Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Il problema dell'ordinamento: algoritmo Quicksort

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

Sulla base delle slides a cura di T. Calamoneri e G. Bongiovanni per il corso di informatica generale AA 2019/2020

Quicksort

L'algoritmo ***quicksort*** (***ordinamento veloce***) ha costo $O(n^2)$ nel caso peggiore ma nella pratica è spesso la soluzione migliore per grandi valori di *n* perché:

- il suo tempo di esecuzione atteso è $\Theta(n \log n)$
- i fattori costanti nascosti sono molto piccoli
- permette l'ordinamento "in loco".

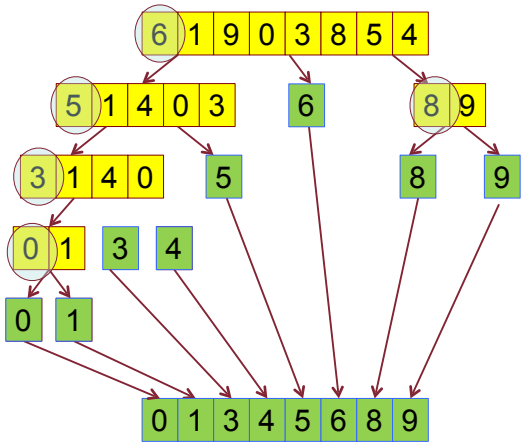
Riunisce i vantaggi del Selection sort (ordinamento in loco) e del Merge sort (ridotto tempo di esecuzione). Ha però lo svantaggio dell'elevato costo computazionale nel caso peggiore.

Quicksort

Anche l'algoritmo ***quicksort*** è un algoritmo ricorsivo che adotta una tecnica algoritmica detta ***divide et impera***:

- ***divide***: nella sequenza di *n* elementi si seleziona un ***pivot***. Il pivot viene posizionato nella sua giusta posizione in modo da ottenere due sottosequenze :quella degli elementi minori o uguali al pivot, e quella degli elementi maggiori al pivot
- ***impera***: le due sottosequenze vengono ordinate ricorsivamente
- ***passo base*:** la ricorsione procede fino a quando le sottosequenze sono costituite da un solo elemento che è ovviamente ordinato.
- ***combina***: non occorre.

Quicksort



- ***divide***: nella sequenza di *n* elementi seleziona un ***pivot***. Viene individuata la giusta posizione del pivot e otteniamo due sottosequenze: quella degli elementi minori o uguali del pivot, e quella degli elementi maggiori del pivot;
- ***impera***: le due sotto-sequenze vengono ordinate ricorsivamente;
- ***passo base*:** la ricorsione procede fino a quando le sottosequenze sono costituite da un solo elemento;
- ***combina***: non occorre

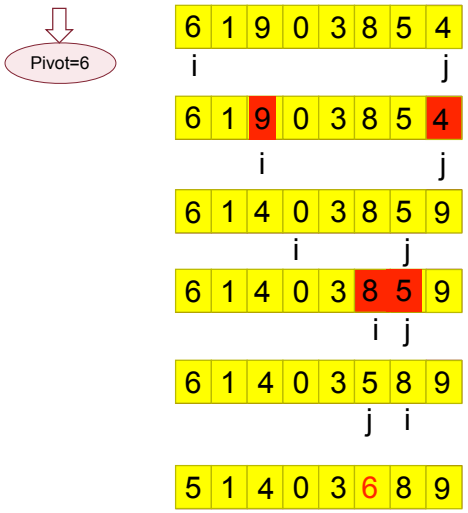
Lo pseudocodice del Quicksort è il seguente:

```
def Quick_sort(A, i, j):
   if i < j:
     m = Partiziona(A, i, j)
     Quick _sort(A, i, m-1)
     Quick _sort(A, m+1, j)
```

In questa implementazione `m` è l'indice della posizione corretta assunta dall'elemento di pivot (alla sua sinistra ci sono elementi minori o uguali del pivot ed alla sua destra elementi maggiori del pivot. Il suo valore quindi è ***sempre*** compreso fra `i` e `j`

Quicksort

funzionamento di partiziona su un esempio



Quicksort

```
def Partiziona(A, i, j):
  pivot = i          #scelta arbitraria
  while i<j:
       while i<=j and A[j]>A[pivot]:
            j -= 1
       while i<=j and A[i]<=A[pivot]:
            i += i
       if i < j:
            A[i],A[j] = A[j],A[i] #scambio  A[i] e A[j]
            i+=1
            j-=1
  A[pivot],A[j] = A[j],A[pivot] #scambio A[pivot] e A[j]
  return j
```

Quicksort

Analizziamo il costo computazionale della funzione `Partiziona()`:

- Le prime tre istruzioni costano $\Theta(1)$.
- Il `while` ha un costo pari alla somma dei costi di ciò che avviene al suo interno, che è $\Theta(n)$ poiché:
  - ciascuna iterazione di ognuno dei due `while` costa *Θ(1)* e avvicina di una posizione un indice all'altro;
  - quindi complessivamente si effettuano $\Theta(n)$ iterazioni dei due `while`;
  - terminati i due `while` si trova un `if` ed un possibile scambio di elementi con conseguente incremento degli indici, $\Theta(1)$

Dunque il costo di `Partiziona()` è $\Theta(n)$.

