Corso di laurea in Informatica
Progettazione d'algoritmi

Tecnica Divide et Impera 1

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

Il problema della selezione

Data una lista A di n interi distinti, ed un intero k, con $1 \leq k \leq n$, vogliamo sapere quale elemento occuperebbe la posizione k se il vettore venisse ordinato:

Casi particolari:

- Per $k = 1$ avremo il *minimo* di A.
- Per $k = n$ avremo il *massimo* di A.
- Per $k = \left\lceil \frac{n}{2} \right\rceil$ avremo il *mediano* di A.

Il problema della selezione

Un semplice algoritmo di complessità $\Theta(n \log n)$ è il seguente:

```
def selezione1(A, k):
    A.sort()
    return A[k-1]

>>> A = [30, 20, 50, 60, 10, 40, 90, 80, 70]
>>> selezione1(A, 4)
40
```

- Se $k = 1$ o $k = n$ il problema si riduce alla ricerca del *minimo* o del *massimo* e questi casi sono risolvibili in tempo $\Theta(n)$.
- Vedremo che, utilizzando la tecnica del *divide_et_impera*, il problema può risolversi in $\Theta(n)$ anche nel caso generale.
- In altre parole dimostreremo che il problema della selezione è computazionalmente più semplice di quello dell'ordinamento (che richiede invece tempo $\Omega(n \log n)$).

Approccio basato sul Divide et impera:

- scegli nella lista A l'elemento in posizione $A[0]$ (che chiameremo perno).
- a partire da A costruisci due liste A_1 ed A_2, la prima contenente gli elementi di A minori del perto e la seconda gli elementi di A maggiori del perno.
- dove si trova l'elemento di rango k?
  - se $|A_1| \geq k$ allora l'elemento di rango k è nel vettore A_1
  - se $|A_1| = k - 1$ allora l'elemento di rango k è proprio il perno x.
  - se $|A_1| < k - 1$ allora l'elemento di rango k è in A_2 è l'elemento di rango $k - |A_1| - 1$ in A_2

NOTA che dopo aver costruito dalla lista A di n elementi le due liste $A1$ e $A2$, grazie al test sulla cardinalità di $A1$ il problema della selezione dell'elemento di rango k o risulta risolto o viene ricondotto alla selezione di un elemento in una lista con meno di n elementi.

```python
def selezione2(A, k):
    '''restituisce l'elemento di rango k dove 1 <= k <=len(A)'''
    if len( A ) == 1:
        return A[0]
    perno = A[0]
    A1, A2 = [], []
    for i  in range(1, len(A)):
        if A[i] < perno:
            A1.append( A[i] )
        else:
            A2.append( A[i] )
    if len( A1 ) >= k:
        return selezione2(A1, k)
    elif len( A1 ) == k-1:
        return perno
    return selezione2(A2, k - len(A1) - 1)
```

La procedura che tripartisce la lista in $A1$, A[O] e $A2$ può restituire una partizione massimamente sbilanciata in cui si ha ad esempio $|A1| = 0$ e $|A2| = n - 1$, questo accade quando il perno risulta l'elemento minimo nella lista.
Qualora questo evento sfortunato si ripetesse sistematicamente nel corso delle varie partizioni eseguite dall'algoritmo (questo può accadere quando cerco il massimo in una lista ordinata), allora la complessità dell'algoritmo al caso pessimo viene catturata dalla seguente equazione:

$$T(n) = T(n-1) + \Theta(n)$$

che risolta dà

$$T(n) = \Theta(n^2)$$

In generale la complessità superiore alla procedura è catturata dalla ricorrenza

$$T(n) = T(m) + \Theta(n)$$

dove $m = max\{|A1|, |A2|\}$

Se avessimo una regola di scelta del perno in grado di garantire una partizione bilanciata, ossia:

$$m = \max\{|A1|, |A2|\} \approx n/2$$

allora per la complessità $T(n)$ dell'algoritmo avremmo

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(n)$$

che risolta dà $T(n) = \Theta(n)$.

Chiedere che la partizione sia perfettamente bilanciata è forse chiedere troppo, potremmo allora accontentarci di chiedere che la scelta del perno garantisca partizioni non troppo sbilanciate come ad esempio quelle per cui

$$m = \max\{|A1|, |A2|\} \approx 3n/4.$$

In questo caso si ha

$$T(n) \leq T\left(\frac{3}{4}n\right) + \Theta(n)$$

che risolta dà ancora $T(n) = \Theta(n)$.

In generale finché m è una frazione di n (anche piuttosto vicina ad n come ad esempio $\frac{99}{100}n$) la ricorrenza dà sempre $T(n) = \Theta(n)$.

IDEA:

scegliamo il perno p a caso in modo equiprobabile tra gli elementi della lista.

