Programmazione

Dipartimento di Matematica

Ing. Cristiano Gregnanin

Corso di laurea in Matematica

11 maggio 2015

Algoritmi di ordinamento

Panoramica

Metodo per ordinare un insieme di oggetti.
Il problema dell'ordinamento è un problema classico dell'informatica:

- valenza didattica: problema in se molto semplice e chiunque è in grado di comprenderne i termini generali
- valenza in ambito applicativo: si ritrova spesso come sotto problema in molti **casi reali**

Sono stati proposti molti algoritmi diversi ed eleganti:
se presentati in opportuna successione permettono di evidenziare gli aspetti fondamentali della progettazione e della costruzione di un algoritmo efficiente

Panoramica

Il problema dell'ordinamento (sorting) può essere posto nei seguenti termini:
Dati un insieme di elementi qualsiasi $A = a_1, a_2, \therefore, a_n$, su cui sia possibile definire una relazione d'ordine totale $\leq$ si richiede di produrre una permutazione degli elementi in modo che
$a_{i_h} \leq a_{i_k}$ $\forall h, k = 1, 2, \therefore, n$ e $h \leq k$
Una relazione d'ordine totale è una relazione riflessiva, antisimmetrica e transitiva definita su ogni coppia di elementi dell'insieme *quindi* **la soluzione è unica a meno di elementi uguali**

Panoramica

Soluzioni via via sempre più sofisticate ma con complessità computazionale sempre più bassa

Algoritmo	**Caso migliore**	**Caso medio**	**Caso peggiore**
Selection Sort	n^2		n^2
Insertion Sort	n		n^2
Bubble Sort	n		n^2
Quick Sort	$n \log_2 n$	$n \log_2 n$	n^2
Merge Sort	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$
Heap Sort	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$
Counting Sort	n	n	n
Bucket Sort			n

Panoramica

selection sort, insertion sort, bubble sort sono estremamente elementari e consentono un'implementazione semplice:
il costo da pagare alla semplicità di questi algoritmi sta nell'elevata complessità computazionale, che in questi casi è $O(n^2)$.
Quicksort consente di raggiungere una complessità di $O(n \log_2 n)$ nel caso medio, mentre nel caso più sfavorevole ritorna ad una complessità di $O(n^2)$.

Panoramica

Mergesort e **Heapsort** consentono di raggiungere una complessità di $O(n \log_2 n)$ anche nel caso peggiore:
è possibile dimostrare che il limite inferiore alla complessità computazionale del problema dell'ordinamento mediante confronti (senza dunque poter sfruttare altre informazioni sull'insieme da ordinare) è proprio pari a $nlog_2 n$

Panoramica

counting sort e **bucket sort** sono invece basati su altri criteri e strategie, diverse dal confronto fra i valori degli elementi dell'insieme da ordinare, e sfruttano pertanto altre informazioni sui dati in input:
grazie a questo consentono di ridurre ulteriormente la complessità nel caso peggiore, arrivando ad una **complessità lineare** di (n).
Sotto a tale soglia è impossibile scendere, dal momento che per ordinare un insieme di **n** elementi è necessario esaminarli tutti almeno una volta.
Ci si concentra nello studio della complessità del caso peggiore, dal momento che è il parametro più conservativo per la valutazione dell'efficienza di un algoritmo.

Selection Sort

è utile quando *l'insieme da ordinare è composto da pochi elementi* e dunque anche un algoritmo non molto efficiente può essere utilizzato con il vantaggio di non rendere troppo sofisticata la codifica del programma che lo implementa.
Si ripete per n volte una procedura in grado di selezionare alla *i-esima* iterazione l'elemento più piccolo nell'insieme e di scambiarlo con l'elemento che in quel momento occupa la posizione *i*

Selection Sort

- Alla prima iterazione dell'algoritmo verrà selezionato l'elemento più piccolo dell'intero insieme e sarà scambiato con quello che occupa la prima posizione;
- Alla seconda iterazione è selezionato il secondo elemento più piccolo dell'insieme, ossia l'elemento più piccolo dell'insieme **ridotto** costituito dagli elementi $a_2, a_3, \dots, a_n$ ed è scambiato con l'elemento che occupa la seconda posizione;
- Si ripete fino ad aver collocato nella posizione corretta tutti gli n elementi.

Selection Sort

Il numero di operazioni è:
$(n-1)+(n-2)+\therefore +2+1 \Rightarrow n(n-1)/2 \Rightarrow O(n^2)$
Un inconveniente dell'algoritmo di ordinamento per selezione è che il tempo di esecuzione dipende solo in modo modesto dal grado di ordinamento in cui si trova il file.
La ricerca del minimo elemento durante una scansione del file non fornisce informazioni circa la posizione del prossimo minimo nella scansione successiva.

