Java e gli algoritmi di ordinamento

In questa ultima puntata presentiamo quello che può essere considerato il miglior algoritmo di ordinamento, il Quicksort.

Quarta puntata

di Fabrizio Ciacchi

Nelle scorse puntate abbiamo parlato dei vari algoritmi di ordinamento. Abbiamo presentato un algoritmo "base" – l'Insertion Sort – un algoritmo con metodologia ricorsiva – il Merge Sort – e quello che utilizza una struttura dati ausiliaria – l'Heap Sort.

Abbiamo visto che è importante sia progettare un algoritmo più raffinato (e più intelligente), nonché utilizzare un'adeguata struttura dati per poter risparmiare la memoria utilizzata dall'algoritmo.

Il Quicksort è un algoritmo di tipo ricorsivo che non utilizza una struttura dati particolare. Il suo vantaggio è di essere uno degli algoritmi più efficienti e meno dispendiosi di memoria. Il Quicksort, per quanto efficiente, è tuttavia un algoritmo estremamente "fragile", in quanto errori nella sua implementazione possono portare ad un drastico calo delle prestazioni.

Fabrizio Ciacchi fabrizio@ciacchi.it

È uno sviluppatore e consulente informatico. 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.

La teoria del Quicksort

Il Quicksort è un algoritmo di ordinamento ideato da Charles Antony Richard Hoare nel 1960 ed è uno di quelli che, in genere, ha le prestazioni migliori. Utilizza la stessa tecnica del "Divide et Impera" del Merge Sort, richiede pochissime risorse e opera direttamente sui dati da ordinare (algoritmo *in place*). Purtroppo non è un algoritmo stabile, cioè non preserva eventuali dati già ordinati (contrariamente all'Insertion Sort e al Merge Sort) e in alcuni casi può avere un'elevata complessità.

L'idea alla base del Quicksort è abbastanza semplice, analogamente al Merge Sort: ad ogni stadio si effettua un ordinamento parziale e poi si procede all'ordinamento sui vari sottoinsiemi. Nel caso particolare si cerca un elemento come perno (o pivot), che è generalmente un elemento di valore intermedio tra il minimo ed il massimo da ordinare, e poi si spostano tutti gli elementi più piccoli di quello alla sua sinistra e più grandi alla sua destra. In questo modo si avranno due insiemi non ordinati, ma con elementi minori e maggiori del perno (che quindi si trova nella sua corretta posizione).

Si applica, quindi, il Quicksort ai due sottoinsiemi (stabilendo nuovi perni) fino a quando l'insieme degli elementi non è correttamente ordinato.

Uno dei principali vantaggi del Quicksort è che risulta essere estremamente semplice sia da capire che da implementare. Non a caso nel tempo sono state cercate diverse ottimizzazioni all'algoritmo che, tuttavia, è praticamente rimasto immutato proprio per la sua efficienza originaria.

Volendo il Quicksort può essere "convertito" in un algoritmo completamente iterativo o misto ricorsivo-iterativo, tuttavia ciò rappresenta soprattutto un buon esercizio di stile, anche se apporta dei vantaggi in termini di risorse occupate.

Implementazione in Java

Scrivere l'algoritmo Quicksort in Java non presenta grosse difficoltà. Si crea, infatti, la classe principale che estende la classe SortAlgorithm (l'algoritmo di ordinamento generico):

```
public class
QuickSortAlgorithm
extends SortAlgorithm
{
```

Il Quicksort prende in input l'array di interi da ordinare, il limite inferiore ed il limite superiore. La prima chiamata viene fatta su tutti gli elementi (lo0 = 0 e hi0 = a.length – 1) dell'array (**Figura 1**).

```
void QuickSort(int
a[], int lo0, int hi0)
throws Exception {
```

Si assegna, quindi, il valore dei parametri alle variabili locali temporanee e si dichiara la variabile *mid*, ovvero il perno.

```
int lo = lo0;
int hi = hi0;
int mid;
```

Viene poi richiamato il metodo di pausa per ridisegnare l'applet

```
pause(lo, hi);
```

per poi procedere con l'esecuzione dell'algoritmo, la cui condizione base è che il margine superiore sia maggiore di quello inferiore.

```
if ( hi0 > lo0) {
```

Si assegna, quindi, un valore "intermedio" al perno *mid* (**Figura 2**)

```
mid = a[ ( lo0 + hi0 ) / 2 ];
```

e si effettua un ciclo fino a quando gli indici superiore ed inferiore non si sovrappongono, ovvero non si è fatto il ciclo su tutti gli elementi a sinistra e a destra del perno.

```
while( lo <= hi ) {
```

FIGURA 1 L'array non ordinato su cui viene applicato il Quicksort.

