Java e gli algoritmi di ordinamento

In questo primo articolo della serie introdurremo i concetti base e studieremo un primo algoritmo dimostrativo.

Prima puntata

di Fabrizio Ciacchi

Un algoritmo è un elenco di istruzioni sequenziali che permettono la soluzione di un problema. In sostanza, il mondo della programmazione è pieno di algoritmi, i quali fanno le operazioni più disparate; si può pensare ad un programma come ad un insieme di algoritmi che cooperano per raggiungere un risultato finale. In questo senso esistono algoritmi più o meno efficienti, ecco perché c'è bisogno di misurare questa efficienza al di là del singolo programma o linguaggio di programmazione. Ci vuole una metodologia ben precisa per fare un'analisi statica del programma, in modo da sapere a priori la velocità probabile con cui funzionerà. Questa metodologia prende il nome di Analisi della complessità.

Fabrizio Ciacchi fabrizio@ciacchi.it

È impiegato come consulente per Vodafone. Tra le sue maggiori aree di interesse ci sono GNU/Linux, il linguaggio di programmazione Java e lo sviluppo di siti web in PHP. Nel tempo libero scrive articoli su GNU/Linux e legge libri di Asimov.
Il suo sito è http://fabrizio.ciacchi.it

Basi di Algoritmica

Il calcolo della complessità di un algoritmo (e più in generale del codice di un programma) è da considerarsi un calcolo asintotico della sua velocità; per capirsi, se io ho un'istruzione, ad esempio un assegnamento, posso dire che ha complessità O(1), cioè impiega un tempo finito per svolgere tale operazione. Quando abbiamo a che fare con dei cicli, invece, questi lavorano su un input che non possiamo conoscere a priori. Viene quindi posta una complessità O(n), dove n, appunto, rappresenta la dimensione dell'input, ciò ad identificare che la velocità di esecuzione dipende, per la maggior parte, dagli input che

> In informatica (e non solo) uno dei problemi ricorrenti è il problema dell'ordinamento

prenderà. Il calcolo della complessità finale è abbastanza semplice: si sommano o si moltiplicano le varie complessità come in un'equazione matematica, ottenendo così il risultato finale.

In informatica (e non solo) uno dei problemi ricorrenti è il problema dell'ordinamento (sorting problem), il quale consiste nell'ordinare in modo crescente un insieme di valori. La complessità minima con cui può essere implementato un algoritmo di ordinamento è $O(n \bullet \log n)$, la complessità massima è, invece, di $O(n^2)$, corrispondente a due cicli for. Inoltre studiare il problema dell'ordinamento è un buon metodo per imparare a capire e progettare gli algoritmi, questo perché è abbastanza semplice da poter essere facilmente implementato con numerosi algoritmi, ma sufficientemente complesso da poterne studiare la complessità.

Un algoritmo può essere iterativo se, in pratica, lavora facendo dei cicli, anche annidati, oppure ricorsivo quando richiama se stesso; si può ben capire che l'implementazione di un algoritmo in modo iterativo o ricorsivo può cambiare drasticamente il metodo con cui esso viene scritto. Gli algoritmi di ordinamento, come ogni altro algoritmo, possono essere divisi in:

- Algoritmi di ordinamento iterativi
  - Insertion Sort
  - Heap Sort
  - Bubble Sort (anche ricorsivo)
- Algoritmi di ordinamento ricorsivi
  - Merge Sort
  - Quick Sort

Adesso spiegheremo come funziona l'applet Java che mostra a video l'algoritmo di ordinamento, e poi porremo l'attenzione su uno dei più semplici programmi di ordinamento: l'Insertion Sort.