NOTA: il valore j restituito da `Partiziona()` sulla lista di n elementi vale:

- 0 se il pivot è più piccolo degli altri elementi;
- $n-1$ se il pivot è più grande degli altri elementi.

Quicksort

Valutiamo ora il costo computazionale del Quicksort:

```
def Quick_sort(A, i, j):
   if i < j:
         m = Partiziona(A, i, j)
         Quick _sort(A, i, m-1)
         Quick _sort(A, i, m+1, j)
```

$T(n)$
$\Theta(1)$
$\Theta(n)$
$T(k)$
$T(n-k)$

$$T(n) = T(k) + T(n-k) + \Theta(n) \quad \text{se } n \geq 2$$
$$= \Theta(1) \quad \text{altrimenti}$$

Equazione che non sappiamo risolvere con alcuno dei metodi studiati…

Quicksort

Possiamo facilmente derivare la soluzione per due situazioni, il caso migliore e quello peggiore.

- Caso migliore:
  è quello in cui, ad ogni passo, la dimensione dei due sotto-problemi è identica. L'equazione di ricorrenza diventa:
  $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$ che ha soluzione $T(n) = \Theta(n \log n)$

- Caso peggiore:
  è quello in cui, ad ogni passo, la dimensione di uno dei due sotto-problemi da risolvere è 1. L'equazione di ricorrenza diventa:
  $T(n) = T(n-1) + \Theta(n)$ che ha soluzione $T(n) = \Theta(n^2)$

QuickSort

Valutiamo ora il costo computazionale nel caso medio, nell'ipotesi che la posizione corretta del pivot possa essere con uguale probabilità una qualunque delle n posizioni del vettore.

Questo significa che dalla sottosequenza di dimensione n con uguale probabilità $\frac{1}{n}$ vengono generate da ordinare due sottosequenze di dimensione k e n-1-k rispettivamente con $0 \leq k < n$. Abbiamo dunque

$$T(n) = \frac{1}{n}\left[\sum_{k=0}^{n-1}(T(k) + T(n-1-k))\right] + \Theta(n)$$

Ora, per ogni valore di q e $0 \leq q < n$ il termine $T(q)$ compare due volte nella sommatoria, la prima quando $k = q$ e la seconda quando $k = n-1-q$. Valutiamo dunque il valore di

$$T(n) = \frac{2}{n}\sum_{q=0}^{n-1} T(q) + \Theta(n) \text{ per } n \geq 2$$
$$T(n) = \Theta(1) \text{ altrimenti}$$

Quick Sort

Per risolvere la ricorrenza utilizziamo il metodo di sostituzione, e quindi eliminiamo per prima cosa la notazione asintotica:

$$T(n) = \frac{2}{n}\sum_{q=0}^{n-1} T(q) + \Theta(n) \quad \text{per } n \geq 2$$
$$= \Theta(1) \quad \text{altrimenti}$$

$$T(n) = \frac{2}{n}\sum_{q=1}^{n-1} T(q) + a \cdot n \quad \text{per } n \geq 2$$
$$= k \quad \text{altrimenti}$$

Ipotizziamo ora la soluzione $T(n) = O(n \log n)$. Per provarlo basterà dimostrare che esiste una costante c per cui si ha:

$$T(n) \leq c \cdot n \log_2 n \quad \text{per } n \geq 2$$

Quicksort

Sostituiamo la soluzione innanzi tutto nel caso base.

$$T(2) = \frac{2}{2}\sum_{q=1}^{1} T(q) + 2a = k + 2a$$

deve quindi aversi:

$$T(2) = k + 2 \cdot a \leq 2 \cdot c \log_2 2 = 2 \cdot c$$

che è vera per *c* opportunamente grande (basta prendere ad esempio $c \geq k + 2a$).

Per il passo induttivo possiamo scrivere:

$$T(n) = \frac{2}{n}\sum_{q=1}^{n-1} T(q) + an$$
$$\leq \frac{2}{n}\sum_{q=1}^{n-1} c \cdot q \cdot \log_2 q + an$$
$$= \frac{2c}{n}\sum_{q=1}^{n-1} q \cdot \log_2 q + an$$

dimostreremo che che: $\sum_{q=1}^{n-1} q \cdot \log_2 q \leq \frac{n^2}{2}\log_2 n - \frac{n^2}{8}$

possiamo scrivere:

$$T(n) \leq \frac{2c}{n}\left(\frac{n^2}{2}\log_2 n - \frac{n^2}{8}\right) + a \cdot n$$
$$= c \cdot n \cdot \log n - \frac{cn}{4} + a \cdot n$$
$$\leq c \cdot n \cdot \log n$$

dove l'ultima diseguaglianza segue se prendiamo c sufficientemente grande in modo da avere $-\frac{c \cdot n}{4} + a \cdot n \leq 0$ *(vale a dire $c \geq 4a$)*

Resta solo da dimostrare che $\sum_{q=1}^{n-1} q \cdot \log_2 q \leq \frac{n^2}{2}\log_2 n - \frac{n^2}{8}$.