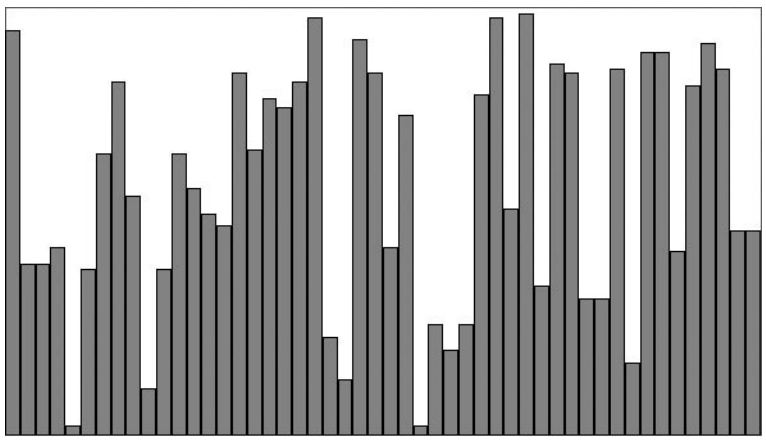
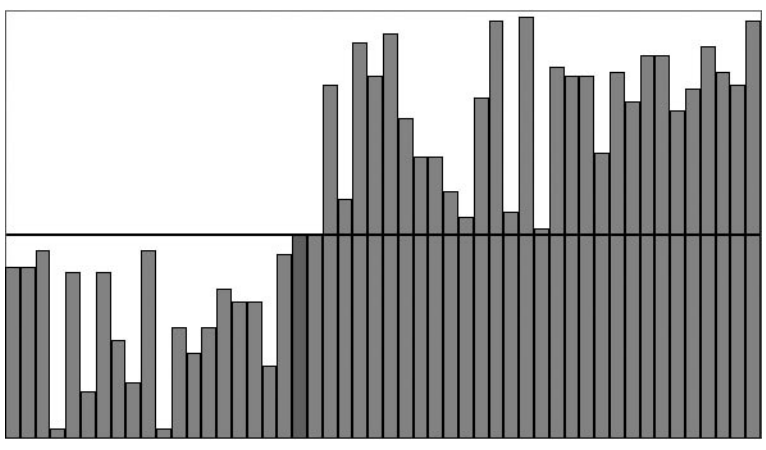


FIGURA 2 Scelto il pivot, si procede con l'ordinamento degli elementi.



Si cerca, poi, il primo elemento che è maggiore od uguale al perno mid partendo dall'indice sinistro,

```
while( ( lo < hi0 ) && ( a[lo] < mid ) )
++lo;
```

e si ripete l'analoga operazione, cercando un elemento minore od uguale al perno, partendo dall'indice destro

```
while( ( hi > lo0 ) && ( a[hi] > mid ) ) --
hi;
```

Se gli indici non si sono sovrapposti, si scambiano i valori degli indici

```
if( lo <= hi ) {
 swap(a, lo, hi);
 pause();
```

Si spostano ulteriormente gli indici e poi si chiude sia l'if che il ciclo while, che si ripete fino a quando la condizione precedente non è verificata.

```
++lo;
--hi;
} }
```

Se l'indice destro originario non ha raggiunto il lato sinistro dell'array, si esegue il Quicksort sulla partizione sinistra

```
if( lo0 < hi ) QuickSort(a, lo0, hi );
```

Analogamente se l'indice sinistro originario non ha raggiunto il lato destroy dell'array, si esegue il Quicksort sulla partizione destra

```
if( lo < hi0 ) QuickSort(a, lo, hi0 );
```

