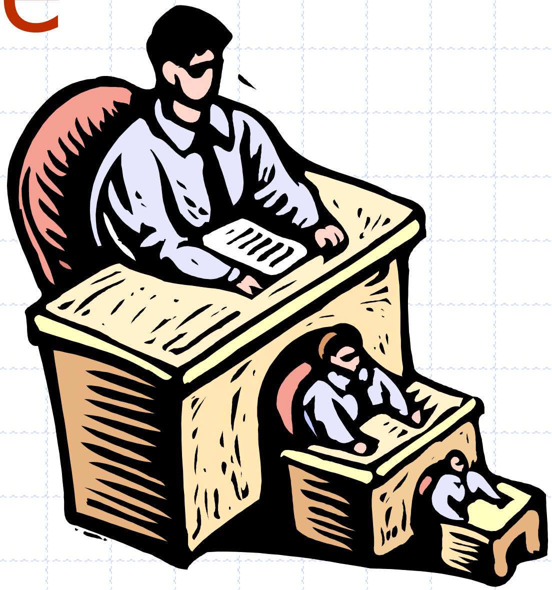
Usare le ricorsione



Richiamo dello schema classico

- **Ricorsione**: quando un metodo chiama se stesso
- Un classico esempio: la funzione fattoriale
  - n! = 1· 2· 3· ··· · (n-1)· n
- Definizione ricorsiva:

$$f(n) = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot f(n-1) & altrimenti \end{cases}$$

- Metodo Java:

```
// funzione fattoriale ricorsiva
public static int  recursiveFactorial(int n) {
   if  (n  ==  0)  return  1;      // caso base
   else return  n  *  recursiveFactorial(n- 1);  // caso ricorsivo
}
```

Ricorsione lineare

- **Verifica dei *casi base*.**
  - Verifica dapprima un insieme di casi di base (ce ne dovrebbe essere almeno uno).
  - Ogni possibile catena di chiamate ricorsive **deve** alla fine raggiungere uno dei casi base; la gestione di ciascun caso base non dovrebbe usare ricorsione.
- ***Una ricorsione.***
  - Eseguire una singola chiamata ricorsiva. (Questo passo ricorsivo può comportare un test per decidere quale fra più chiamate ricorsive debba essere eseguita, ma ciascuna volta dovrebbe scegliere esattamente una chiamata ricorsiva).
  - Definire ogni chiamata ricorsiva in modo di ottenere un effettivo avvicinamento al caso base.

Semplice esempio di ricorsione lineare

Algorithm LinearSum(A, n):

Input:

Un array di interi A e un intero $n \geq 1$, tali che A ha almeno n elementi

Output:

La somma dei primi n interi di A

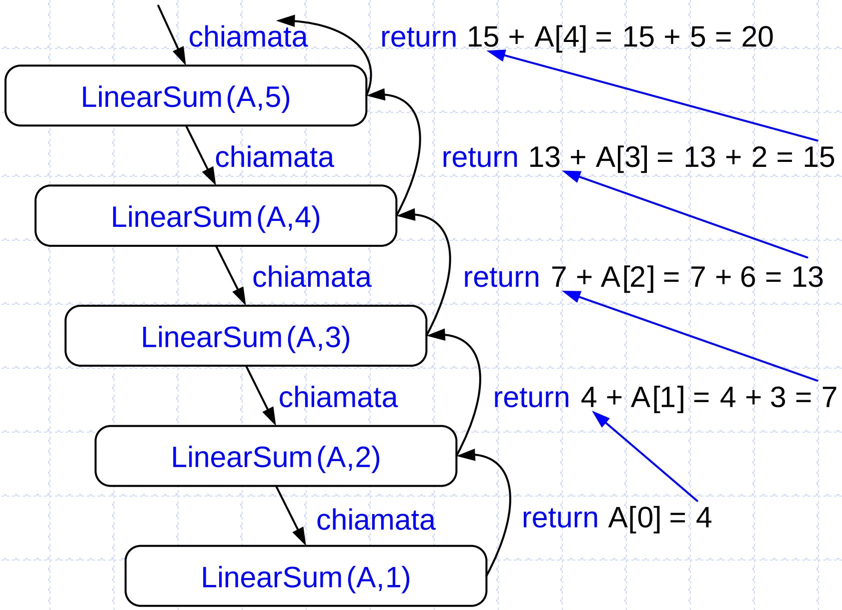
if $n = 1$ **then**

return $A[0]$

else

return LinearSum(A, n - 1) + $A[n - 1]$

Esempio di traccia di ricorsione:



Inversion di un array

Algorithm ReverseArray(*A*, *i*, *j*):

Input: Un array *A* e due indici interi non negativi *i* e *j*

Output (attraverso side-effect): Rovesciamento degli elementi di *A* ad iniziare dall'indice *i* fino a *j*

if $i < j$ **then**

Scambia $A[i]$ e $A[j]$

ReverseArray(A, $i + 1$, $j - 1$)

return

Individuare gli argomenti per la ricorsione

- Nel progetto di algoritmi ricorsivi è importante definire i metodi in modo tale da facilitare la ricorsione.
- Ciò a volte suggerisce l'introduzione di parametri addizionali da passare al metodo.
- Ad esempio, nel caso dell'inversione dell'array abbiamo scelto ReverseArray(A, i, j), piuttosto che ReverseArray(A).

Calcolo di potenze

- La funzione potenza, $p(x,n)=x^n$, può essere definita ricorsivamente:

$$p(x,n) = \begin{cases} 1 & \text{se } n = 0 \\ x \cdot p(x, n-1) & \text{altrimenti} \end{cases}$$

- La definizione conduce in maniera naturale ad un algoritmi ricorsivo che viene eseguito in tempo O(n) (poiché fa n chiamate ricorsive).
- Possiamo far meglio di ciò, tuttavia.

Calcolo attraverso elevamenti al quadrato ricorsivi

- Possiamo ottenere un algoritmo ricorsivo più efficiente usando ripetutamente l'elevamento al quadrato:

$$p(x,n) = \begin{cases} 1 & \text{se } x = 0 \\ x \cdot p(x,(n-1)/2)^2 & \text{se } x > 0 \text{ è dispari} \\ p(x,n/2)^2 & \text{se } x > 0 \text{ è pari} \end{cases}$$

- Esempi

$$2^4 = (2^{4/2})^2 = (2^2)^2 = ((2^{2/2})^2)^2 = ((2^1)^2)^2 = (((2\cdot(2^{0/2})^2)^2)^2 = (((2\cdot(2^0)^2)^2)^2 = (((2\cdot(1)^2)^2)^2 = (((2)^2)^2 = 4^2 = 16$$

$$2^5 = 2\cdot(2^{4/2})^2 = 2\cdot(2^2)^2 = 2\cdot((2^{2/2})^2)^2 = 2\cdot((2^1)^2)^2 = 2\cdot((2\cdot(2^{0/2})^2)^2)^2 = 2\cdot((2\cdot 1^2)^2)^2 = 2\cdot(2^2)^2 = 2\cdot(4^2) = 2\cdot 16 = 32$$

Un metodo ricorsivo basato sull'elevamento al quadrato

Algorithm Power(x, n):

Input: Un numero x e un intero $n \geq 0$

Output: Il valore x^n

if $n = 0$ **then**

  return 1

if n è dispari **then**

  $y = \text{Power}(x, (n - 1)/2)$

  return $x \cdot y \cdot y$

else

  $y = \text{Power}(x, n/2)$

  return $y \cdot y$

Analisi del metodo dei quadrati ricorsivi

Algorithm Power(x, n):
Input: Un numero x e un intero $n \geq 0$
Output: Il valore x^n
if $n = 0$ **then**
    return 1
if n è dispari **then**
    $y = \text{Power}(x, (n - 1)/2)$
    return $x \cdot y \cdot y$
else
    $y = \text{Power}(x, n/2)$
    return $y \cdot y$

Ogni volta che facciamo una chiamata ricorsiva dimezziamo il valore di n, perciò facciamo log n chiamate ricorsive. Dunque, il metodo viene eseguito in tempo O(log n).

