Test sull'ordinamento: Insertion Sort e Merge Sort

di Pagliocca Marco

1 Il problema dell'ordinamento

Data una sequenza di numeri "a_0,a_1, ..., a_N", si vuole definire una sua permutazione "a'_0,a'_1, ..., a'_N" tale che $a'_0 \leq a'_1 \leq \ldots \leq a'_N$.

1.1 Insertion Sort

Insertion Sort é un algoritmo in grado di risolvere il problema dell'ordinamento ed é particolamente efficacie quando gli elementi da ordinare sono pochi.

```
INSERTION-SORT(A)
1- for j <--- 2 to A.length
2-     key = A[j]
3-     i <--- j-1
4-     while j>0 and key > A[j]
5-         A[j+1] <--- A[j]
6-         j <--- j+1
7-     A[j+1] <--- key
```

Ha un approccio **incrementale**: in ogni momento inserisce il nuovo elemento nella posizione giusta. Tale proprietá viene espressa in modo formale dalla sua **invariante di ciclo**:

- All'inizio di ogni iterazione del ciclo *for*(righe 1-7), il sottoarray A[1 ... j-1] é ordinato ed é formato dagli stessi elementi che originariamente erano in A[1 ... j-1].

Prestazioni: il tempo di esecuzione dell'algoritmo dipende sia dalla lunghezza che dalla bontá del vettore da ordinare, quest'ultima intesa come quanto esso sia giá parzialmente ordinato.
In effetti, il **caso migliore** si ha quando é completamente ordinato: non viene mai eseguito il corpo del ciclo *while* e si ottiene un tempo di esecuzione lineare.
$\Rightarrow$ T(N)= Θ(N)
Invece, il **caso peggiore** si ha quando il vettore é ordinato al contrario: il ciclo *while* viene eseguito j-1 volte ad ogni j-esima iterazione del ciclo *for*,

determinando un tempo di esecuzione quadratico.
$\Rightarrow$ T(N)= $\Theta(N^2)$
Lo studio del caso peggiore pone un limite superiore per ogni input, dunque rappresenta una buona stima sul tasso di crescita dell'algoritmo.

Descrizione degli esperimenti: le considerazioni sulle prestazioni ci sono utili per definire su quali tipi di vettore concentrare i test. Si utilizza:

- un vettore ordinato:
  realizzato semplicemente utilizzando i primi N numeri naturali, corrisponde al miglior caso di ordinamento con Insertion Sort.
- un vettore ordinato al contrario:
  realizzato utilizzando i primi N numeri naturali disposti in ordine decrescente, corrisponde al caso peggiore di ordinamento con Insertion Sort.
- un vettore di valori casuali:
  realizzato utilizzando i primi N numeri naturali disposti in modo casuale, corrisponde ad un generico caso.

I test sono eseguiti su dati di dimensione crescente (vettore con 10,1000,1000,... dati), arrestandosi quando il tempo di esecuzione dell'algoritmo, per ogni istanza, é maggiore di 25 secondi.
Il **tempo di esecuzione di Insertion Sort**, invece, viene misurato come differenza tra il tempo prima e dopo la chiamata della funzione, quando il vettore é giá stato costruito, in modo da evitare tempi aggiuntivi. Tuttavia all'interno dell'algoritmo viene periodicamente verificato che il tempo di esecuzione non sia superiore al *limite* imposto, in modo tale da bloccare un ordinamento troppo lento. Questa aggiunta ha conseguenze sulle prestazioni, ma nel complesso non dovrebbe disturbare il rapporto previsto tra N e T(N). Inoltre **i test verranno eseguiti 4 volte per ogni tipo di vettore**, in modo da evitare casi particolari, relativi alla struttura hardware della macchina elaboratrice, e prendere un valore medio piú affidabile.

Specifiche del calcolatore: gli esperimenti vengono eseguiti su un *ASUS Laptop X541UV* a 8GB di memoria RAM e con processore Intel Core i7-6500U a 2.50GHz di frequenza.
L'architettura a 64-bit ospita come Sistema Operativo Windows 10 Home.

