Ricorsione

Walter Cazzola

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano

Ricorsione
Outline

- Definizioni
- La ricorsione come strumento di programmazione
- implementazione di metodi ricorsivi
- problemi tipici
- confronto tra ricorsione e iterazione
- alcuni esempi

Ricorsione
Definizione

Dal dizionario Garzanti (www.garzantilinguistica.it).

Lemma	ricorsivo
Etimologia	Deriv. di *ricorrere*, sul modello del fr. *récursif*
Definizione	agg. (mat.) si dice di una successione di funzioni ognuna delle quali si ricavi dalla precedente.

o considerando una definizione più pragmatica:

. . . ricorsivo: aggettivo, vedi ricorsivo . . .

Ricorsione
Definizione Intuitiva

Problema: Dover lavare una pila di piatti.

- approccio pigro :-)

Algoritmo:

- se il lavandino è vuoto non c'è niente da dover fare;
- se non è vuoto allora:
  - lavare il primo piatto e
  - fermare la persona più vicina e dirle di lavare i restanti piatti.

Nota, ogni nuova persona coinvolta dovrà eseguire un lavoro in qualche modo minore di quella che la coinvolge.

Ricorsione
Definizione: Funzione Ricorsiva

Una funzione viene detta ricorsiva se è definita in termini di sé stessa.

Esempio: Fattoriale.

- $5! = 5 * 4 * 3 * 2 * 1$
- Nota che: $5! = 5 * 4!$, $4! = 4 * 3!$ e così via.

Possibile computazione ricorsiva, dalla definizione matematica:

$$n! = \begin{cases} 1 & \text{se } n = 0, \\ n * (n-1)! & \text{altrimenti.} \end{cases}$$

Per $n = 0$ si ha il *caso base* (assioma) mentre il secondo caso rappresenta il *passo induttivo*.

Ricorsione
E nel caso di Java?

Un metodo si dice ricorsivo quando la sua esecuzione può prevedere una chiamata a sé stesso.

- in modo *diretto*, i.e. il corpo del metodo prevede una chiamata al metodo stesso;
- in modo *indiretto*, i.e. il corpo del metodo prevede chiamate a metodi che a loro volta chiamano il metodo originario.

```
class Factorial {
  public static int fact(int n) {
    int result;
    if (n<=1) result = 1;
    else result = n*fact(n-1);
    return result;
  }
  public static void main(String args[]) {
    int n = Integer.parseInt(args[0]);
    System.out.println(n+"! = "+fact(n));
  }
}
```


Ricorsione
Esecuzione: Cosa Succede?

```
[18:01]cazzola@ulik:>java Factorial 4
4!  = 24
```

```
class Factorial {
  public static int fact(int n) {
    int result;
    if (n<=1)
      result = 1;
    else result = n*fact(n-1);
    return result;
  }
  public static void main(String args[]) {
    int n = Integer.parseInt(args[0]);
    System.out.println(n+!  = +fact(n));
  }
}
```

Eseguo fact(4):
- si crea un frame con n = 4.
- n non è minore di 2.
- result = 4*fact(3)6, ritorno result = 24

Eseguo fact(3):
- si crea un frame con n = 3.
- n non è minore di 2.
- result = 3*fact(2)2, ritorno result = 6

Eseguo fact(2):
- si crea un frame con n = 2.
- n non è minore di 2.
- result = 2*fact(1)1, ritorno result = 2

Eseguo fact(1):
- si crea un frame con n = 1.
- result = 1, ritorno result = 1

Ricorsione
Note sull'Esecuzione.

Ad ogni invocazione il sistema crea un *record di attivazione* in cui memorizza i valori correnti
- delle variabili locali, dei parametri e la locazione del valore di ritorno.

La creazione del frame per ogni attivazione, permette di:
- tenere traccia dell'esecuzione;
- salvare lo stato corrente per ripristinarlo al ritorno dell'esecuzione;
- lavorare sulle variabili locali ad un'attivazione senza interferenze.

Attenzione:

Senza controllare il raggiungimento dell'assioma si continua ad applicare il passo induttivo.
- Il passo induttivo verrà applicato fintantoché la memoria a disposizione della virtual machine non viene saturata.

Ricorsione
Progettazione: Requisiti

Le soluzioni ricorsive si applicano meglio a problemi con possibile definizione induttiva.