E' importante usare una variabile due volte anziché chiamare il metodo due volte.

Ricorsione di coda

- La ricorsione di coda si verifica quando un metodo linearmente ricorsivo effettua la sua chiamata ricorsiva come ultimo passo.
- Il metodo per invertire un array ne è un esempio.
- Metodi di questo tipo possono essere facilmente convertiti in metodi ricorsivi, il che consente dei risparmiare risorse.
- Esempio:

Algorithm IterativeReverseArray(A, i, j):

Input: Un array A ed indici interi non negativi i e j

Output (attraverso side-effect): Rovesciamento degli elementi di A ad iniziare dall'indice i fino a j

while $i < j$ **do**

Scambia $A[i]$ e $A[j]$

$i = i + 1$

$j = j - 1$

return

Ricorsione binaria

- Si verifica ricorsione binaria quando ci sono **due** chiamate ricorsive per ciascun caso non base.
- Esempio: il metodo DrawTicks per tracciare trattini (tick) su di un righello (di tipo inglese).

```
---- 0
-
--
-
---
-
--
-
---- 1
-
--
-
---
-
--
-
---- 2
```

```
----- 0
-
--
-
---
-
--
-
----
-
--
-
---
-
--
-
----- 1
```

```
--- 0
-
--
-
--- 1
-
--
-
--- 2
-
--
-
--- 3
```

Un metodo per disegnare tick basato su ricorsione binaria

```
        // traccia un tick senza etichetta
public static void  drawOneTick(int  tickLength) {  drawOneTick(tickLength,  - 1);  }
        // traccia un tick
public static void  drawOneTick(int  tickLength,  int  tickLabel) {
    for  (int i =  0; i <  tickLength;  i++)
         System.out.print("-");
    if  (tickLabel  >=  0)  System.out.print(" "  +  tickLabel);
    System.out.print("\n");
}
public static void  drawTicks(int  tickLength)  {  // traccia tick di lunghezza assegnata
    if  (tickLength  >  0)  {                      // stop quando la lungh. scende a 0
         drawTicks(tickLength- 1);                 // traccia ricorsivamente tick a sx
         drawOneTick(tickLength);                  // traccia tick centrale
         drawTicks(tickLength- 1);                 // traccia ricorsivamente tick a dx
    }
}
public static void  drawRuler(int  nInches,  int  majorLength)  {  // disegna righello
    drawOneTick(majorLength,  0);                  // traccia tick 0 e la sua etichetta
    for  (int i =  1; i <=  nInches;  i++)         {
         drawTicks(majorLength- 1);                // traccia tick per il pollice corrente
         drawOneTick(majorLength,  i);             // traccia tick i e la sua etichetta
    }
}
```

