Strutture Dati

Lezione 18
Quicksort

Oggi parleremo di …

- Quicksort
  - algoritmo
  - implementazione
  - analisi della complessità

Ordinamento veloce (quicksort)

- L'array A[p…u] viene "suddiviso" intorno ad un elemento perno in due sottoarray A[p…q-1] e A[q+1…u] in cui ogni elemento di A[p…q-1] è minore di ogni elemento di A[q+1…u] :
  - l'algoritmo sceglie un valore dell'array che fungerà da elemento "spartiacque" tra i due sottoarray, detto valore pivot.
  - sposta i valori maggiori del pivot verso l'estremità destra dell'array e i valori minori verso quella sinistra.
- q dipenderà dal valore del pivot: sarà l'indice della posizione in cui si troverà alla fine l'elemento pivot.
- Solitamente si sceglie come pivot il primo elemento dell'array.

Ordinamento veloce (quicksort)

```
Quicksort(A,p,u)
 IF p < u
  THEN q = Partiziona(A,p,u)
       Quicksort(A,p,q - 1)
       QuicksSort(A,q + 1,u)
```

L'elemento scelto come pivot è il primo elemento dell'array, cioè A[p].

La partizione sull'array A[p,…,u] produce la permutazione seguente:

$$\underbrace{A[p \ldots q\text{-}1]}_{<\ \text{pivot}},\quad \underbrace{A[q]}_{\text{pivot}},\quad \underbrace{A[q+1 \ldots u]}_{>\ \text{pivot}}$$

Ricorsivamente si ordinano gli elementi da p a q-1 e da q+1 ad u.

Chiave dell'algoritmo è la partizione dell'array intorno al pivot!

Ordinamento veloce (quicksort)

Partizione

```
int Perno(float A[ ], int primo, int ultimo)
{
  /*
  organizza gli elementi A[primo]....A[ultimo] intorno al perno
  restituendo prima gli elementi minori del perno,
  poi il perno,
  quindi gli elementi maggiori del perno
  */

  int i = primo;
  int j = ultimo + 1;
  int pivot = A[primo];

  do{
       do i++;  while(A[i]<pivot && i<j);
       do j--;  while(A[j]>pivot && j>primo);
       if(i<j)  Scambia(&A[i],&A[j]);
     }while(i<j);
  Scambia(&A[primo], &A[j]) ;
  return j;
}
```

$O(n)$

```
void Scambia(float *f1, float *f2)
  { float tmp = *f1 ; *f1 = *f2 ; *f2 = tmp ; }
```

Ordinamento veloce (quicksort)

```
void Quicksort(float A[ ], int u, int v)

 /* ordina A[u]....A[v] in ordine non decrescente
    A[u] e' scelto come perno
    q sara' la posizione assunta dal perno nell'ordinamento
 */
{
 int q;

 if(u == v) return;
 q = Perno(A, u, v);
 if( u < q )  Quicksort(A, u, q-1);
 if( q < v )  Quicksort(A, q+1, v);
}
```

Complessità del quicksort

- Caso peggiore
  - insieme ordinato
  - insieme inversamente ordinato
- Caso migliore
  - partizione in due sottoinsiemi quasi uguali (ogni volta!)
- Caso medio

- La complessità del quicksort è descritta dalla seguente relazione di ricorrenza:

$$T(n) \leq cn + T(j) + T(n-j-1) \qquad \text{con } T(0) = 0$$

dove j è la posizione assunta dal pivot
- gli elementi minori del pivot vanno da 0 a j-1
- gli elementi maggiori del pivot vanno da j+1 a n-1.

Complessità nel caso peggiore

- Caso peggiore
  - insieme ordinato
  - insieme inversamente ordinato
- Uno dei due sottoinsiemi è sempre vuoto:

$$T(n) \leq cn + T(j) + T(n-j-1) \Longrightarrow T(n) \leq cn + T(0) + T(n-1)$$

con $T(0) = 0$

$$\begin{aligned} T(n) &\leq cn + T(0) + T(n-1) \\ &= cn + T(n-1) \\ &\leq cn + c(n-1) + T(n-2) \\ &= c[n + (n-1)] + T(n-2) \\ &\leq c[n + (n-1) + (n-2)] + T(n-3) \end{aligned}$$

$$T(n) \leq c[n + (n-1) + ... + 1] + T(0)$$

$$T(n) \leq c\sum_{i=1}^{n} i + 0$$

$$T(n) \leq c\frac{n(n-1)}{2} \Longrightarrow \boxed{T(n) \in O(n^2)}$$

Complessità nel caso migliore

- Caso migliore
  - partizione in due sottoinsiemi quasi uguali (ogni volta!)