L'applet Java

La seguente porzione di codice non fa altro che creare l'immagine degli oggetti da ordinare, e, in base all'algoritmo scelto, ordinarli, mostrando a video la sua esecuzione. Le particolarità di questa classe sono il fatto che deve essere richiamata come applet all'interno di una pagina html, ed il fatto che, in pratica, viene implementata come thread, ovvero un'entità di programmazione autonoma che lavora concorrentemente ad altri processi. Apriamo l'editor di testi preferito ed iniziamo a scrivere il contenuto del file SortItem.java:

```
import java.awt.*;
import java.io.InputStream;
import java.util.Hashtable;
import java.net.*;

public class SortItem extends java.applet.
                Applet implements Runnable {
```

Si può notare che vengono importate diverse librerie e, cosa più importante, l'inizializzazione della classe è diversa da quella che si fa di solito, infatti si estende la classe Applet (per poter far funzionare la classe come tale) e si implementa l'interfaccia Runnable (senza la quale non si potrebbe lavorare con i thread).

> **Studiare il problema dell'ordinamento è un buon metodo per imparare a capire e progettare gli algoritmi**

Come prima cosa vengono definite le variabili utilizzate all'interno della classe, che sono: la variabile che tiene traccia del thread della classe, l'array di oggetti che dovranno essere ordinati, i due indici di ordinamento, il nome dell'algoritmo e l'oggetto di riferimento rappresentante l'algoritmo vero e proprio.

```
private Thread kicker;
int arr[];
int h1 = -1;
int h2 = -1;
String algName;
SortAlgorithm algorithm;
```

Il primo metodo che viene presentato è scramble(), il quale ha il compito di creare un array di dimensione variabile con elementi dal valore casuale,

```
void scramble() {
```

```
int a[] = new int[size().height / 2];
double f = size().width / (double) a.length;
for (int i = a.length; --i >= 0;) {
a[i] = (int)(a.length * f * Math.random()); }
arr = a; }
```

Particolare attenzione va posta all'assegnamento delle varie posizioni dell'array; infatti ogni elemento può assumere solo valori compresi tra zero e la lunghezza dell'array; ciò assicura che durante l'ordinamento si abbiano dei valori coerenti ed uniformi per tutti gli intervalli. Infine si copia l'array nella variabile locale arr, in modo da renderlo disponibile agli altri metodi della classe, e si chiude il metodo.

Il prossimo metodo è pause(), di cui esistono ben tre versioni, la prima senza parametri, la seconda con un solo valore, la seconda a due valori. Se non vengono messi parametri si richiama la versione a due parametri, passando ad entrambi il valore -1:

```
void pause() {
pause(-1, -1); }
```

Nella versione ad un parametro, si compie sempre l'operazione di richiamo a quella a due parametri, ma passandogli la variabile H1, la quale fa riferimento alla variabile locale h1,

```
void pause(int H1) {
pause(H1, -1); }
```

La versione completa di pause() prende in input due variabili intere (H1 ed H2), le quali rappresentano due indici, il primo, h1 (in blu), è quello superiore che fa da limite tra gli elementi già ordinati e quelli non ancora ordinati, il secondo, h2, è quello inferiore (contraddistinto nell'applet da una linea rossa) che visualizza lo scorrimento dell'algoritmo (in pratica quale elemento si sta processando); per disegnare la loro posizione vengono aggiornati tali indici con i parametri passati e viene chiamata la funzione repaint().

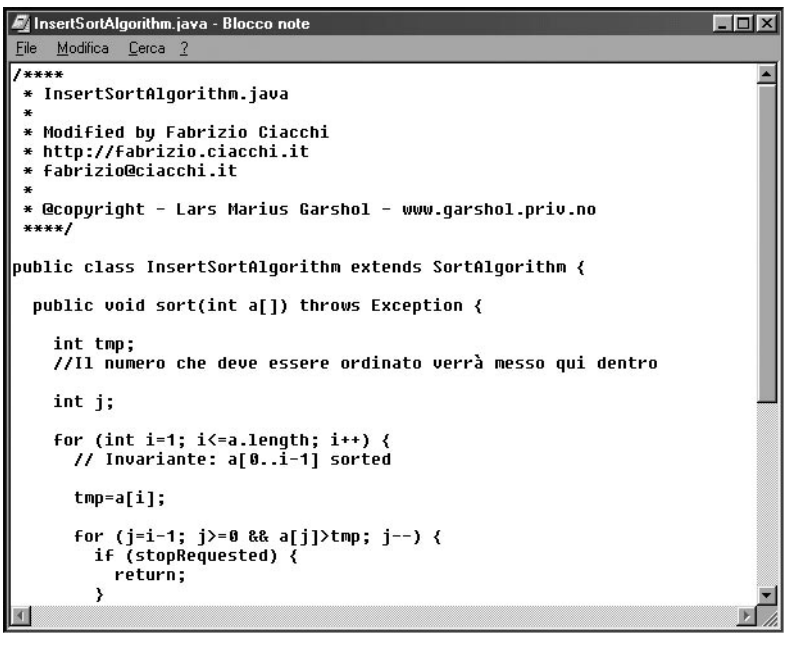
```
void pause(int H1, int H2) {
h1 = H1;
h2 = H2;
if (kicker != null) { repaint(); }
try {Thread.sleep(20);} catch
                    (InterruptedException e){}
}
```

