Ricerca di una chiave

Data una sequenza lineare di n elementi memorizzata in un array, la ricerca di una chiave K consiste nel verificare se la sequenza contiene un elemento il cui valore è uguale a K.

Se la sequenza non è ordinata, è necessario esaminare tutti gli elementi. Costo $O(n)$.

```
                                  array di n elementi
                                  |   chiave da ricercare
                                  v   v
RicercaSequenziale (a , K)

indice = 0;

WHILE (indice < n && a[indice] != K)
  {
     indice = indice + 1;
  }

IF (indice >= n) indice = -1;

RETURN indice;
```

L'algoritmo scandisce tutto l'array, elemento dopo elemento. Se trova K, l'indice restituito alla fine è la posizione dell'array in cui K viene trovata. Altrimenti è -1.

Se la sequenza è ordinata possiamo far meglio.

Ricerca binaria

Il costo si riduce a $O(\log n)$ nel caso pessimo.

Idea: Supponiamo che l'array a sia ordinato in senso non decrescente. Usiamo semplici confronti tra due elementi.

Confrontiamo K con l'elemento in posizione centrale nell'array, a $[n/2]$.

- se $K < a[n/2]$ ripetiamo l'operazione sulla prima metà dell'array
- se $K > a[n/2]$ ripetiamo l'operazione sulla seconda metà dell'array
- se $K = a[n/2]$, l'abbiamo trovata.

Il procedimento termina nel momento in cui K coincide con l'elemento centrale oppure la "metà" diventa vuota, nel qual caso la chiave K non c'è.

Possiamo formalizzare questa strategia di ricerca attraverso una funzione ricorsiva

RicercaBinariaRicorsiva (a, K, sinistra, destra)

a → array di n elementi ORDINATO
K → chiave

$0 \leq sinistra \leq destra \leq n-1$

```
IF ( sinistra == destra) {
   IF ( K == a[sinistra]) {
      RETURN sinistra ;
   }
    ELSE {
      RETURN -1 ;
   }
}
centro = (sinistra + destra)/2 ;
IF ( K ≤ a[centro]) {
   RETURN RicercaBinariaRicorsiva (a, K, sinistra, centro);
}  ELSE {
   RETURN RicercaBinariaRicorsiva (a, K, centro+1, destra);
}
```

Il costo può essere espresso facilmente usando una ricorrenza. Infatti, se il segmento dell'array in cui stiamo cercando ha un solo elemento, il numero di operazioni eseguito è costante.

Altrimenti, il costo è dato da una costante più il numero di passi richiesto dalla ricerca su un segmento di taglia dimezzata. Pertanto:

$$T(n) \leq \begin{cases} c_0 & \text{se } n \leq 1 \\ T(n/2) + c & \text{altrimenti.} \end{cases}$$

Applicando il Master Theorem, risultano:

$$\alpha = 1, \quad \beta = 2 \quad \text{e} \quad f(n) = 1$$

Pertanto,

$$\alpha f(n/\beta) \leq \gamma f(n) \iff 1 \cdot 1 \leq \gamma \cdot 1$$

Quindi, per $\gamma = 1$ e $n_0' = 1$, la precedente risulta vera $\forall n \geq n_0$.

Siamo nel caso 2. e possiamo concludere che la soluzione della ricorrenza è $O(f(n) \cdot \log_\beta n) = O(\log_2 n)$.

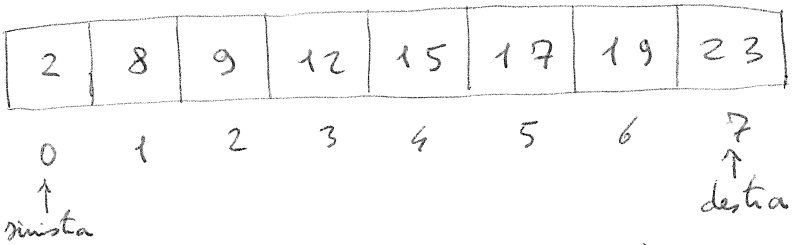
Abbiamo allora dimostrato il seguente teorema.

