo lo spazio a poche variabili in tutto:

```
algoritmo fibonacci4(intero n) → intero
    a ← b ← 1
    for i = 3 to n do
        c ← a+b
        a ← b
        b ← c
    return b
```

a rappresenta Fib[i-2]

b rappresenta Fib[i-1]

c rappresenta Fib[i]

Potenze ricorsive

fibonacci4 non è il miglior algoritmo possibile.

E' possibile dimostrare per induzione la seguente proprietà di matrici:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} = \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix}$$

Useremo questa proprietà per progettare un algoritmo più efficiente.

Algoritmo fibonacci5

algoritmo `fibonacci5`$(intero\ n) \rightarrow intero$

1. $M \leftarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
2. **for** $i = 1$ **to** $n - 1$ **do**
3. $\quad M \leftarrow M \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
4. **return** $M[0][0]$

PSEUDOCODICE !

Il tempo di esecuzione è ancora O(n).

Cosa abbiamo guadagnato?

Calcolo di potenze

Possiamo calcolare la n-esima potenza elevando al quadrato la (n/2)-esima potenza.

Se n è dispari eseguiamo una ulteriore moltiplicazione

Esempio:

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

Algoritmo fibonacci6

Tutto il tempo richiesto da fibonacci6 è speso nella funzione *potenzaDiMatrice* che calcola ricorsivamente la potenza della matrice elevando al quadrato la sua potenza (n/2)-esima.

algoritmo `fibonacci6`$(intero\ n) \rightarrow intero$

1. $M \leftarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
2. `potenzaDiMatrice`$(M, n-1)$
3. **return** $M[0][0]$

procedura `potenzaDiMatrice`$(matrice\ M,\ intero\ n)$

4. **if** ($n > 1$) **then**
5. `potenzaDiMatrice`$(M, n/2)$
6. $M \leftarrow M \cdot M$
7. **if** (n è dispari) **then** $M \leftarrow M \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

Riepilogo

	Tempo di esecuzione	Occupazione di memoria
`fibonacci2`	$O(2^n)$	O(n)
`fibonacci3`	O(n)	O(n)
`fibonacci4`	O(n)	O(1)
`fibonacci5`	O(n)	O(1)
`fibonacci6`	O(log n)	O(log n)

Morale

Progettare algoritmi efficienti può avere un effetto **drammatico** sull'incremento delle prestazioni. (Ricordiamoci, ad esempio, che se n vale un miliardo $\log_2 n$ sarà pari a 30 !!!)

Come misurare l'efficienza di un algoritmo?

- q.tà tempo di calcolo (tempo di CPU)
  indipendente dalle tecnologie e dalle piattaforme
- q.tà spazio

Non vogliamo valutare i dettagli della particolare istanza del problema, ma riferirci alla dimensione dell'istanza di ingresso.

Una stima dell'ordine di grandezza può darci le informazioni necessarie (notazione asintotica).