Usare la soluzione di una versione più piccola/semplice del problema per arrivare alla soluzione del problema:

- Chiamata ricorsiva;

Saper riconoscere quando il problema è abbastanza semplice da essere risolto direttamente

- terminazione

Le soluzioni ricorsive sono in generale più eleganti ed intuitive delle soluzioni iterattive.

Ricorsione
Esempio: I Numeri di Fibonacci.

Nel 1223 a Pisa si svolse una gara tra abachisti e algoritmisti. Il quesito era:

> *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 la riproduzione inizi al secondo mese di vita?*

Leonardo Pisano, detto Fibonacci, vinse la gara indicando come soluzione una sequenza il cui generico elemento era pari alla somma dei due precedenti

Ricorsione
Esempio: I Numeri di Fibonacci.

La sequenza di Fibonacci ha una definizione ricorsiva:

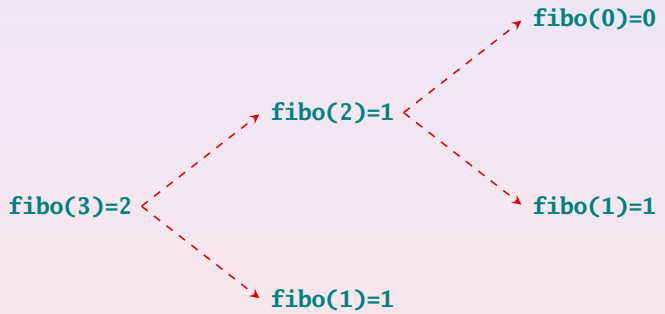
$$f(n) = \begin{cases} 0 & \text{se } n = 0, \\ 1 & \text{se } n = 1 \text{ o } n = 2, \\ f(n-1) + f(n-2) & \forall n > 2. \end{cases}$$

L'implementazione è conseguenza della definizione:

```
class Fibonacci {
  public static long fibo(long n) {
    if(n<=1) return n;
    else return fibo(n-1) + fibo(n-2);
  }

  public static void main(String args[]) {
    long n = Integer.parseInt(args[0]);
    System.out.println("fib(" + n + ")=" + fibo(n));
  }
}
```


Ricorsione
Esempio: I Numeri di Fibonacci (Albero di Esecuzione).



Ricorsione
Ricorsione + Semplice ed Elegante.

La soluzione ricorsiva è più intuitiva:

```
class Fibonacci {
  public static long fibo(long n) {
    if(n<=1) return n;
    else return fibo(n-1) + fibo(n-2);
  }
}
```

La soluzione iterativa è più oscura:

```
class FibonacciI {
  public static long fibo(long n) {
    long fibN_1 = 0, fibN_2 = 1, fibN=1;
    if(n<=1) return n;
    else {
      for(int i=2; i<=n; i++) { fibN=fibN_1+fibN_2; fibN_1=fibN_2; fibN_2=fibN; }
      return fibN;
    }
  }
}
```


Ricorsione
Iterazione + Efficiente.

La soluzione iterativa risulta molto più performante:

```
[23:41]cazzola@ulik:~>time java FibonacciI 100
fib(100)=3736710778780434371
0.032u 0.014s 0:00.07 57.1% 0+0k 0+0io 1pf+0w
```

L'esecuzione della stessa invocazione usando la versione ricorsiva dopo 10 minuti non era ancora terminata.

La creazione di record di attivazione si paga in termini di efficienza ma soprattutto in termini di memoria occupata.

Ad es. la chiamata `fibo(51308)`

- fornisce una risposta (errata) usando la versione iterativa;
- solleva un'eccezione `StackOverflowError` nel caso ricorsivo.

```
[0:01]cazzola@ulik:~>java FibonacciI 51308
fib(51308)=-1983210647217758115
[0:01]cazzola@ulik:~>java Fibonacci 51308
Exception in thread "main" java.lang.StackOverflowError
    at Fibonacci.fibo(Fibonacci.java:4)
```


Ricorsione
Ricorsione vs Iterazione

Ripetizione

- Iterazione : ciclo esplicito
- Ricorsione: chiamate ripetute di funzione

Terminazione

- Iterazione : fallisce la condizione del ciclo
- Ricorsione : passo base (livello assiomatico) riconosciuto

Possibilità di cicli infiniti per entrambe.