Teorema. La ricerca binaria in un array ordinato richiede $O(\log n)$ passi.

Esempio di esecuzione

chiave $K = 15$

$a =$



RicercaBinariaRicorsiva (a, 15, 0, 7)

↓ centro $= \lfloor \frac{0+7}{2} \rfloor = 3$ $\quad K = 15 > 12 = a[3]$

RicercaBinariaRicorsiva (a, 15, 4, 7)

↓ centro $= \lfloor \frac{4+7}{2} \rfloor = 5$ $\quad K = 15 \leq 17 = a[5]$

RicercaBinariaRicorsiva (a, 15, 4, 5)

↓ centro $= \lfloor \frac{4+5}{2} \rfloor = 4$ $\quad K = 15 \leq 15 = a[4]$

RicercaBinariaRicorsiva (a, 15, 4, 4)

sinistra = destra = 4

$15 = a[4]$

RETURN 4 ;

Nota: se la chiave cercata fosse stata $K = 13$ avremmo eseguito esattamente gli stessi passi ma all'ultimo confronto, essendo $13 \neq a[4]$ l'algoritmo avrebbe restituito -1.

Lower bound per gli algoritmi di ricerca di una chiave basati su operazioni di confronto

Teorema 3.3. Ogni algoritmo di ricerca per confronti richiede nel caso peggiore $\Omega(\log n)$ confronti.

Dimostrazione. L'argomento è identico a quello usato per provare il lower bound per gli algoritmi di ordinamento per confronti.

In questo caso ci sono $n+1$ situazioni possibili da distinguere (1 per l'assenza della chiave e n per ogni posizione dell'array in cui la chiave può trovarsi).

Poiché t confronti permettono di "distinguere" 2^t possibili situazioni, deve essere

$$2^t \geq n+1$$

ovvero, $t \geq \log(n+1) = \Omega(\log n)$ ∎

Possiamo concludere allora che la complessità del PROBLEMA della ricerca di una chiave usando confronti è $\Theta(\log n)$ e che la ricerca binaria è un algoritmo OTTIMO.

Osservazioni: Se l'array ordinato contiene occorrenze multiple delle chiavi (i.e., è un multi-insieme) il codice restituisce la posizione più a sinistra della chiave K.

Il codice, inoltre, può essere modificato per calcolare il rango r della chiave K.
Il valore r è tale che $0 \leq r \leq n$ ed è definito come il numero di elementi dell'array a che sono minori o uguali a K.

Precisamente, le prime righe diventano:

```
IF ( sinistra == destra){
    IF ( K < a[sinistra])}
      RETURN sinistra ;
    }ELSE{
      RETURN sinistra+1;
    }
}
```

È facile osservare che, nei due esempi considerati in precedenza, il calcolo del rango della chiave K=15 nell'array a restituirebbe r=5, mentre per la chiave K=13 si avrebbe r=4.

Analogamente possiamo cercare in un array a una chiave K quando l'array è ordinato in modo non crescente, semplicemente invertendo l'ordine delle chiamate ricorsive.

Esercizio svolto. Progettare un algoritmo non ricorsivo che implementi la ricerca binaria all'interno di un array ordinato.

```
                                  array ordinato
                                  ↙        ← chiave K.
RicercaBinariaIterativa (a, K)
 sinistra = 0;
 destra = n-1;
 trovato = FALSE;
 indice = -1;
  WHILE ((sinistra ≤ destra) && (!trovato)) {
    centro = (sinistra + destra)/2;
    IF (a[centro] > K) {
        destra = centro-1;
    }ELSE IF (a[centro] < K) {
        sinistra = centro+1;
    } ELSE {
        indice = centro;
        trovato = TRUE;
    }
 }
 RETURN indice;
```

Esempio di esecuzione

chiave
K = 15

a =

2	8	9	12	15	17	19	23
0	1	2	3	4	5	6	7

sinistra = 0, destra = 7

trovato = FALSE;
indice = -1;

Prima iterazione: centro = $\lfloor (0+7)/2 \rfloor = 3$ a[3] = 12 < 15

⇒ sinistra = centro + 1 = 3 + 1 = 4;

Seconda iterazione: ...