1.1.1 Risultati degli esperimenti

Per una visione complessiva dei dati degli esperimenti guardare il file allegato *SelectionSortDatiEsperimenti*. La seguente tabella indica solo i valori

dei tempi medi registrati.

Dimensione	10	100	1000	10000	100000	1000000	10000000
Caso Migliore	0.00003002	0.0002208	0.002248	0.01926	0.1783	1.636	15.84
Caso Peggiore	0.00003328	0.0007795	0.07822	7.455	-	-	-
Casuale	0.00002685	0.0005685	0.03962	3.888	-	-	-

1.1.2 Osservazioni

Dai dati forniti dall'esecuzione dell'algoritmo si possono notare alcune previsioni teoriche precedentemente esposte: in 25 secondi Selection Sort é in grado di completare l'ordinamento su un vettore di 10^7 elementi, ma non appena viene modificato l'ordine si ferma molto prima.
Questo é da imputare alla forte connessione tra tempi di esecuzione e la dimensione dei dati, ma anche alla relazione con i tipi di istanza, dove nel caso migliore é lineare, mentre nel caso peggiore é quadratica.
Si osservi, inoltre, come per un vettore casuale, sebbene tenda ad assumere una relazione quadratica, questa si fa 'sentire' asintoticamente: per vettori di 10 elementi i tempi di esecuzione sono addirittura migliori del caso migliore, ma questi vanno a peggiorare giá per 100 elementi, fino a non rendere possibile l'ordinamento nemmeno per 10^5 elementi. Tuttavia i tempi rimangono sempre piú bassi del caso peggiore.
Infine vale la pena notare la dipendenza lineare nel vettore giá ordinato. All'aumentare della dimensione (x10) si ottengono tempi di esecuzioni altrettanto aumentati (circa x10), mettendo in luce l'ottimo utilizzo di Selection Sort in certe casistiche.

1.2 Merge Sort

Mentre Insertion Sort é un algoritmo *iterativo*, Merge Sort fa parte degli algoritmi *ricorsivi*. Di fatto Merge Sort ordina l'intero vettore inoltrano a chiamate di se stesso la risoluzione di sotto-problemi di dimensione decrescente:

Divide il vettore A[p ... r] in A[p ... q] e A[q+1 ... r], dove q é il punto mediano del vettore;

Impera ordinando ricorsivamente A[p ... q] e A[q+1 ... r];

Combina i due sotto-vettori in un singolo vettore ordinato.

```
MERGE-SORT(A, p, r)
1- if(p < r)
2-     q <--- (p+r)/2
3-     MERGE-SORT(A, p, q)
4-     MERGE-SORT(A, q+1, r)
5-     MERGE(A, p, q, r)
```

La procedura Merge si occupa della fusione dei sotto-vettori. Inizia ad operare quando la suddivisione arriva al caso base, ovvero sotto-vettori di dimensione unitaria. Da lí in poi confronta i primi, e non ancora copiati, elementi di ciascuna lista e inserisce quello piú piccolo nella prima locazione vuota del vettore-fusione.
Poiché sappiamo in anticipo che ci saranno r-p+1 elementi da ordinare, possiamo terminare il processo non appena si raggiunge questo numero di passi:

```
MERGE(A, p, q, r)
1-  n1 <--- q-p+1
2-  n2 <--- r-q
3-  #crea gli array L[1 ... n1+1] e R[1 ... n2+1]
4-  for i <--- 1 to n1
5-      L[i] <--- A[p+i-1]
6-  for j <--- 1 to n2
7-      R[j] <--- A[q+j]
8-  L[n1+1] <---
9-  R[n2+1] <---
10- i <--- 1
11- j <--- 1
12- for k <--- p to r
13-   if(L[i] <= R[j])
14-   A[k] <--- L[i]
15-   i <--- i+1
16-   else
17-   A[k] <--- R[j]
18-   j <--- j+1
```

Prestazioni: dalla descrizione di Merge Sort si puó notare come esso non dipenda in modo stretto dai dati da ordinare o dal loro ordine.
Non é una coincidenza che il tempo di esecuzione sia piú o meno lo stesso per ogni istanza, ovvero non vi é né un caso migliore, né un caso peggiore.
$\Rightarrow$ T(N)= Θ(NlgN)

Descrizione degli esperimenti: dalle considerazioni sulle prestazioni viene fuori che i test su Merge Sort possono essere eseguiti unicamente su vettori di valori casuali, ottenuti allo stesso modo di Selection Sort.
Tuttavia verranno presi in considerazione anche test su vettori ordinati e su vettori ordinati al contrario. In questo modo potremmo:

- confrontare i risultati di Selection e Merge Sort;
- verificare che effettivamente non vi siano particolari casi di Merge Sort.