Anche se la scelta "casuale" non produce necessariamente una partizione bilanciata, quanto visto ci fa intuire che la complessità rimane lineare in n.

```
def selezione2R(A,k):
    '''restituisce l'elemento di rango k dove 1 <= k <=len(A)'''
    if len(A) == 1:
        return A[0]
    perno = A[ randint(0, len(A)-1) ]
    A1, A2 = [], []
    for x in A:
        if x < perno:
            A1.append( x )
        elif x > perno:
            A2.append( x )
    if len(A1) >= k:
        return selezione2R(A1, k)
    elif len(A1) == k-1:
        return perno
    return selezione2R(A2, k - len(A1) - 1)
```

Analisi formale del caso medio:

Con la randomizzazione introdotta per la scelta del perno possiamo assumere che uno qualunque degli elementi del vettore, con uguale probabilità $\frac{1}{n}$, diventi perno e, poiché la scelta dell'elemento di rango k produce $|A1| = k-1$ e $|A2| = n-k$, per il tempo atteso dell'algoritmo va studiata la ricorrenza:

$$T(n) \leq \frac{1}{n}\sum_{k=1}^{n} T\Big(\max\big(T(k-1),\ T(n-k)\big)\Big) + \Theta(n) \leq \frac{1}{n}\sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} 2T(k) + \Theta(n)$$

possiamo dimostrare che per questa ricorrenza vale T(n)=O(n) col metodo di sostituzione. Studiando

PROVA per sostituzione:

$$T(n) = \begin{cases} \frac{1}{n}\sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} 2T(k) + a \cdot n & \text{se } n \geq 3 \\ b & \text{altrimenti} \end{cases}$$

Dimostriamo $T(n) \leq cn$ per una qualche $c > 0$ costante.

Per $n \leq 3$ abbiamo $T(n) \leq b \leq 3c$ che è vera ad esempio per $c \geq b$.

Sfruttando l'ipotesi induttiva $T(k) \leq c \cdot k$ per $k < n$ abbiamo $T(n) \leq \frac{2c}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} k + a \cdot n$

da cui ricaviamo

$$T(n) \leq \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor \frac{n}{2} \rfloor - 1} k\right) + a \cdot n \leq \frac{2c}{n}\left(\frac{n(n-1)}{2} - \frac{(n/2-1)(n/2-2)}{2}\right) + a \cdot n \leq \frac{c}{n}\left(\frac{3n^2}{4} + \frac{n}{2} - 2\right) + a \cdot n$$
$$\leq \frac{3cn}{4} + \frac{c}{2} + a \cdot n = cn - \left(\frac{cn}{4} - \frac{c}{2} - a \cdot n\right) \leq cn$$

dove l'ultima diseguaglianza segue prendendo c in modo che $\left(\frac{cn}{4} - \frac{c}{2} - a \cdot n\right) \geq 0$

Basta ad esempio prendere $c \geq 8a$

- L'analisi rigorosa appena fatta dimostra che, se la scelta del perno avviene in modo equiprobabile a caso tra i vari elementi della lista A, il tempo di calcolo dell'algoritmo risulta con alta probabilità lineare in n.
- il tempo dell'algoritmo *selezione2R* è dunque $O(n)$ con alta probabilità.
- Ovviamente nel caso peggiore, quando nelle varie partizioni che si succedono nell'iterazione dell'algoritmo si verifica che il perno scelto a caso risulta sempre vicino al massimo o al minimo della lista, la complessità dell'algoritmo rimane $O(n^2)$. Però questo accade con probabilità molto piccola.

Vedremo ora un algoritmo deterministico che garantisce complessità $O(n)$ anche al caso pessimo.

Abbiamo visto che riuscire a selezionare un perno in grado di garantire che nessuna delle due sottoliste $A1$ e $A2$ abbia più di $c \cdot n$ elementi, per una qualche costante $0 < c < 1$, avrebbe come conseguenza una complessità di calcolo $O(n)$.

Descriviamo ora un metodo (noto come *il mediano dei mediani*) per selezionare un perno che garantisce di produrre sempre due sottoliste $A1$ ed $A2$ ciascuna delle quali ha non più di $\frac{3}{4}n$ elementi.

