# Ricorsione

## Laboratorio di Programmazione II

Corso di Laurea in Bioinformatica
Dipartimento di Informatica - Università di Verona

# Sommario





- La ricorsione
- Implementazione di semplici funzioni ricorsive in Java
- Utilizzo ricorsione per processare dati in java
- Ricorsione vs. Iterazione
- Ricorsione multipla

# La ricorsione

# Ricorsione





## Algoritmi Ricorsivi

- Algoritmo Ricorsivo: richiama se stesso
- Tecnica di programmazione molto elegante e semplice da realizzare
- Tipicamente meno efficiente dell'iterazione (gestione chiamate ricorsive)
- Particolarmente adatta per domini definiti induttivamente.

# Domini definiti induttivamente



## Caratterizzazione induttiva degli elementi del dominio

1. Un numero finito di elementi appartengono al dominio
2. Un numero finito di regole se applicate ad un sottoinsieme degli elementi del dominio restituiscono elementi appartenenti al dominio stesso
3. Solo gli elementi cosí definiti appartengono al dominio.

## Example (Numeri naturali $N$)

1. 0 appartiene ad $N$
2. se $n \in N$ allora il successore di n appartiene ad $N$
3. niente altro appartiene ad $N$

# Ricorsione e domini definiti induttivamente



## Ricorsione su domini definiti induttivmente

Ricorsione puó manipolare ed elaborare in maniera molto conveniente gli elementi di un dominio definito induttivamente.

## Example (Fattoriale)

$$fatt(n) = \begin{cases} 1, & \text{if } n = 0 \quad \text{(caso base)} \\ n * fatt(n-1), & \text{if } n > 0 \quad \text{(caso ricorsivo)} \end{cases}$$

# Calcolo di funzioni ricorsive





## Schema generale

- La definizione ricorsiva ricalca la struttura della definizione induttiva del dominio su cui opera la funzione.
  1. uno (o più) casi base, per i quali il risultato può essere determinato direttamente;
  2. uno (o più) casi ricorsivi, per i quali si riconduce il calcolo del risultato al calcolo della stessa funzione su un valore più piccolo/semplice.
- Dominio definito induttivamente $\rightarrow$ prima o poi arriviamo ad uno dei casi base.

# Ricorsione in Java



## Scrivere programmi ricorsivi in Java

In Java la ricorsione e' supportata in maniera diretta: una funzione puo' richiamare se stessa

## Example (Fattoriale Ricorsivo in Java)

```
public static long fatt(int n) {
  if(n==0){
    return 1;
  }
  else {
    return n*fatt(n-1);
  }
}
```

# Somma Ricorsiva



## Somma ricorsiva in Java

Definizione ricorsiva:

$$somma(x, y) = \begin{cases} x, & \text{if } y = 0 \quad \text{(caso base)} \\ 1 + somma(x, y - 1), & \text{if } y > 0 \quad \text{(caso ricorsivo)} \end{cases}$$

## Example (Somma Ricorsiva in Java)

```
public static int sommaRicorsiva(int x, int y) {
  if(y==0){
    return x;
  }
  else {
    return 1 + sommaRicorsiva(x,y-1);
  }
}
```

# Esercizi Ricorsione



## Esercizio

Implementare i metodi

- moltiplicazione ricorsiva
- potenza ricorsiva

della classe FuzioniRicorsive.java.

# Processare dati utilizzando la ricorsione



## Schema processamento dati ricorsivo

Calcolare l'operazione op per la collezione di elementi *C*

```
se C e' vuota
   return elemento neutro di op
altrimenti {
   return primo elemento op risultato chiamata ricorsiva
}
```

# Ricerca di un elemento utilizzando la ricorsione



## Schema ricerca ricorsiva

Ricerca dell'elemento $x$ nell'insieme $S$

```
se l'insieme e' vuoto
   return false
altrimenti {
   return (primo elemento == elemento cercato) or
    elemento presente resto insieme
}
```

# Esempio: ricerca di un intero in un file di interi



## Ricerca ricorsiva

```
private static boolean trovaRicorsivoInput(int e,
  BufferedReader br) throws IOException {
  String line = br.readLine();
   if (line == null){
     return false;
   } else {
        int x = Integer.valueOf(line);
        return ((x == e) || trovaRicorsivoInput(e,br))
  }
}
```

# Esempio: ricerca di un carattere in una stringa





## Ricerca ricorsiva stringa

```
private static boolean
 trovaRicorsivoStringa(char c, String string) {
   System.out.println("trova ricorsivo:
    processo la stringa:"+string);
   if (string.length() == 0){
      return false;
   } else {
      return ((c == string.charAt(0))
       || trovaRicorsivoStringa(c, string.substring(1)));
   }
}
```

# Conteggio di elementi utilizzando la ricorsione



## Schema conteggio ricorsivo

Contare occorrenze di $x$ nel multi insieme $M$

```
se l'insieme e' vuoto
   return 0
altrimenti {
   se il primo elemento e' l'elemento cercato
      return 1 + numero occorrenze resto insieme;
   } else {
      return numero di occorrenze resto insieme
   }
}
```

# Esercizi



## Esercizi processare dati con la ricorsione

Implementare i metodi

1. contaRicorsivoInput
2. contaRicorsivoStringa
3. massimoRicorsivoInput

della classe ProcessaDatiRicorsione.java
Scaricare il file interi.txt

# Confronto tra iterazione e ricorsione





## Iterazione vs. ricorsione

- Tutti i metodi ricorsivi possono essere implementati in maniera iterativa simulando la ricorsione con apposite strutture dati (pile).
- Alcuni metodi ricorsivi possono essere implementati in maniera iterativa direttamente
- Metodi iterativi piu' efficienti: non dobbiamo gestire le chiamate delle funzioni ricorsive.

