**CAROSELLI STEFANO**

# INFORMATICA GENERALE

## TESINA

### « LA TECNICA DEL "*DIVIDE-ET-IMPERA*" E DUE SUE IMPORTANTI APPLICAZIONI: IL *QUICK SORT* E IL *MERGE SORT* »

La tecnica del DIVIDE-ET-IMPERA è un metodo di risoluzione per problemi tramite algoritmi che presentano una struttura ricorsiva : tale metodo consiste nello scomporre il problema di partenza in sottoproblemi simili a quello originale, ma di dimensione inferiore, indipendenti l'uno dall'altro e risolvibili con procedimenti elementari; risolti i sottoproblemi ricorsivamente si chiude la ricorsione combinando insieme le soluzioni parziali, creando così una soluzione del problema di partenza.

In ogni ciclo di ricorsione la tecnica del divide-et-impera è caratterizzata dai seguenti tre passi fondamentali:

- **Divide**. Si suddivide il problema in un certo numero di sottoproblemi, simili a quello d'origine.
- **Impera**. Tramite ricorsione si arriva a risolvere problemi elementari, in modo diretto.
- **Combina**. Le soluzioni ottenute sono combinate tra loro per ottenere la soluzione del problema di partenza.

Una ricorrenza per il tempo di esecuzione di un algoritmo divide-et-impera è basata sui tre passi base : sia $T(n)$ il tempo di esecuzione di un problema di dimensione $n$; se $n \le c$ per qualche costante $c$ allora anche il tempo è costante, e di conseguenza : $T(n) = \Theta(1)$. Nel caso generale il problema si divide in $\alpha$ sottoproblemi, ciascuno di dimensione $n / \beta$ ; sia $D(n)$ il tempo relativo alla scomposizione, e $C(n)$ quello relativo alla combinazione delle soluzioni; allora:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \le c, \\ \alpha\, T(n/\beta) + D(n) + C(n) & \text{altrimenti.} \end{cases}$$

Il Quick Sort e il Merge Sort sono due algoritmi di ordinamento che utilizzano la tecnica del divide-et-impera: si considera un vettore di variabili ordinabili (ad es. di interi); si divide il vettore in sottovettori con un procedimento ricorsivo fino ad arrivare a vettori semplici, su cui è possibile fare un ordinamento immediato e infine si richiude la ricorsione.

## IL QUICK SORT

Il Quick Sort (in inglese: *ordinamento rapido*) è un algoritmo molto usato nella pratica perché in media è notevolmente efficiente.

È dato il vettore $v$ di $n$ elementi; in particolare supponiamo che gli elementi (le chiavi) siano valori interi; il primo passo consiste nel confrontare la prima chiave del vettore con le rimanenti, cercando di separare quelle maggiori da quelle minori ($n - 1$ confronti) e ponendo la chiave considerata come elemento di separazione.

Quindi al termine del primo ciclo la prima chiave è nella sua posizione definitiva, e il problema passa ora allo studio dei due sottovettori che si sono generati.

Si continua questa divisione fino ad arrivare a sottovettori di lunghezza $\leq 2$, il cui ordinamento è immediato; una volta ordinati questi vettori si ritorna al vettore di partenza, il quale risulta essere completamente ordinato.

Il processo divide-et-impera del Quick Sort consiste quindi nei seguenti passi :

- **Divide**. Il vettore $v[i..k]$ è riordinato in due sottovettori non vuoti $v[i..j]$ e $v[j..k]$ in modo tale che ogni elemento del primo sia minore (al più uguale) di ogni elemento del secondo. Viene quindi fissato l'indice $j$ nella sua posizione definitiva.
- **Impera**. I due sottovettori sono ordinati con chiamate ricorsive alla procedura Quick Sort.
- **Combina**. Poiché ogni divisione fissa un indice di divisione, il vettore si ordina automaticamente, pertanto non è necessario ricombinare le soluzioni.