Algoritmo per la selezione:

- **Dividi l'insieme A, contenente n elementi, in gruppi da 5 elementi ciascuno. L'ultimo gruppo potrebbe avere meno di 5 elementi. Considera soltanto i primi $\lfloor \frac{n}{5} \rfloor$ gruppi, ciascuno composto esattamente da 5 elementi.**
- **Trova il mediano all'interno di ciascuno di questi $\lfloor \frac{n}{5} \rfloor$ gruppi.**
- **Calcola il mediano p dei mediani ottenuti al passo precedente.**
- **Usa p come elemento pivot per l'insieme A.**

ESEMPIO per la scelta del perno:

A=[15, 2, 10, 16, 21,12, 1, 9, 11, 17, 22, 3, 8,13, 14, 4, 9, 19, 5, 6, 20, 23,18, 7]

15, 2, 10, 16, 21 | 12, 1, 9, 11, 17 | 22, 3, 8, 13, 14 | 4, 19, 5, 6, 20 | 23, 18, 7

2,10, 15, 16, 21 | 1, 9, 11,12, 17 | 3, 8, 13, 14, 22 | 4, 5, 6, 19, 20 |

15, 11, 13, 6

6, 11, 13, 15

p=11

PROPRIETA':

Se la lista A contiene almeno 120 elementi e il perno p con cui partizionarla viene scelto in base alla regola appena descritta si può esser sicuri che la dimensione di ciascuna delle due sottoliste $A1$ e $A2$ ottenute sarà limitata da $\frac{3}{4}n$.

PROVA:

Il perno scelto p ha la proprietà di trovarsi in posizione $\left\lceil \frac{n}{10} \right\rceil$ nella lista degli $\left\lfloor \frac{n}{5} \right\rfloor$ mediani selezionati in A. Ci sono dunque $\left\lceil \frac{n}{10} \right\rceil - 1$ mediani di valore inferiore a p e $\left\lfloor \frac{n}{5} \right\rfloor - \left\lceil \frac{n}{10} \right\rceil$ mediani di valore superiore a p.

Prova che $|A2| < \frac{3}{4}n$:

Considera i $\left\lceil \frac{n}{10} \right\rceil - 1$ mediani di valore inferiore a p. Ognuno di questi mediani appartiene ad un gruppo di 5 elementi in n. Ci sono dunque in A altri 2 elementi inferiori a p per ogni mediano. In totale abbiamo $3\left(\left\lceil \frac{n}{10} \right\rceil - 1\right) \geq 3\frac{n}{10} - 3$ elementi di A che finiranno in $A1$.

Abbiamo dunque $|A2| \leq n - \left(3\frac{n}{10} - 3\right) = \frac{7}{10}n + 3 \leq \frac{3}{4}n$ (dove l'ultima diseguaglianza segue dal fatto che $n \geq 120$).

Prova che $|A1| < \frac{3}{4}n$:

Ci sono $\left\lfloor \frac{n}{5} \right\rfloor - \left\lceil \frac{n}{10} \right\rceil \geq \left(\frac{n}{5} - 1\right) - \left(\frac{n}{10} + 1\right) = \frac{n}{10} - 2$ mediani di valore superiore a p. Ognuno di questi mediani appartiene ad un gruppo di 5 elementi in A. Ci sono dunque in A altri 2 elementi superiori a p per ogni mediano. In totale abbiamo almeno $3\frac{n}{10} - 6$ elementi di A che finiranno in $A2$.

Abbiamo dunque $|A1| \leq n - \left(3\frac{n}{10} - 6\right) = \frac{7}{10}n + 6 \leq \frac{3}{4}n$ (dove l'ultima diseguaglianza segue dal fatto che $n \geq 120$)

Implementazione dell'algoritmo selezione:

```
from math import ceil

def selezione(A, k):
    ''' restituisce l'elemento di rango k dove 1<=k<=len(A)
     scegliendo ogni volta il perno con la regola del mediano dei mediani'''
    if len(A) <= 120:
        A.sort()
        return A[k-1]
    # inizializza B con i mediani dei len(A)//5 gruppetti di 5 elementi di A
    B=[ sorted( A[5*i : 5*i+5] )[2] for i in range( len(A)//5 ) ]
    #individua il perno p con la regola del mediano dei mediani
    perno = selezione(B, ceil(len(A)/10) )
    A1, A2 = [], []
    for x in A:
        if x < perno: A1.append(x)
        elif x > perno: A2.append(x)
    if len(A1) >= k:
        return selezione(A1, k)
    elif len(A1) == k-1:
        return perno
    return selezione(A2, k - len(A1) - 1)
```

Implementazione dell'algoritmo selezione:

```python
from math import ceil

def selezione(A, k):
    ''' restituisce l'elemento di rango k dove 1<=k<=len(A)
     scegliendo ogni volta il perno con la regola del mediano dei mediani'''
    if len(A) <= 120:
        A.sort()
        return A[k-1]
    # inizializza B con i mediani dei len(A)//5 gruppetti di 5 elementi di A
    B=[ sorted( A[5*i : 5*i+5] )[2] for i in range( len(A)//5 ) ]
    #individua il perno p con la regola del mediano dei mediani
    perno = selezione(B, ceil(len(A)/10) )
    A1, A2 = [], []
    for x in A:
        if x < perno: A1.append(x)
        elif x > perno: A2.append(x)
    if len(A1) >= k:
        return selezione(A1, k)
    elif len(A1) == k-1:
        return perno
    return selezione(A2, k - len(A1) - 1)
```

Complessità:

- nota che:
  - ordinare 120 elementi richiede tempo $O(1)$.
  - ordinare una lista di n elementi in gruppetti da 5 richiede tempo $\Theta(n)$.
  - selezionare i mediani dei mediani di gruppi da 5 da una lista in cui gli elementi sono stati ordinati in gruppetti da 5 richiede tempo $\Theta(n)$.