$$T(n) \leq cn + T(j) + T(n-j-1) \Longrightarrow T(n) \leq cn + 2T(n/2)$$

con $T(0) = 0$ Posto $n = 2^h$

$$\begin{aligned} T(2^h) &\leq c2^h + 2T(2^{h-1}) \\ \frac{T(2^h)}{2^h} &\leq \frac{T(2^{h-1})}{2^{h-1}} + c \\ &\leq \frac{T(2^{h-2})}{2^{h-2}} + 2c \\ &\leq \frac{T(2^{h-3})}{2^{h-3}} + 3c \end{aligned}$$

$$\frac{T(2^h)}{2^h} \leq \frac{T(2^{h-h})}{2^{h-h}} + hc \; = hc$$

$$\frac{T(2^h)}{2^h} \leq c\log n$$

$$T(n) \leq cn\log n \Longrightarrow \boxed{T(n) \in O(n\log n)}$$

Complessità in media

- Caso medio
  - Sia $T_{med}(n)$ il tempo previsto per ordinare n elementi.
  - $T_{med}(n) \in O(n\log n)$

$$\begin{aligned} T_{med}(n) &\leq cn + \frac{1}{n}\sum_{j=0}^{n-1}[T_{med}(j) + T_{med}(n-j-1)] \\ &= cn + \frac{1}{n}[T_{med}(0) + T_{med}(n-1) + ... + T_{med}(n-1) + T_{med}(0)] \\ &= cn + \frac{2}{n}\sum_{j=0}^{n-1}T_{med}(j) \qquad \text{per } n \geq 2 \end{aligned}$$

Ipotesi: $T_{med}(0) \leq b \qquad T_{med}(1) \leq b$

Tesi: $T_{med}(n) \leq kn\log n$ per $n \geq 2$ e $k = 2(b+c)$

Complessità in media

Dimostrazione per induzione:

Base: $n = 2 \qquad T_{med}(n) \leq cn + \frac{2}{n}\sum_{j=0}^{n-1}T_{med}(j)$

$$T_{med}(2) \leq c \times 2 + \frac{2}{2}[T_{med}(0) + T_{med}(1)] \quad \text{poiché } T_{med}(0) \leq b \quad T_{med}(1) \leq b$$

$$\leq 2c + b + b = 2(b+c) = k \leq kn\log_e 2$$

Ipotesi: $T_{med}(n) \leq kn\log n$ per $1 \leq n < m$

$$\begin{aligned} T_{med}(m) &\leq cm + \frac{2}{m}\sum_{j=0}^{m-1}T_{med}(j) \\ &= cm + \frac{2}{m}[T_{med}(0) + T_{med}(1)] + \frac{2}{m}\sum_{j=2}^{m-1}T_{med}(j) \end{aligned}$$

poiché $T_{med}(0) \leq b$, $T_{med}(1) \leq b$, $T_{med}(j) \leq kj\log j$

Complessità in media

$$\begin{aligned} T_{med}(j) &\leq cm + \frac{4b}{m} + \frac{2}{m}k\sum_{j=2}^{m-1} j\log j \\ &\leq cm + \frac{4b}{m} + \frac{2k}{m}\int_2^m x\log x\,dx \\ &\leq cm + \frac{4b}{m} + \frac{2k}{m}\left[\frac{m^2\log m}{2} - \frac{m^2}{4}\right] \\ &= cm + \frac{4b}{m} + km\log m - \frac{km}{2} \quad \leq km\log m \end{aligned}$$

$T_{med}(n) \leq kn\log n$ per ogni $n \geq 2$ $\Longrightarrow \boxed{T_{med}(n) \in O(n\log n)}$

Complessità del quicksort

- Caso peggiore ⟹ $T(n) \in O(n^2)$
- Caso migliore ⟹ $T(n) \in O(n \log n)$
- Caso medio ⟹ $T_{med}(n) \in O(n \log n)$

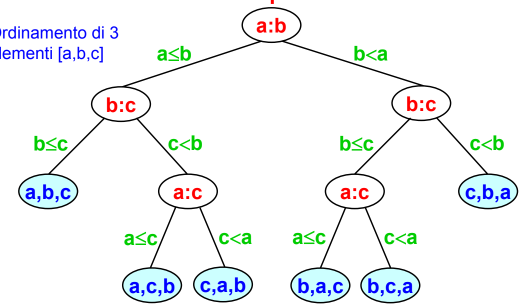
Complessità degli algoritmi semplici

- Selezione ⟹ $T(n) \in O(n^2)$
- Inserzione ⟹ $T(n) \in O(n^2)$

Quale è il **limite inferiore** per un algoritmo di ordinamento?

Albero decisionale per l'ordinamento

Ordinamento di 3 elementi [a,b,c]



Il numero delle possibili soluzioni è 3!=6.
L'albero non è un albero binario completo di altezza 4: ha meno di $2^{4-1} = 8$ nodi terminali.
L'albero decisionale ha un numero di foglie sufficienti per ordinare 3 elementi.

Limite inferiore per l'ordinamento

- Ogni albero decisionale che ordina n elementi distinti ha un'altezza almeno pari a $\log_2(n!)+1$
  - ordinando n elementi, si hanno $n!$ possibili soluzioni
  - ogni albero decisionale deve avere almeno $n!$ foglie
  - se k è l'altezza dell'albero, poiché il numero massimo di foglie di un albero binario è 2^{k-1}, k deve essere tale che $2^{k-1} \geq n!$
  - l'altezza k deve essere maggiore o uguale a $\log_2(n!)+1$
- Qualsiasi algoritmo di ordinamento deve avere un tempo di $\Omega(n \log n)$
  - nell'albero decisionale esiste un percorso di lunghezza $\log_2(n!)$
  - $n! = n(n-1)(n-2) \ldots (3)(2)(1) \geq (n/2)^{(n/2)}$
  - $\log_2(n!) \geq (n/2) \log_2(n/2) \in O(n \log n)$