La chiave di questo algoritmo è il riordinamento nel passo "Divide": il vettore è partizionato nei due sottovettori; per tale motivo è necessaria una procedura ausiliaria, *Ordina*$(m,n,j)$ che per mezzo di ripetuti scambi crea due regioni nel vettore dipendenti da una determinata chiave : la prima regione contiene solo elementi minori della chiave considerata, la seconda solo elementi maggiori; la procedura termina quando tutti gli elementi sono stati collocati nella propria regione.

La procedura *QuickSort* ($m$, $n$) esamina il vettore nell'intervallo da $m$ a $n$, lo ripartisce usando *Ordina*, e passa ad esaminare i due sottovettori; quando i sottovettori sono tutti di dimensioni unitarie, il vettore risulta ordinato.

Ad esempio consideriamo il seguente vettore:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 26 | 5 | 37 | 1 | 61 | 11 | 59 | 15 | 48 | 19 |

Confrontiamo la chiave 26 con le rimanenti e dopo $\sim n$ confronti otteniamo il nuovo vettore:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 11 | 5 | 19 | 1 | 15 | **26** | 59 | 61 | 48 | 37 |

Continuando nella risoluzione del problema, consideriamo le chiavi 11 e 59, confrontandole con i restanti valori dei rispettivi sottovettori; dopo $\sim n/2 + n/2 = n$ confronti si ottiene:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | **11** | 19 | 15 | **26** | 48 | 37 | **59** | 61 |

Al passo successivo il vettore è ordinato:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 11 | 19 | 15 | 26 | 37 | 49 | 59 | 61 |

Quindi ogni ciclo necessita di $O(n)$ confronti, e nel caso medio servono $k$ cicli, dove $2^k = n$; in media servono quindi $n \cdot lg_2(n)$ confronti per ordinare un vettore.

Tuttavia nel caso di un vettore già ordinato ogni ciclo di $n$ confronti fissa una sola chiave (l'i° ciclo fissa l'i° elemento): l'algoritmo degenera in un Selection Sort, ed occorrono quindi $n - 1$ cicli per terminare la procedura.

*In definitiva nel caso peggiore il Quick Sort ha un costo di $O(n^2)$.*

Malgrado questo tempo di esecuzione lento nel caso peggiore, è un algoritmo molto comodo perché il caso peggiore si verifica con scarse probabilità, e nel caso medio il suo tempo di esecuzione è $\Theta(n \cdot lg_2(n))$: un costo notevolmente basso rispetto a $O(n^2)$.

È riportato ora un algoritmo di Quick Sort, utilizzante un vettore v. Vengono richiamate la procedura elementare *Scambia*, e la procedura *Ordina*, base dell'algoritmo, che vengono descritte qui di seguito; la procedura *Ordina* è ricorsiva e agisce sul vettore v, da un indice x ad un indice y.

```
Procedure Scambia(var a,b:integer);
var c:integer;
  Begin
    c:=a; a:=b; b:=c;
  End;
```

```
Procedure Ordina(var v:vet; x,y:integer);
var i,j:integer;
  Begin
    i:=x; j:=y+1;
    while (i<j) do
      begin
        repeat j:=j-1 until (v[j]<=v[x]);
        repeat i:=i+1 until (v[i]>=v[x]);
        if (i<j) then scambia (v[i],v[j]);
      end;
    if x<>j then scambia (v[x],v[j]);
    if (m<j-1)  then quicksort(v,m,j-1);
    if (n>j+1)  then quicksort(v,j+1,n);
  End;
```

```
Procedure QuickSort(var v:vet; n:integer);
  Begin
    Ordina(v,1,n);
  End;
```

## IL MERGE SORT

Il Merge Sort (in inglese: merge = *fusione*) è un algoritmo è un semplice algoritmo di ordinamento che ha il vantaggio di avere un basso costo anche nel caso peggiore.