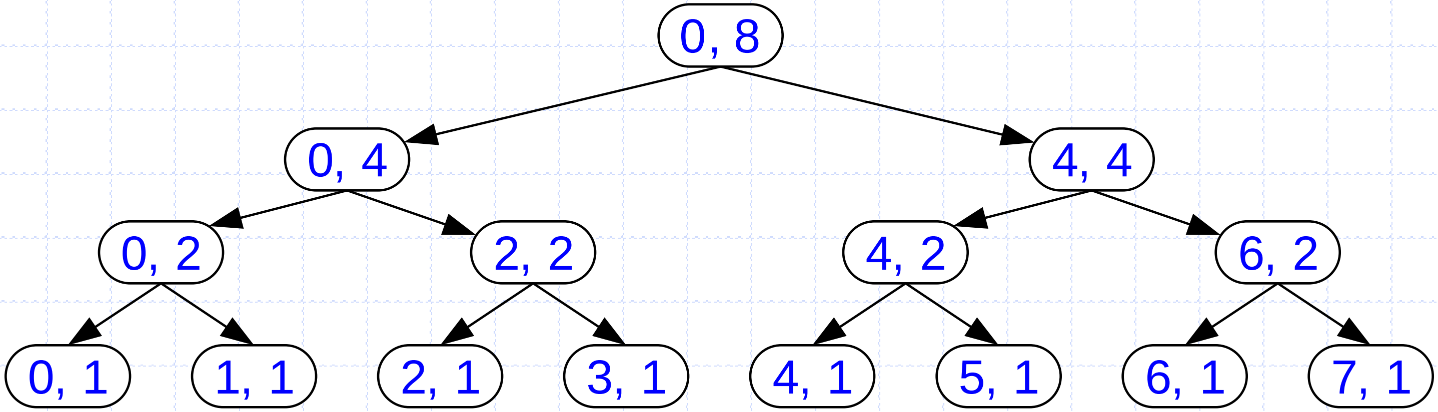
Notare le due chiamate ricorsive

Un altro metodo basato su ricorsione binaria

- Problema: sommare tutti i numeri in array di interi A:

Algorithm BinarySum(A, i, n):
  Input: Un array A e interi i ed n
  Output: La somma di n interi in A iniziando dall'indice i
 if $n = 1$ **then**
 return $A[i]$
 return BinarySum(A, i, $n/2$) + BinarySum(A, $i + n/2$, $n/2$)

- **Traccia di esempio:**



Calcolo dei numeri di Fibonacci

- I numeri di Fibonacci sono definiti ricorsivamente:

$F_0 = 0$

$F_1 = 1$

$F_i = F_{i-1} + F_{i-2}$ per $i > 1$.

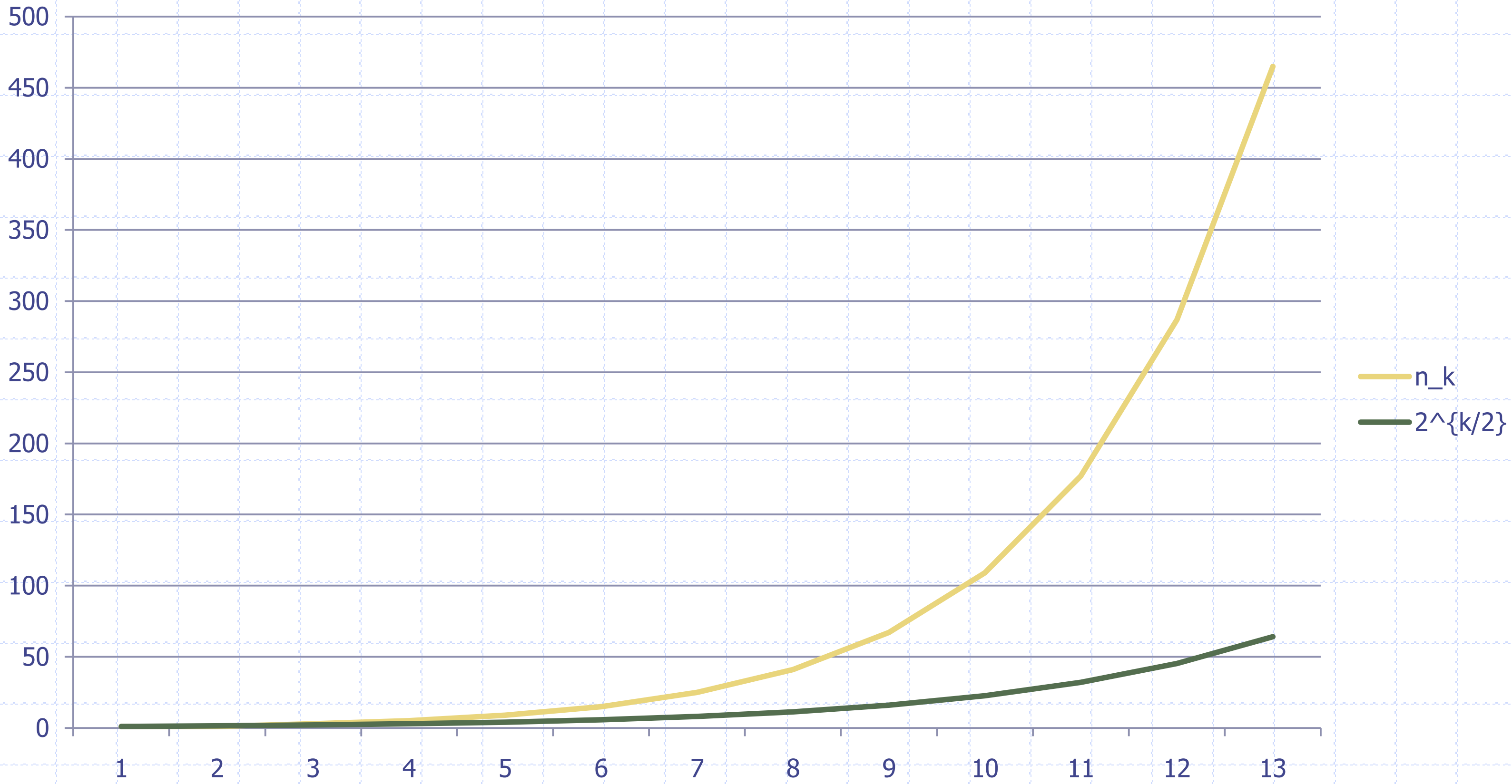
- Algoritmo ricorsivo (primo tentativo):