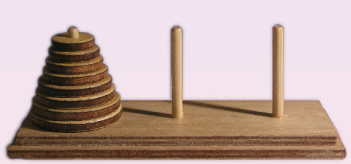
Bilancio

- Scelta tra efficienza (iterazione) e semplicità ed eleganza (ricorsione)

Le Torri di Hanoi
Definizione (Édouard Lucas, 1883)

Descrizione del Problema

Si hanno tre pali ed un certo numero di dischi forati al centro da impilare sui bastoni. I dischi hanno dimensione variabile e si può impilare i dischi solo su un disco più grande.



Il gioco consiste nello spostare, un disco alla volta, l'intera pila di dischi dal primo palo all'ultimo senza **mai** violare le regole.

Le Torri di Hanoi
Algoritmo Ricorsivo

Algoritmo per 3 Dischi

Algoritmo per n Dischi

Base: n=1, basta spostare il disco dalla sorgente (S) alla destinazione (D);

P. Ind.: spostare n-1 dischi da S al palo libero (L), spostare il rimanente disco in D, spostare gli n-1 dischi da L a D.

Le Torri di Hanoi
L'Implementazione in Java

```
public class Hanoi {
  int disks;
  int pegs[][];

  public Hanoi(int disks) {
    this.disks = disks;
    pegs = new int[3][disks];
    for(int i=0;i<disks;i++) { pegs[0][i] = i+1; pegs[1][i] = 0; pegs[2][i] = 0; }
    System.out.println("Start!");
    display();
    moveDisks(disks, 0, 2, 1);
  }

  public void moveDisks(int disks, int from, int to, int empty) {
    if(disks <= 1) {
      System.out.println("moving from "+from+" to "+to);
      move(from, to);
    } else {
      moveDisks(disks-1, from, empty, to);
      System.out.println("moving from "+from+" to "+to);
      move(from, to);
      moveDisks(disks-1, empty, to, from);
    }
  }
```


Le Torri di Hanoi
L'Implementazione in Java (Segue)

```
  public void move(int from, int to) {
    int i=0;
    while( (pegs[from][i] == 0) && (i<disks) ) i++;
    int tmpFrom = (i==disks) ? disks-1 : i;
    i=disks-1;
    while( (pegs[to][i] != 0) && (i>0) ) i--;
    pegs[to][i] = pegs[from][tmpFrom];
    pegs[from][tmpFrom] = 0;
    display();
  }

  public void display() {
    for(int j=0;j<disks;j++) {
      for(int i=0;i<3;i++) System.out.print(" "+pegs[i][j]+" ");
      System.out.println("");
    }
    System.out.println("");
  }

  public static void main(String args[]) {
    Hanoi h = new Hanoi(3);
  }
}
```


Le Torri di Hanoi
Esecuzione per 3 Dischi

```
[15:56]cazzola@ulik:~>java Hanoi
Start!              moving from 0 to 1   moving from 0 to 2   moving from 1 to 2
  1    0    0         0    0    0         0    0    0         0    0    0
  2    0    0         0    0    0         0    1    0         0    0    2
  3    0    0         3    2    1         0    2    3         1    0    3

moving from 0 to 2   moving from 2 to 1   moving from 1 to 0   moving from 0 to 2
  0    0    0         0    0    0         0    0    0         0    0    1
  2    0    0         0    1    0         0    0    0         0    0    2
  3    0    1         3    2    0         1    2    3         0    0    3
```

Le Torri di Hanoi

Leggenda

La leggenda vuole che dei monaci Buddisti devoti a Brahmā si dovessero cimentare col problema con 64 dischi in oro e che una volta risolto il mondo sarebbe finito.

Si può stare tranquilli?

Quante operazioni servirebbero per terminare l'algoritmo?

Ad ogni chiamata a `moveDisks()` si effettuano (almeno) due chiamate ricorsive alla funzione stessa. Che si prova essere approssimabile con 2^n.

Se si riuscisse a spostare un disco al secondo si avrebbe:

2^{64} = 18446744073709551616 secondi

cioè circa 586549 miliardi di anni, mentre l'età attualmente stimata dell'universo è: 13.7 miliardi di anni.