È dato un vettore $w$ di lunghezza $n$, dove pre semplicità supponiamo $n = 2^h$ per qualche $h \in \mathbf{N}$; per prima cosa il vettore $w$ viene diviso in $n / 2 = 2^{h-1}$ sottovettori di lunghezza 2: tali coppie di elementi sono facilmente ordinabili con una eventuale procedura di "scambio". Successivamente si fondono coppie di sottovettori tramite una procedura di "fusione" tra vettori ordinati, passando da $2^{h-1}$ a $2^{h-2}$ sottovettori ordinati.

Ripetendo quest'operazione si ottiene dopo più fusioni un nuovo vettore di lunghezza $n$, questa volta ordinato.

Il processo divide-et-impera del Merge Sort consiste quindi nei seguenti passi:

- **Divide**. Il vettore di dimensione $m$ viene diviso in due sottovettori, ciascuno di lunghezza $m / 2$.
- **Impera**. Usando ricorsivamente il Merge Sort vengono ordinati i due sottovettori.
- **Combina**. Fonde le due sottosequenze ordinate, in un'unica sequenza ordinata.

L'operazione chiave dell'algoritmo è la fusione dei due sottovettori ordinati, presente nel passo "combina" del divide-et-impera. Per eseguire una tale fusione si usa una procedura ausiliaria *Fusion(A, i, j)* dove $A$ è un array, $i$ e $j$ sono indici di elementi di $A$, con $i < j$; la procedura, aiutandosi con un vettore d'appoggio uguale ad a, considera i due sottovettori di $A$ : da $i$ a $\frac{i+j}{2}$ e da $\frac{i+j}{2}+1$ a $j$, che si suppone siano già stati ordinati, e li riordina in modo tale che il vettore da $i$ a $j$ sia ordinato; il passo iniziale di questo procedimento è il considerare sottovettori unitari ($j = i$), che sono banalmente ordinati; il passo successivo è quindi ordinare coppie contigue di elementi, con un eventuale semplice scambio.

La procedura *MergeSort(v, n, m)* esamina il vettore $v$ dall'indice $m$ all'indice $n$: se il vettore è una coppia di elementi procede al suo ordinamento in modo diretto (con uno scambio) altrimenti passa ad esaminare i due sottovettori e, dopo averli ordinati, li fonde in un unico vettore ordinato con la procedura *Fusion*.

Ad esempio consideriamo il seguente vettore di lunghezza $16 = 2^4$ :

$w =$

| 2 | 6 | 21 | 8 | 15 | 32 | 28 | 7 | 9 | 11 | 17 | 5 | 16 | 13 | 24 | 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Costruiamo una tabella, dove nella $j$-ma riga sono rappresentati i $2^{4-j}$ sottovettori ordinati di lunghezza $2^j$ in modo tale che nella prima riga ci sia il vettore $w$ di partenza, e nell'ultima riga risulti il vettore $w$ ordinato:

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | (2) | (6) | (21) | (8) | (15) | (32) | (28) | (7) | (9) | (11) | (17) | (5) | (16) | (13) | (24) | (25) |
| 1 | (2 | 6) | (8 | 21) | (15 | 32) | (7 | 28) | (9 | 11) | (5 | 17) | (13 | 16) | (24 | 25) |
| 2 | (2 | 6 | 8 | 21) | (7 | 15 | 28 | 32) | (5 | 9 | 11 | 17) | (13 | 16 | 24 | 25) |
| 3 | (2 | 6 | 7 | 8 | 15 | 21 | 28 | 32) | (5 | 9 | 11 | 13 | 16 | 17 | 24 | 25) |
| 4 | (2 | 5 | 6 | 7 | 8 | 9 | 11 | 13 | 15 | 16 | 17 | 21 | 24 | 25 | 28 | 32) |

Ad ogni ciclo si applica la procedura *Fusion* che costa sempre di $n/2$ scambi per fondere i sottovettori ordinati; il numero totale di cicli è $h = lg_2(n)$, indipendentemente dall'ordinamento di partenza del vettore.