Infine si chiude l'if ed il metodo (**Figura 3**).

```
} }
```

Il metodo di swap è abbastanza semplice e scambia i valori dell'array dei due indici passati.

```
private void swap(int a[], int i, int j) {
 int T;
 T = a[i];
 a[i] = a[j];
 a[j] = T; }
```

Il metodo sort, il costruttore della classe, chiama Quicksort, come già detto, sull'array di interi a, con indici da 0 alla lunghezza (meno uno) dell'array (**Figura 4**).

```
public void sort(int a[]) throws Exception {
 QuickSort(a, 0, a.length - 1); } }
```

L'applet Java

L'applet si compone di tre elementi, di cui due fissi, il file **SortItem.java**, adibito alla creazione dell'applet ed il file **SortAlgorithm.java** che opera sugli oggetti di tipo SortItem per ordinarli; il terzo file può invece cambiare, poiché definisce l'algoritmo specifico da usare per ordinare i numeri (la classe java dell'algoritmo di ordinamento, infatti, generalmente estende la classe SortAlgorithm), ed il suo nome viene passato come parametro quando si richiama l'applet.

In questo modo è abbastanza facile implementare diversi algoritmi di ordinamento con nomi diversi, e poi darli in pasto al codice dell'applet passandogli il nome dell'algoritmo.

FIGURA 3 La chiamata ricorsiva del Quicksort sui sottoinsiemi.

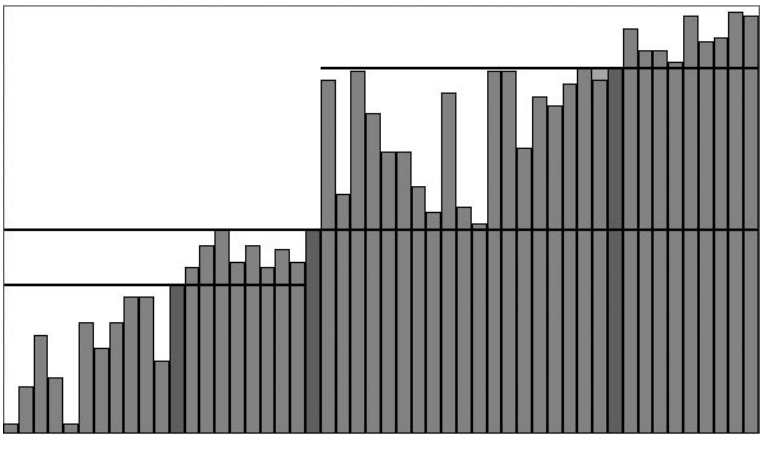
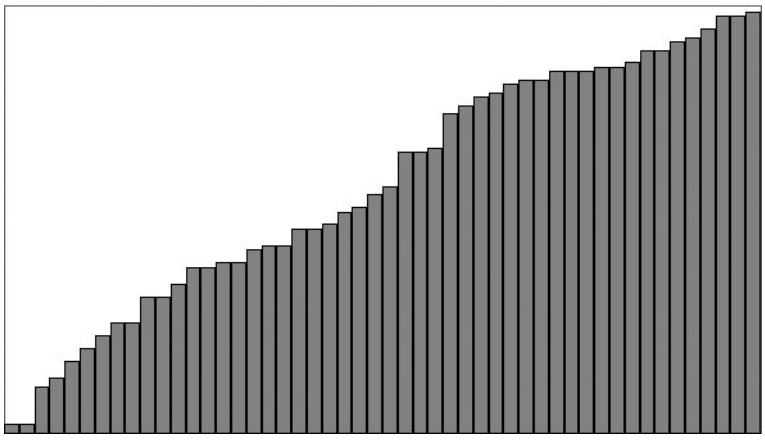


FIGURA 4 Infine l'array è ordinato.



Nell'esempio è stato creato un file di nome **QuickSortAlgorithm.java**, la cui classe estende la classe SoltAlgorithm. Quando poi dovremo richiamare l'applet dalla pagina web, richiameremo il file **SortItem.class** (la versione compilata in bytecode di SortItem.java), dicendogli che l'algoritmo da usare sarà il QuickSort.