Il metodo init() non fa altro che settare il tipo di algoritmo usato. Come già detto all'applet può essere passato il nome dell'algoritmo da usare, nel caso manchi viene usato un algoritmo di default (in questo caso lo stesso che richiamiamo, visto che, per ora non abbiamo implementato altri algoritmi).

```
public void init() {
String at = getParameter("alg");
if (at == null) { at = "InsertSort"; }
algName = at + "Algorithm";
scramble();
resize(100, 100);
}
```

Il metodo paint() rappresenta il fulcro centrale della classe, infatti esso è adibito a disegnare gli

FIGURA 1 Il file InsertSortAlgorithm.java che implementa l'algoritmo Insertion Sort.



elementi da ordinare, viene richiamato ogni qual volta avviene un cambiamento (viene ordinato un elemento, vengono aggiornati gli indici).

```
public void paint(Graphics g) {
 int a[] = arr;
 int y = size().height - 1;
 g.setColor(Color.lightGray);
 for (int i = a.length; --i >= 0; y -= 2) {
  g.drawLine(arr[i], y, size().width, y); }
 g.setColor(Color.black);
 y = size().height - 1;
 for (int i = a.length; --i >= 0; y -= 2) {
  g.drawLine(0, y, arr[i], y);}
 if (h1 >= 0) {
  g.setColor(Color.red);
  y = h1 * 2 + 1;
  g.drawLine(0, y, size().width, y); }
 if (h2 >= 0) {
  g.setColor(Color.blue);
  y = h2 * 2 + 1;
  g.drawLine(0, y, size().width, y); }
}
```

Senza addentrarci troppo nella struttura del paint(), basti sapere che, in base all'array degli elementi ed alla dimensione dell'applet, questi vengono disegnati (assieme agli indici) nella finestra dell'applet. Da notare che, prima di scrivere il codice per disegnare qualcosa, viene impostato il colore adatto; nel nostro caso il grigio (lightgray) rappresenta il colore di sfondo dell'applet, il nero (black) viene usato per gli elementi da ordinare, il rosso (red) per l'indice inferiore ed il blu (blue) per l'indice superiore. Il metodo update() non fa altro che richiamare nuovamente la funzione di disegno paint():

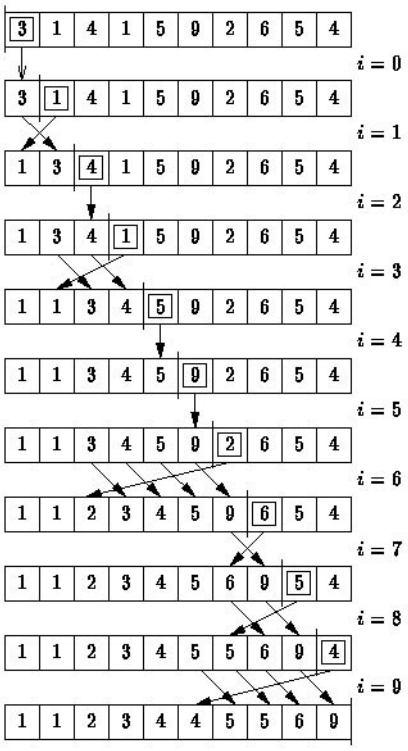
```
public void update(Graphics g) {
 paint(g); }
```