*Il Merge Sort ha anche nel caso peggiore un costo di $O(n \cdot lg_2(n))$.*

È riportato di seguito un algoritmo di Merge Sort, utilizzante un vettore *x* e due interi *x*1 e *x*2, che definiscono l'intervallo di chiavi (e quindi il sottovettore) in esame; verrà descritta la procedura *Fusion*, ma non la procedura elementare *Scambia*.

```
Procedure Mergesort(var x:vet; x1,x2:integer);
var xm:integer;
Begin
  if (x2-x1=1)    then  begin if (x[x1]>x[x2]) then scambia(x[x1],x[x2]); end
                  else  begin
                          xm:=(x1+x2) div 2;
                          mergesort(x,x1,xm);
                          mergesort(x,xm+1,x2);
                          fusion(x,x1,xm,x2);
                        end;
End;
```

```
Procedure Fusion(var v:vet; v1,vm,v2:integer);
var   w:vet;
      i,j,k:integer;
Begin
  for i:=v1 to v2 do w[i]:=v[i];
  i:=v1; j:=vm+1;  k:=v1;
  While (i<=vm) and (j<=v2) and (k<=v2) do
    begin
      if w[i]<w[j] then begin v[k]:=w[i]; i:=i+1 end
                   else begin v[k]:=w[j]; j:=j+1 end;
      k:=k+1;
    end;
  if i>vm then while k<=v2 do begin v[k]:=w[j]; j:=j+1; k:=k+1; end;
  if j>v2 then while k<=v2 do begin v[k]:=w[i]; i:=i+1; k:=k+1; end;
End;
```

## PRODOTTO DI DUE INTERI

Un altro problema risolvibile mediante la tecnica del divide-et-impera è il prodotto di due interi di n cifre in base B.

Sapendo che il livello di complessita del calcolo dipende da tutte i possibili prodotti tra le cifre, otteniamo che in generale il costo sarà di $O(n^2)$; tuttavia per mezzo della tecnica divide-et-impera, questo risultato si può migliorare.

Siano x, y due numeri interi di n cifre in base B; supponiamo per semplicità che n sia una potenza di 2; possiamo scrivere:

$$x = x_1 \cdot B^{n/2} + x_0 \quad y = y_1 \cdot B^{n/2} + y_0$$

dove $x_0$, $x_1$, $y_0$, $y_1$ sono numeri interi in base B, di n/2 cifre. Il prodotto può essere espresso nel seguente modo:

$$x \cdot y = x_0 \cdot y_0 + (x_0 \cdot y_1 + x_1 \cdot y_0) \cdot B^{n/2} + x_1 \cdot y_1 \cdot B^n$$

Il problema è stato diviso in 4 prodotti tra interi di n/2 cifre: si hanno quindi 4 sooto problemi di costo $O(n^2/4)$. Il costo totale non è variato.

Un diverso risultato si verifica ponendo:

$$a = x_0 \cdot y_0 \; ; \qquad b = x_1 \cdot y_1 \; ; \qquad c = (x_0 - x_1) \cdot (y_1 - y_0) \; ;$$

il problema diviene:

$$x \cdot y = a + (a + b + c) \cdot B^{n/2} + b \cdot B^n \, .$$

Ci siamo ricondotti quindi al calcolo di 3 prodotti di interi di n/2 cifre. Si può quindi iterare questo procedimento, semplificando sempre più i calcoli, ottenendo la seguente relazione di ricorrenza:

$$T(n) \leq \begin{cases} b \text{ (costante)} & \text{se } n = 1 \\ 3T(n/2) + bn & \text{se } n > 1 \end{cases}$$

la cui soluzione è data da:

$$T(n) \leq O(n^{\lg_2(3)}) \sim O(n^{1,59}).$$