# Calcolo Fattoriale Iterativo





## Example (fattoriale iterativo)

```
public static long fattIter(int n) {
 long ris = 1;
 while(n>0){
    ris = ris * n;
    n--;
  }
  return ris;
}
```

# Esempio: copia inversa



## Example (Copia inversa)

Vogliamo leggere una serie di interi da input e stamparli dall'ultimo al primo

```
public static void copiaInversa(Scanner sc){
 if (sc.hasNextInt()){
  int a = sc.nextInt();
  copiaInversa(sc);
  System.out.println(a);
 }
}
```

# Copia inversa: discussione





## Tail recursion

- Copia inversa non puo' essere tradotta direttamente in un metodo iterativo
- Differenza fondamentale dai metodi visti fino ad ora: chiamata ricorsiva non e' l'ultima istruzione del metodo tail recursion.
- tail recursion alcuni compilatori traducono direttamente metodi ricorsivi con tail recursion in metodi iterativi (piu' efficienti).

# Copia inversa: i record di attivazione



## Ricorsione e Record di Attivazione

- Per tradurre copia inversa in un metodo iterativo abbiamo bisogno di una struttura di appoggio apposita (pila) per le righe lette
- Questa struttura e' direttamente disponibile ai metodi ricorsivi grazie ai Record di attivazione

# Gestione memoria a run time



## Gestione memoria di lavoro della JVM

- zona che contiene il Java bytecode (ovvero il codice eseguibile dalla JVM) -
  - determinata a tempo di esecuzione al momento del caricamento della classe
  - dimensione fissata per ogni metodo a tempo di compilazione
- heap: zona di memoria che contiene gli oggetti
  - cresce e descresce dinamicamente durante l'esecuzione
  - allocazione/deallocazione oggetti
- pila dei record di attivazione (o stack): zona di memoria per i dati locali alle funzioni (variabili e parametri) -
  - cresce e descresce dinamicamente durante l'esecuzione
  - viene gestita con un meccanismo a pila

# Heap e garbage collection





## Gestione Heap della JVM

In Java Il programmatore non gestisce direttamente lo Heap
Allocazione memoria:

- Un oggetto viene creato tramite l'operatore new e la chiamata al costruttore.
- La zona di memoria per l'oggetto stesso viene riservata nello heap.

Deallocazione memoria:

- Quando un oggetto non è più utilizzato da un programma, la zona di memoria può essere liberata e resa disponibile per nuovi oggetti
- Operazione effettuata automaticamente dal garbage collector, quando l'oggetto non è più accessibile.

# Deallocazione automatica della memoria





## Garbage Collector

- Componente JVM che decide quando un oggetto non e' piu' utilizzabile: quando non ci sono piu' riferimenti attivi.
- Invocato automaticamente dalla JVM
- L'invocazione del GC puo' essere forzata dal programmatore: System.gc()

# Pila di record di attivazione



## Gestione pila RdA

Pila (o stack): struttura dati accesso LIFO: Last In First Out
La JVM gestisce i RdA con una pila:

- ogni attivazione di metodo genera un nuovo RdA posto in cima alla pila
- al termine dell'attivazione del metodo il RdA viene rimosso dalla pila

# Informazioni RdA



## Informazioni associate a ciascun RdA

Ogni RdA contiene informazioni necessarie all'attivazione del metodo

- locazioni di memoria per parametri formali (incluso eventuale riferimento all'oggetto di invocazione)
- locazioni di memoria per le variabili locali
- il valore di ritorno dell'invocazione del metodo
- indirizzo di ritorno: indirizzo della locazione di memoria dove e' memorizzata la prossima istruzione da eseguire nel metodo chiamante

## Example (evoluzione record di attivazione Ricorsione)

Vedere file outputRdA.txt (esecuzione di EsempioRdA.java)

# Esempio copia inversa



## Gestione RdA

- gli interi sono memorizzate tramite occorrenze successive della variabile a nei RdA.
- Pila RdA: struttura dati temporanea nella quale memorizzare gli interi
- Implementazione iterativa $\rightarrow$ leggere tutti gli interi e memorizzarli, prima di stampare.
- Servirebbe quindi una struttura dati addizionale (non avendo pila RdA ricorsiva).

# Esempio: interi simmetrici





## definizione problema

- Data una sequenza di interi tutti positivi tranne per uno 0 in posizione centrale.
- La sequenza e' simmetrica se coincide con la sequenza invertita
- Dato un file con una sequenza di interi, uno per riga, con uno zero in posizione centrale.
- Determinare se il file rappresenta una sequenza simmetrica.