Quando viene richiamato il thread kicker tramite il comando start() (come vedremo più avanti), viene avviato il metodo pubblico run():

```
public void run() {
 try {
```

Viene quindi aperto un blocco try per catturare eventuali eccezioni generiche; le istruzioni nel corpo del metodo, come prima cosa, controllano se è stato scelto il tipo di algoritmo, in caso negativo viene creato un thread della classe di

FIGURA 2 Il funzionamento dell'algoritmo Insertion Sort rappresentato graficamente.



ordinamento (nel nostro caso InsertSortAlgorithm), il cui nome viene ricavato dal parametro alg passato all'applet (InsertSort).

```
if (algorithm == null) {
 algorithm = (SortAlgorithm)Class.
                forName(algName).newInstance();
 algorithm.setParent(this); }
```

Il thread creato dalla classe InsertSortAlgorithm ha a disposizione tutti i metodi della classe SortAlgorithm, come init() (che come vedremo più avanti setta una variabile), più il proprio metodo di ordinamento sort(), che avvia l'algoritmo vero e proprio. Infine si mette il comando catch.

```
  algorithm.init();
  algorithm.sort(arr);
 } catch(Exception e) {}
}
```

Il metodo stop(), invece, ha il compito di fermare tutti i thread attivi, cioè le istanze della classe SortItem e quelle della classe di ordina-

mento che ha esteso SortAlgorithm (nel caso specifico InsertSortAlgorithm)

```
public synchronized void stop() {
```

Se esiste un kicker (un'istanza della classe stessa) questo viene fermato ed, eventualmente, viene ignorata l'eccezione da lui generata; analogamente viene fermato il thread relativo alla classe di ordinamento:

```
if (kicker != null) {
 try { kicker.stop(); }
 catch (IllegalThreadStateException e) {}
 kicker = null; }

if (algorithm != null){
 try { algorithm.stop(); }
 catch (IllegalThreadStateException e) {}
} }
```

Adesso viene creato il metodo che ha il compito vero e proprio di iniziare l'esecuzione dell'applet; il metodo è privato e synchronized, quest'ultimo attributo per dire ai thread della classe che vi possono accedere solo uno alla volta,

```
private synchronized void startSort() {
```

Se non esistono thread attivi (variabile kicker) viene avviata una nuova sessione di ordinamento, la quale per prima cosa crea un array con tanti elementi mischiati, poi ridisegna lo schermo, crea un nuovo thread e lo avvia:

```
 if (kicker == null
|| !kicker.isAlive())
{
  scramble();
  repaint();
  kicker = new
Thread(this);
  kicker.start();}
}
```

L'ultima parte di codice ha il compito, invece, di iniziare l'esecuzione dell'applet (richiamando il metodo StartSort() appena spiegato) quando si clicca sopra ad essa con il mouse,

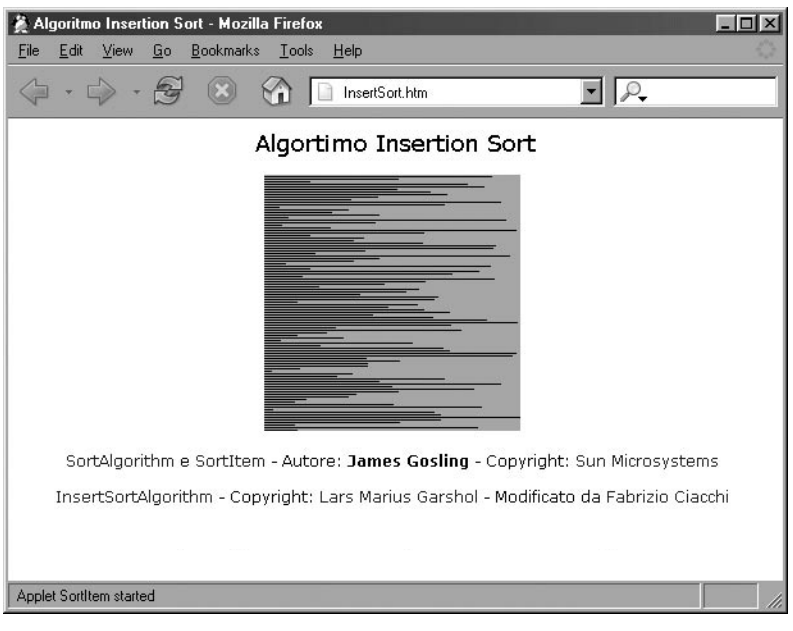
```
 public boolean mouseUp(java.awt.Event evt,
                                int x, int y) {
  startSort();
  return true;
 }
}
```