Algorithm BinaryFib(k):
 Input: Un intero non negativo k
 Output: Il k-esimo numero di Fibonacci F_k
 if $k = 1$ **then**
  return k
 else
  return BinaryFib(k - 1) + BinaryFib(k - 2)

Analisi della ricorsione binaria dell'algoritmo di Fibonacci

- Denotiamo con n_k il numero di chiamate ricorsive fatte da BinaryFib(k). Abbiamo:
  - $n_0 = 1$
  - $n_1 = 1$
  - $n_2 = n_1 + n_0 + 1 = 1 + 1 + 1 = 3$
  - $n_3 = n_2 + n_1 + 1 = 3 + 1 + 1 = 5$
  - $n_4 = n_3 + n_2 + 1 = 5 + 3 + 1 = 9$
  - $n_5 = n_4 + n_3 + 1 = 9 + 5 + 1 = 15$
  - $n_6 = n_5 + n_4 + 1 = 15 + 9 + 1 = 25$
  - $n_7 = n_6 + n_5 + 1 = 25 + 15 + 1 = 41$
  - $n_8 = n_7 + n_6 + 1 = 41 + 25 + 1 = 67.$
- Notiamo che $n_k > 2^{k/2}$. Esponenziale!

Confronto grafico



Un algoritmo migliorato per il cancolo dei numeri di Fibonacci

- Basato su ricorsione lineare:

```
Algorithm LinearFibonacci(k):
   Input: Un intero non negativo k
   Output: Coppia di numeri di Fibonacci (F_k, F_k-1)
  if k = 1 then
      return (1, 0)
  else
      (i, j) = LinearFibonacci(k - 1)
      return (i + j, i)
```

- Tempo di esecuzione $O(k)$.

Ricorsione multipla

- Esempio di applicazione: risoluzione di rompicapo aritmetico (a lettera uguale corrisponde cifra uguale)
  - *pot* + *pan* = *bib*
  - *dog* + *cat* = *pig*
  - *boy* + *girl* = *baby*
- Ricorsione multipla: fa potenzialmente molte chiamate ricorsive per ogni passo non base (non solo una o due).

Tradizionale schema algoritmico per ricorsione multipla

Algorithm PuzzleSolve(k,S,U):

Input: Un intero k, una sequenza S e un insieme U (l'universo di elementi da testare)

Output: Una enumerazione di tutte le estensioni di lunghezza k aggiunte ad S usando elementi in U senza ripetizioni

```
for all e in U do
    Rimuovi e da U  { ora usiamo e }
    Aggiungi e al termine di S
    if k = 1 then
            Test se S è una configurazione che risolve il rompicapo
            if S risolve il rompicapo then
                        return "Trovata la soluzione: " S
    else
            PuzzleSolve(k - 1, S,U)
    Re-inserisci e in U           { e non è più usato }
    Rimuovi e dalla fine di S
```

Visualizzazione di PuzzleSolve