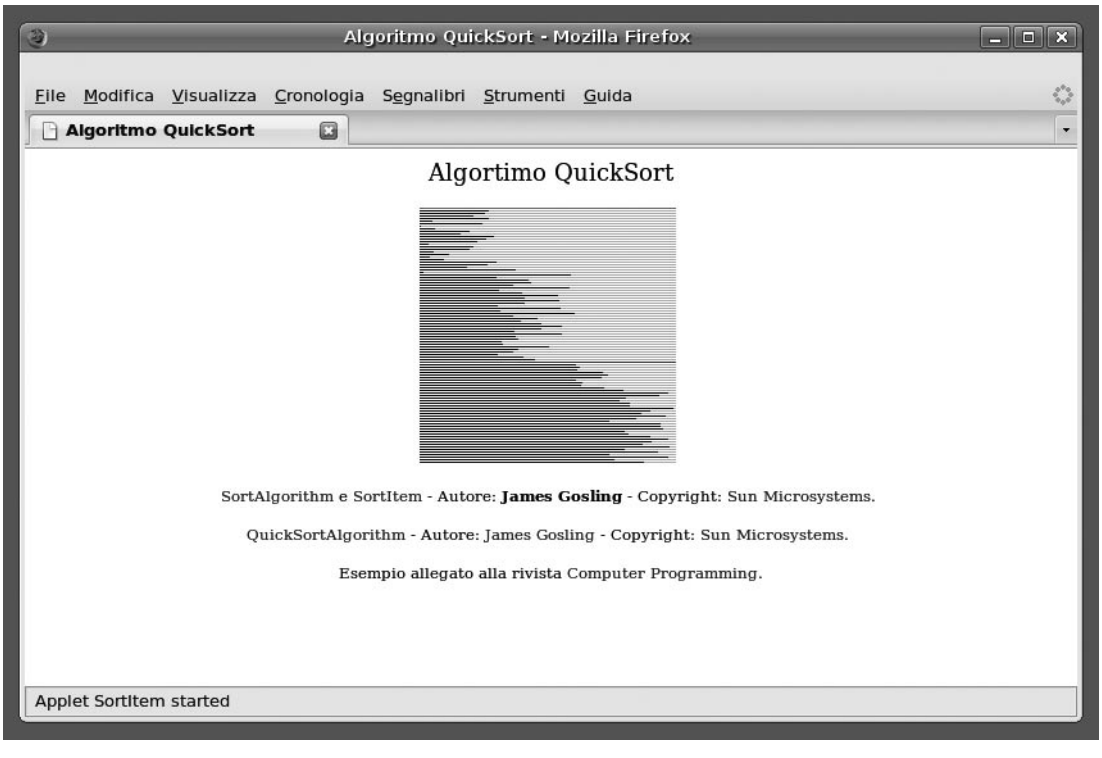
Sul sito ftp.infomedia.it è possibile trovare i sorgenti dell'algoritmo di ordinamento. Per far funzionare correttamente l'applet di ordinamento è necessario compilare i file sorgenti e poi creare una pagina html che permetta il caricamento dell'applet. Una volta scaricato ed installato il JDK è possibile compilare i file Java con un unico comando, dalla cartella che contiene i sorgenti delle tre classi proposte:

```
javac
QuickortAlgorith.java
```

A questo punto è necessario creare una pagina html con il codice per richiamare l'applet in questione; basta aprire un comunissimo editor di testi e creare un file con estensione ".htm" (ad esempio QuickSort.htm), inserendo questo tag all'interno del body:

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

FIGURA 5 L'applet di ordinamento in funzione sotto Linux.



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 e QuickSort (cioè la prima parte del nome della classe) come suo valore.

Per visualizzare l'applet, infine, aprire il file html con il proprio browser internet (**Figura 5**).

Calcolo della complessità

La complessità del Quicksort è nell'ordine di O(n*log*n), così come il suo utilizzo di memoria. Come abbiamo già spiegato, la complessità di un algoritmo è in funzione della dimensione di input dell'algoritmo. Un algoritmo come il Quicksort ha quindi una complessità "poco influenzata" dalla dimensione n dei dati che gli si possono passare.