Selection Sort

Nonostante l'approccio brutale adottato, ordinamento per selezione ha un'importante applicazione:
poiché *ciascun elemento viene spostato al più una volta*, questo tipo di ordinamento è il metodo da preferire quando *si devono ordinare file costituiti da record estremamente grandi e da chiavi molto piccole*.
Per queste applicazioni il costo dello spostamento dei dati è prevalente sul costo dei confronti e nessun algoritmo è in grado di ordinare un file con spostamenti di dati sostanzialmente inferiori a quelli dell'ordinamento per selezione.

Insertion Sort

è un algoritmo relativamente semplice per ordinare un array.
Non è molto diverso dal modo in cui un essere umano ordina un mazzo di carte.
Algoritmo in place, cioè ordina l'array senza doverne creare una copia, risparmiando memoria.

Insertion Sort

Ad ogni istante (*iterazione*), l'array è costituito da una parte iniziale ordinata (che aumenta di volta in volta) e da la parte rimanente che contiene i valori da ordinare.
Per ogni valore ancora da inserire, viene fatta una ricerca binaria nella parte ordinata del vettore e vengono spostati in avanti tutti gli elementi per liberare la posizione.
Nella posizione liberata viene inserito il valore.

Insertion Sort

Complessità:

- Caso peggiore $O(n^2)$
- Caso migliore $O(n) \Rightarrow$ vettore già ordinato

Molto vantaggioso per vettori quasi ordinati

Bubble Sort

Fa emergere pian piano gli elementi più piccoli verso l'inizio dell'insieme da ordinare facendo sprofondare gli elementi maggiori verso il fondo:
La strategia adottata è quella di **scorrere più volte** la sequenza da ordinare, verificando ad ogni passo l'ordinamento reciproco degli elementi contigui, a_i e a_{i+1}, ed eventualmente **scambiando la posizione di quelle coppie di elementi non ordinate.**

Bubble Sort

Procedendo dall'inizio fino alla fine della sequenza, al termine di ogni scansione si è ottenuto che l'elemento massimo è finito in fondo alla sequenza stessa, mentre gli elementi più piccoli hanno cominciato a spostarsi verso l'inizio della sequenza.
Dunque alla fine della prima scansione possiamo essere certi che l'elemento massimo ha raggiunto la sua posizione corretta nell'ultima posizione della sequenza quindi la scansione successiva potrà fermarsi senza considerare l'ultimo elemento dell'insieme e riuscendo così a collocare nella posizione corretta (la penultima) il secondo elemento più grande dell'insieme;
Si ripete fino ad aver completato l'ordinamento dell'intera sequenza.

Bubble Sort

Il numero di operazioni è:
$(n-1)+(n-2)+\therefore+2+1 \Rightarrow n(n-1)/2 \Rightarrow O(n^2)$

Quicksort Sort

Ordinamento ricorsivo in place che si basa sul paradigma *divide et impera*. La base del suo funzionamento è l'**utilizzo ricorsivo della procedura partition**:

- preso un elemento da un array si pongono gli elementi minori a sinistra rispetto a questo e gli elementi maggiori a destra
- La stessa procedura poi è richiamata in modo ricorsivo sui due sotto insiemi

Ha, in generale, prestazioni migliori tra quelli basati su confronto.

Quicksort Sort

L'idea base può esprimersi agevolmente in **termini ricorsivi**. Ad ogni stadio si effettua un ordinamento parziale di una sequenza di oggetti da ordinare. Assunto un elemento come perno dello stadio, si confrontano con esso gli altri elementi e si posizionano alla sua sinistra i minori e a destra i maggiori, senza tener conto del loro ordine.
Dopo questo stadio si ha che il perno è nella sua posizione definitiva.
Successivamente si procede in modo ricorsivo all'ordinamento parziale delle sotto-sequenze di elementi rimasti non ordinati, fino al loro esaurimento.

Quicksort Sort

Caso peggiore array ordinato in modo inverso: $O(n^2)$
Caso migliore/medio: $O(n \log_2 n)$

Merge Sort

Utilizza un processo di risoluzione **ricorsivo**.
L'idea alla base del merge sort è il procedimento *Divide et Impera*, che consiste nella suddivisione del problema in sotto-problemi via via più piccoli.
Il merge sort opera quindi dividendo l'insieme da ordinare in due metà e procedendo all'ordinamento delle medesime ricorsivamente.
Quando si sono divise tutte le metà si procede alla loro fusione (merge appunto) costruendo un insieme ordinato.

Merge Sort

- **Fase 1:** *divide* L'insieme di elementi viene diviso in 2 metà. Se l'insieme è composto da un numero dispari di elementi, viene diviso in 2 sottogruppi dei quali il primo ha un elemento in meno del secondo.
- **Fase 2:** *impera* Supponendo di avere due sequenze già ordinate. Per unirle, l'algoritmo mergesort estrae ripetutamente il minimo delle due sequenze in ingresso e lo pone in una sequenza in uscita.

Merge Sort

Complessità:
Caso **peggiore** $\Rightarrow O(n \log n)$