# Esempio: interi simmetrici





## caratterizzazione ricorsiva

- La sequenza costituita solo da 0 e' simmetrica
- Una sequenza n s m e' simmetrica se s e' una sequanza simmetrica ed n ed m sono due interi positivi uguali.
- niente altro e' una sequenza simmetrica.

# Esempio: interi simmetrici



## codice soluzione

```
private static boolean interiSimmetrici(BufferedReader br)
  throws IOException {
   String line = br.readLine();
   int n = Integer.valueOf(line);
   if (n==0){
      return true;
   } else {
      boolean sim = interiSimmetrici(br);
      String line2 = br.readLine();
      int m = Integer.valueOf(line2);
      return ((sim) && (m == n));
   }
}
```

# Esercizi





## Esercizi ricorsione avanzata

Implementare i seguenti metodi

- interiInversi
- stringaPalindroma

della classe RicorsioneAvanzata.java

Nota: per eseguire il main e verificare la corretta esecuzione del programma scaricare i file

- copiaInversa.txt
- interiNonSimmetrici.txt (opzionale interiSimmetrici.txt)
- interiInversi.txt

# Ricorsione Multipla

# La Ricorsione multipla



## Definizione

- Si ha ricorsione multipla quando l'attivazione di un metodo puo' causare piu' di un attivazione ricorsiva dello stesso metodo.

## Example (calcolo dell'n-esimo numero di Fibonacci)

- Stima del numero di individui di una popolazione al variare delle generazioni
- $F(n)$ numero individui alla generazione n-esima.

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \quad \text{(caso base)} \\ 1, & \text{if } n = 1 \quad \text{(caso base)} \\ F(n-2) + F(n-1), & \text{if } n > 1 \quad \text{(caso ricorsivo)} \end{cases}$$

# La Ricorsione multipla in Java



## Example (calcolo dell'n-esimo numero di Fibonacci in Java)

```
private static int fibonacci(int n) {
   if (n==0){
      return 0;
   } else if (n==1){
      return 1;
   } else {
      return fibonacci(n-2) + fibonacci(n-1);
   }
}
```

# Esempio Torre Hanoi



## Il problema della torre di Hanoi

- Dati tre perni ed n dischi di dimensione crescente
- Situazione iniziale: tutti i dischi sono posizionati su un perno 1 (ciascun disco poggia sempre su un disco piu' grande)
- Obiettivo: spostare tutti i dischi sul perno 2 usando il perno 3 come appoggio
- Regole:
  1. Tutti i dischi (tranne quello spostato) debbono essere posizionati in uno dei tre perni.
  2. Si puo' spostare un solo disco alla volta, dalla cima di una torre ad un altro perno.
  3. Un disco non puo' mai poggiare su un disco piu' piccolo.

# Esempio Torre Hanoi



## Torre Hanoi soluzione ricorsiva

- per spostare n $>$ 1 dischi da 1 a 2, usando 3 come appoggio:
  - sposta n - 1 dischi da 1 a 3
  - sposta l'n-esimo disco da 1 a 2
  - sposta n - 1 dischi da 3 a 2
- un solo disco puo' essere spostato direttamente.

# Numero attivazioni



## Numero attivazioni di una metodo ricorsivo per ricorsione multipla

- Quando si utilizza la ricorsione multipla il numero di attivazioni di un metodo potrebbe essere esponenziale.
- Esponenziale nella profondita' delle chiamate ricorsive (altezza massima pila RdA)
- Esempio Torre Hanoi:
  - att(n) = numero di attivazioni del metodo ricorsivo che risolve una torre di hanoi di n dischi
  - $$att(n) = \begin{cases} 1, & \text{if } n = 1 \quad \text{(caso base)} \\ 1 + 2 * att(n-1), & \text{if } n > 1 \quad \text{(caso base)} \end{cases}$$
  - in generale $att(n) > 2^{(n-1)}$
- Nota: nel caso della torre di hanoi il numero esponenziale di attivazioni e' una caratteristica del problema
- non esiste una soluzione migliore

# Numero attivazioni Fibonacci ricorsivo





## Calcolo n-esimo numero di Fibonacci

- Si dimostra che $attFibo(n) > 2^{n/2}$ per $n > 1$
- Utilizzando le equazioni di ricorrenza ed il metodo di soluzione iterativo
- Quindi il metodo di Fibonacci ricorsivo ha costo (almeno) esponenziale
- Il problema di calcolare l'n-esimo numero di Fibonacci ha una soluzione migliore (lineare in n)

# Esercizio: torre Hanoi



## Implementare soluzione torre hanoi ricorsiva in java

- Vogliamo realizzare un programma che stampa la sequenza di spostamenti da fare per risolvere una torre di hanoi di n dischi.
- Per ogni spostamento vogliamo stampare un testo del tipo: muovi un disco dal perno x al perno y
- implementare il metodo
  - public static void muovi(int dischi, int sorg, int dest, int aux)

della classe RicorsioneMultipla.java.
Utilizzare il metodo muoviUnDisco(...) della stessa classe.