Valutiamo ora la sommatoria $\sum_{q=1}^{n-1} q \log q$, *spezzandola in due:*

$$\sum_{q=1}^{n-1} q \cdot \log q = \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q \cdot \log q + \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q \cdot \log q$$

$\leq \log \frac{n}{2} = \log n - 1$; $\leq \log n$

Abbiamo dunque

$$\sum_{q=1}^{n-1} q \cdot \log q \leq \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q \cdot (\log n - 1) + \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q \cdot \log n$$

Dunque possiamo scrivere:

$$\sum_{q=1}^{n-1} q \cdot \log q \leq \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q \cdot (\log n - 1) + \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q \cdot \log n$$
$$= (\log n - 1) \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q + \log n \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q$$
$$= \log n \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q - \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q + \log n \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q$$
$$= \log n \sum_{q=1}^{n-1} q - \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q$$

Ora,

$$\log n \sum_{q=1}^{n-1} q - \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q = \log_2 n \frac{(n-1)n}{2} - \frac{\left(\left\lceil \frac{n}{2} \right\rceil - 1 \right) \left\lceil \frac{n}{2} \right\rceil}{2}$$

$$\leq \log_2 n \frac{(n-1)n}{2} - \frac{\left(\frac{n}{2} - 1 \right) \frac{n}{2}}{2}$$

$$= \frac{n^2}{2} \log_2 n - \frac{n}{2} \log_2 n - \frac{n^2}{8} + \frac{n}{4}$$

$$\leq \frac{n^2}{2} \log_2 n - \frac{n^2}{8}$$

Ricapitolando:

$$\sum_{q=1}^{n-1} q \cdot \log_2 q \leq \log_2 n \sum_{q=1}^{n-1} q - \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q \leq \frac{n^2}{2} \log_2 n - \frac{n^2}{8}$$

Quicksort

Abbiamo appena dimostrato che nel caso medio si ha $T(n) = O(n \log n)$.

Inoltre in media non si può avere meglio dell'ottimo che sappiamo essere $\Theta(n \log n)$ da questo deduciamo che per il caso medio vale anche $T(n) = \Omega(n \log n)$.

Possiamo dunque concludere che nel caso medio il Quicksort ha un costo computazionale:

$$T(n) = \Theta(n \log n)$$

Osservazioni

L'analisi ora fatta è valida nell'ipotesi che la posizione del pivot sia equiprobabile e, quando questo è il caso, **il quicksort è considerato l'algoritmo ideale per input di grandi dimensioni**.

A volte però l'ipotesi di equiprobabilità non è soddisfatta (ad esempio quando i valori in input sono "poco disordinati") e le prestazioni dell'algoritmo degradano.

Per ovviare a tale inconveniente si possono adottare delle tecniche volte a ***randomizzare*** la sequenza da ordinare, cioè volte a disgregarne l'eventuale regolarità interna.

Tali tecniche mirano a rendere l'algoritmo indipendente dall'input, e quindi consentono di ricadere nel caso medio. Alcune di tali tecniche sono:

- prima di avviare l'algoritmo, alla sequenza da ordinare viene applicata una permutazione degli elementi generata casualmente;
- l'operazione di partizionamento sceglie casualmente come pivot il valore di uno qualunque degli elementi della sequenza anziché sistematicamente il valore di quello più a sinistra.

Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizi - 1

- Sia dato un vettore di lunghezza *n* contenente solo valori 0 e 2. Si progetti un algoritmo con costo computazionale lineare che modifichi il vettore in modo che tutte le occorrenze di 0 si trovino più a sinistra di tutte le occorrenze di 2.
- Si considerino i valori 0 1 2 3 4 5 6 7. Si determini una permutazione di questi valori che generi il caso peggiore per l'algortimo Quick sort.
- Calcolare il costo computazionale del Quick sort nel caso in cui il vettore contenga tutti elementi uguali e poi nel caso in cui sia già ordinato da destra a sinistra.

Esercizi - 2

- Si progetti un algoritmo il più efficiente possibile per i seguenti problemi:
  - Data una matrice *m x n,* si vogliono rimescolare i suoi elementi in modo che tutti i vettori riga e tutti i vettori colonna siano ordinati in senso non decrescente.
  - Data una matrice *n x n,* si vogliono rimescolare i suoi elementi in modo che tutti gli elementi posizionati al di sopra della diagonale principale siano minori o uguali di tutti gli elementi che giacciono sulla diagonale principale che, a loro volta, siano minori o uguali di tutti gli elementi posizionati al di sotto della diagonale principale.