- Sappiamo che per $n \geq 120$ risulta $|lista1| \leq \frac{3}{4}n$ e $|lista2| \leq \frac{3}{4}n$.
  Dunque per la complessità $T(n)$ dell'algoritmo si ha

$$T(n) \leq \begin{cases} O(1) & \text{se } n \leq 120 \\ T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + \Theta(n) & \text{altrimenti} \end{cases}$$

  Notiamo che la ricorrenza è del tipo

$$T(n) = T(\alpha \cdot n) + T(\beta \cdot n) + \Theta(n) \text{ con } \alpha + \beta = 1/5 + 3/4 = 19/20 < 1.$$

  mostreremo nel prossimo lucido che ricorrenze di questo tipo hanno tutte

  come soluzione

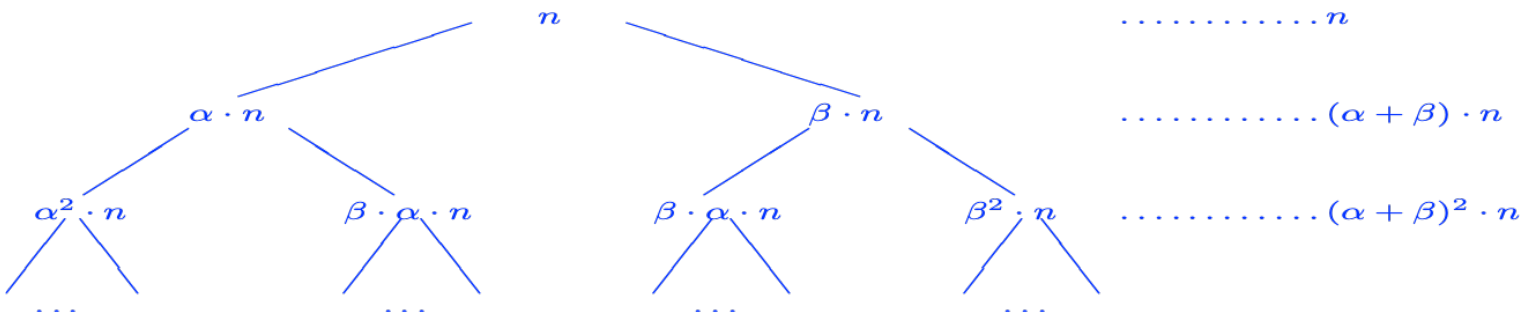
$$T(n) = \Theta(n).$$

ESERCIZIO:

Se $T(n) = T(\alpha \cdot n) + T(\beta \cdot n) + cn$ e $\alpha + \beta < 1$, allora $T(n) = \Theta(n)$.

PROVA:

- Il fatto che $\alpha + \beta$ sia inferiore ad 1 gioca un ruolo fondamentale nella prova.
  Consideriamo l'albero delle chiamate ricorsive generato dalla ricorrenza e analizziamone il costo per livelli.



- Al primo livello abbiamo un costo $(\alpha+\beta)\cdot n$, al secondo un costo $(\alpha+\beta)^2\cdot n$ al terzo un costo $(\alpha+\beta)^3 \cdot n$ e così via.

- Il tempo di esecuzione totale è la somma dei contributi dei vari livelli:

$$T(n) < c\cdot n + c\cdot(\alpha+\beta)\cdot n + c\cdot(\alpha+\beta)^2\cdot n \ldots = c\cdot n\cdot\sum_{i=0}^{\infty}(\alpha+\beta)^i = c\cdot n\frac{1}{1-(\alpha+\beta)} = \Theta(n)$$

dove nel calcolare la serie abbiamo sfruttato il fatto che $\alpha + \beta < 1$ e la serie geometrica $\sum_{i=0}^{\infty} x^i$, con $x < 1$, converge a $\frac{1}{1-x}$.

Abbiamo quindi dimostrato che il problema della selezione può essere risolto in tempo lineare.
Abbiamo infatti un algoritmo che risolve il problema in $O(n)$ al caso pessimo.
Tuttavia, a causa delle grandi costanti moltiplicative nascoste dall'$O(n)$, nella pratica l'algoritmo randomizzato che ha tempo $O(n)$ con alta probabilità si comporta molto meglio.