Per chi avesse un po' di confusione in testa, in pratica il funzionamento dell'applet è il seguente: la pagina html richiama l'applet SortItem, dandogli come parametro il nome dell'algoritmo da usare. La classe SortItem richiama a sua volta la classe dell'algoritmo (nel nostro caso InsertSortAlgorithm), passandogli come parametro se stessa. InsertSortAlgorithm estende quindi la classe SortAlgorithm, rendendo di fatto disponibili tutti i metodi necessari al corretto funzionamento dell'applet.

L'algoritmo di ordinamento generico

L'applet in Java sopra presentata ha il compito di mostrare a video gli elementi mentre vengono ordinati. Come già detto la particolarità di questa applet è che può mostrare l'esecuzione di qualsiasi programma di ordinamento, rendendo quindi più facile lo studio dell'algoritmo usato. Per fare ciò, però, bisogna scrivere un programma

FIGURA 3 Mozilla Firefox mostra l'algoritmo Insertion Sort all'opera.



di ordinamento generico che farà da interprete tra l'algoritmo utilizzato e l'applet. Apriamo il nostro editor e creiamo un file vuoto di nome SortAlgorithm.java, e scriviamo l'intestazione della classe:

```
class SortAlgorithm {
 private SortItem parent;
 protected boolean stopRequested = false;
```

Come si può notare la prima cosa che fa la classe è quella di creare un oggetto SortItem (l'applet), in modo da potergli passare i parametri necessari alla visualizzazione; viene anche creata una variabile booleana, il cui uso verrà spiegato più avanti. Continuando viene creato un metodo pubblico che ha il compito di prendere in input un oggetto di tipo SortItem, metodo che serve esclusivamente per fare in modo che non si perda l'oggetto sul quale stiamo lavorando.

```
public void setParent(SortItem p) {
 parent = p; }
```

Adesso dobbiamo definire il metodo pause() per diversi input, analogamente a quanto abbiamo fatto per la classe SortItem; il codice presentato non fa altro che richiamare il metodo pause() di tale classe.

```
protected void pause() throws Exception {
 if (stopRequested) { throw new Exception
                          ("Sort Algorithm"); }
  parent.pause(parent.h1, parent.h2); }

protected void pause(int H1) throws Exception {
 if (stopRequested) { throw new Exception
                          ("Sort Algorithm"); }
  parent.pause(H1, parent.h2); }

protected void pause(int H1, int H2) throws
                                    Exception {
```

Installare Java
Per poter eseguire correttamente il programma, è necessario il JRE (Java Runtime Enviroment, www.java.com) ed un browser adatto a visualizzare le applet, come Internet Explorer o Mozilla Firefox (www.mozilla.org). Per compilare il programma, invece, è necessario il tool di sviluppo JDK (Java Development Kit), anch'esso scaricabile gratuitamente da www.java.com.

Codice e Licenze
L'applet SortItem e la classe SortAlgorithm sono state create da James Gosling e sono di proprietà della Sun Microsystems. Il codice è disponibile e modificabile gratuitamente per scopo non commerciale o commerciale senza fini di lucro, senza nessun supporto e a patto di non modificare il copyright originale. La classe InsertSort è stata creata da Lars Marius Garshol ed è stata modificata dall'autore dell'articolo.