I test sono eseguiti su dati di dimensione crescente (vettore con 10,1000,1000,... dati), arrestandosi quando il tempo di esecuzione dell'algoritmo, per ogni istanza, é maggiore di 25 secondi.
Il **tempo di esecuzione di Merge Sort**, invece, viene misurato come differenza tra il tempo prima e dopo la chiamata della funzione, quando il vettore é giá stato costruito, in modo da evitare tempi aggiuntivi. Tuttavia all'interno dell'algoritmo viene verificato che il tempo di esecuzione non sia superiore al *limite* imposto quando arriva al primo caso base e prima di ogni suddivisione del sottovettore dx. Di conseguenza viene perso molto tempo per verificare la condizione di uscita, ma ci garantisce che l'esecuzione termini non appena (o poco dopo) venga raggiunto il limite. Inoltre **i test verranno eseguiti 4 volte per ogni tipo di vettore**, in modo da evitare casi particolari, relativi alla struttura hardware della macchina elaboratrice, e prendere un valore medio piú affidabile.

1.2.1 Risultati degli esperimenti

Per una visione complessiva dei dati degli esperimenti guardare il file allegato *MergeSortDatiEsperimenti*. La seguente tabella indica solo i valori dei tempi medi registrati.

Dimensione	**10**	**100**	**1000**	**10000**	**100000**	**1000000**
Ordinato	0.00006835	0.000608	0.007078	0.07762	0.8555	9.868
Al contrario	0.0000726	0.000701	0.007755	0.07515	0.830	9.16
Casuale	0.00006688	0.000647	0.007705	0.0871	0.988	11.65

1.2.2 Osservazioni

Rispetto a Selection Sort, Merge Sort é un algoritmo piú affidabile: anziché raggiungere un 'picco risolutivo', Merge Sort mantiene costante la dimensione dei dati con cui riesce ad effettuare l'ordinamento, cosí come lo sono anche i tempi nei tre casi implementati.
Questi ultimi mostrano come il tempo migliore nei tre tipi di vettore é alternante, verificando la tesi teorica che non avesse un caso migliore e/o peggiore.
Tuttavia, da quanto si nota, sembrerebbe che asintoticamente un vettore ordinato si comporta come un caso medio, un vettore ordinato al contrario presenta i migliori tempi, mentre quelli peggiori sono ottenuti ordinando un vettore di valori casuali. D'altronde, peró, potrebbe essere dovuto al caso e/o al calcolatore utilizzato.
L'asintoticitá a NlgN si manifesta soprattutto nell'ordinamento di 10^6 elementi, mentre rimane quasi silente per dimensioni piú piccole, a causa del 'debole' carattere del logaritmo.

1.3 Note sulle scelte implementative:

- La verifica della condizione di uscita viene eseguita molte volte al fine di evitare una situazione di non sopportabilitá da parte del calcolatore in questione. In particolar modo, é capitato che arrivasse ad uno stato di blocco completo del sistema.
- Per evitare tempi di esecuzione troppo lunghi é stato scelto il valore limite di 25s per istanza, in quanto prove su singole istanze hanno constatato che, fino ad oltre 120 secondi, il numero di dati ordinati rimane nell'ordine di quelli ordinati in 25 secondi.
- Il numero di test eseguiti per tipologia di vettore é 4, per avere sufficienti valori su cui calcolare la media senza appesantire troppo l'esecuzione.
- La precisione dei valori decimali in Phyton é stata impostata a 4 cifre significative, sufficienti per effettuare un'analisi completa.