Ma vediamo come sia possibile risalire alla complessità dell'algoritmo analizzando le varie casistiche:

- CASO PEGGIORE – Nel caso peggiore l'algoritmo ha una complessità di $O(n^2)$; il caso peggiore si ha quando un sottoinsieme ha n-1 elementi e l'altro zero elementi.
  Tale situazione si presenta quando l'array è ordinato in maniera crescente o in maniera decrescente;

- CASO MIGLIORE – Nel caso migliore la chiamata ricorsiva opera su sottoinsiemi, uno di grandezza n/2 e l'altro di grandezza n/2-1, in questo caso $T(n) = 2T(n/2) + O(n)$, la cui complessità risulta essere O(n*log*n);

- CASO MEDIO – La particolarità del Quicksort è che anche con una partizione sfavorevole, esempio 9 a 1, mantiene proprietà interessanti, infatti il tempo impiegato sarebbe dettato dalla funzione $T(n) = T(9n/10) + T(n/10) + cn$.
  In definitiva la ricorsione termina alla profondità di $log_{9/10}n \sim O(log n)$, per cui il Quicksort ha complessità O(n*log*n) e la avrebbe anche se la partizione fosse 99 a 1.

L'applet SortItem, la classe SortAlgorithm e la classe QuickSortAlgorithm 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. Le prime quattro immagini dell'ordinamento sono tratte da Wikipedia inglese e sono sotto pubblico dominio.

RIQUADRO 1 Copyright

Per poter compilare ed eseguire il programma sono necessari tre strumenti, il compilatore Java (per compilare i file sorgenti), le librerie java e il plugin java per il proprio browser (che cambia anche in base al sistema operativo). Fortunatamente per ottenere tutti gli strumenti necessari al corretto funzionamento, sia per Windows che per Linux, è possibile scaricare il JDK (Java Development Kit), il quale contiene al suo interno il JRE (Java Runtime Environment) con i relativi plugin per abilitare la JVM per il proprio browser. Il JDK è scaricabile dal sito http://java.sun.com/javase/downloads/index.jsp.

RIQUADRO 2 Installare Java

L'algoritmo di ordinamento Shell Sort può essere un ottimo sostituto del Quicksort quando si ha bisogno di un algoritmo di più facile implementazione e molto efficiente.
L'idea di base è quella di estendere l'Insertion Sort (trattato nella prima lezione), il quale si comporta generalmente bene su valori già abbastanza ordinati. Per questo si può suddividere un array in porzioni (di volta in volta più piccole) in cui si ordinano i valori di ciascuna porzione. In pratica una volta diviso l'array iniziale in, ad esempio, tre array, si spostano i valori più piccoli nel primo array, e poi quelli più grandi nel secondo e nel terzo.
Aumentando il numero delle partizioni e ricomponendo l'array iniziale i dati sono praticamente già quasi ordinati. Basta applicare, a questo punto, l'Insertion Sort vero e proprio. Per approfondimenti visitare http://it.wikipedia.org/wiki/Shell_sort.

RIQUADRO 3 Lo Shell Sort

Conclusioni

Con questa ultima puntata abbiamo concluso la trattazione degli algoritmi di ordinamento in Java.

Ripercorrendo le scorse puntate si può capire come il Quicksort sia in parte l'algoritmo "conclusivo", ovvero il miglior compromesso per scopi general purpose.

Si spera per il lettore che lo studio della complessità degli algoritmi sia ora una materia meno oscura e che ciò lo porti ad applicare le conoscenze nel proprio studio e nel proprio lavoro.

Se si vuole approfondire l'argomento si può fare ricorso a Wikipedia che, sia in italiano che in inglese, tratta molto bene gli algoritmi di ordinamento, il calcolo della complessità e le varie implementazioni degli algoritmi.

L'autore rimane a disposizione per ulteriori chiarimenti, consigli e precisazioni al proprio indirizzo email.

CODICE ALLEGATO

ftp.infomedia.it  **Ordinamento4**

Implementazione dell'algoritmo Quicksort