Compilazione ed Uso
I tre file creati (SortItem.java, SortAlgorithm.java, InsertSortAlgorithm.java) devono essere compilati affinché sia possibile eseguirli come applet Java. Per compilare i tre file in una volta sola, basta dare il seguente comando:

```
javac InsertSortAlgorith.java
```

A questo punto bisogna creare una pagina html contenente i seguenti tag:

```
<applet codebase="./" code=SortItem.class width=200 height=200>
  <param name="alg" value="InsertSort">
</applet>
```

dove codebase identifica la cartella che contiene le classi Java (se ad esempio si carica l'applet da un percorso differente), code deve contenere il nome della classe SortItem, e width ed height rappresentano la dimensione a video. Per richiamare l'algoritmo specifico bisogna inserire il tag <param> passandogli il parametro alg come nome del parametro ed InsertSort (cioè la prima parte del nome della classe) come suo valore. Per visualizzare l'applet aprire il file html creato con il proprio browser internet.

RIQUADRO 1

```
if (stopRequested) { throw new Exception
                              ("Sort Algorithm");}
  parent.pause(H1, H2); }
```

Infine viene creato il metodo stop(), che ha il compito di fermare l'esecuzione dell'ordinamento, il metodo init() che, al contrario, avvia l'ordinamento, ed il metodo sort(), il quale prende in input un array e, come si può notare, è vuoto; questo perché l'algoritmo vero e proprio verrà implementato in un altro file.

```
public void stop() {
  stopRequested = true; }

public void init() {
  stopRequested = false; }

void sort(int a[]) throws Exception {}
}
```

Un algoritmo può essere iterativo se, in pratica, lavora facendo dei cicli, anche annidati, oppure ricorsivo quando richiama se stesso

Il primo algoritmo: l'Insertion Sort

L'insertion Sort è uno dei più intuitivi e migliori algoritmi per ordinare pochi elementi. Il metodo è molto simile a quello usato quando si mettono a posto le carte da gioco di un determinato seme. Prendiamo una carta alla volta e, in base al valore della carta, andiamo a metterla nella giusta posizione, facendo scorrere di conseguenza tutte le altre. Schematizzato secondo un linguaggio di programmazione (come mostrato in figura), si fanno due cicli: il primo scorre tutti gli elementi, il secondo tiene traccia dell'elemento corrente e di tutti quelli antecedenti; il vero lavoro viene fatto nel corpo del secondo ciclo, infatti se l'elemento trovato è maggiore di quello corrente, i due vengono scambiati, e cosi a ritroso fino a quando non si trova un elemento adiacente minore, in quel caso l'elemento che stavamo spostando è nella giusta posizione.

La complessità media di questo algoritmo è $O(n^2)$, questo semplicemente perché, per poter arrivare al risultato finale dobbiamo fare due cicli for. Possiamo facilmente creare il codice Java dell'algoritmo (salvare il tutto in un file di nome InsertSortAlgorithm.java).

```
public class InsertSortAlgorithm extends
                                  SortAlgorithm {
 public void sort(int a[]) throws Exception {
  int tmp, j;
  for (int i=1; i<=a.length; i++) {
   tmp=a[i];
   for (j=i-1; j>=0 && a[j]>tmp; j--) {
    if (stopRequested) { return; }
    a[j+1]=a[j];
    pause(j,i);
   }
   a[j+1]=tmp;
  }
 }
}
```

Il codice dell'algoritmo è molto semplice ed è stato possibile implementarlo in modo molto simile alla sua descrizione, grazie alle due classi create inizialmente, le quali, come già detto, fanno tutto il lavoro sporco per poter dare vita all'applet. Per poter implementare altri algoritmi, quindi, basterà creare altri file che estendono la classe SortAlgorithm, avendo l'accortezza di far coincidere i nomi della classe e dei parametri dell'applet.

Conclusioni

In questo articolo abbiamo introdotto i concetti base di studio degli algoritmi e dell'analisi dei programmi di ordinamento. Questo tipo di studio è bagaglio importante per chi sviluppa in Java; le applicazioni pratiche sono molte, basti pensare alla manipolazione dei dati estratti da un database. L'autore rimane a disposizione per qualsiasi suggerimento o chiarimento sull'argomento.

CODICE ALLEGATO

ftp.infomedia.it  **Ordinamento1**