15	17	19	23
4	5	6	7

sinistra = 4, destra = 7

centro = $\lfloor (4+7)/2 \rfloor = 5$ a[5] = 17 > 15

⇒ destra = centro - 1 = 5 - 1 = 4;

Terza iterazione: ...

15
4

sinistra = 4, destra = 4

centro = (sinistra + destra)/2 = 4 a[4] = 15 = 15

⇒ indice = 4; trovato = TRUE;

RETURN 4;

D'altra parte, è facile osservare che, se la chiave cercata

Fosse stata $K = 13$, la terza iterazione sarebbe stata

terza iterazione: ... | 15 | ...

sinistra destra

centro = (sinistra + destra)/2 = 4 a $T[4] = 15 > 13$

$\Rightarrow$ destra = centro − 1 = 4 − 1 = 3 ;

In questo modo, la successiva verifica della condizione del ciclo WHILE fallisce (sinistra = 4 > 3 = destra)

e la funzione esegue RETURN −1 (indice è inserito a −1)

Nota: la strategia dicotomica rappresentata dalla ricerca binaria è utile in tutti quei problemi in cui vale una proprietà monotona booleana $P(x)$ al variare dell'intero x, cioè

$P(x)$ è vera per $x \leq x_0$

$P(x)$ è falsa per $x > x_0$

o viceversa, per un qualche intero x_0.

Usando lo schema della ricerca binaria possiamo trovare il valore di x_0.

Esempio 3.3

Un amico sceglie segretamente un numero $n > 0$.

Vogliamo definire uno schema di domande efficiente per individuare n.

Schema banale (e inefficiente...): è $n=1$? è $n=2$? ...

richiede esattamente n domande.

Schema efficiente: domande del tipo "è $x \leq x_0$?"

dove x_0 è il valore n da individuare.

Le domande "è $x \leq x_0$?" sono $x = 1, 2, 4, 8 \ldots$ (potenze crescenti di 2)

Non appena risulta $x = 2^h > x_0$ ci fermiamo e sappiamo che:

$$2^{h-1} \leq \underset{\| \; n}{x_0} < 2^h$$

Il numero di domande effettuate è h: $2^h > n \iff h > \log n$

ovvero $h \leq \log n + 1 = O(\log n)$

A questo punto, posso applicare la ricerca binaria per individuare $x_0 = n$ nell'intervallo $[2^{h-1}, 2^h - 1]$ che

contiene $2^h - 1 - 2^{h-1} = 2 \cdot 2^{h-1} - 1 - 2^{h-1} = 2^{h-1} - 1 \leq 2^{h-1} \leq 2^{\log n} \leq n$

al più n elementi.

Pertanto occorrono altre $O(\log n)$ domande.

In totale lo schema richiede $O(\log n) + O(\log n) = O(\log n)$ domande. Un bel risparmio rispetto ad n.

Istanza concreta

Sia $x_0 = 58$

Domande: $1 \le x_0$? ✓, $2 \le x_0$? ✓, $4 \le x_0$? ✓, $8 \le x_0$? ✓

$16 \le x_0$? ✓, $32 \le x_0$? ✓ $64 \le x_0$? ✗

$32 \le x_0 < 64$ $\quad h \le \log 58 + 1$

$2^5 \qquad 2^6$

<

La ricerca binaria su 

richiede altre 5 domande $\qquad 2^5$ elementi

Se inverto l'ordine nelle domande, cioè "è $x_0 \ge x$?" ottengo lo stesso schema con ricerca finale nell'intervallo

$[2^{h-1}+1, \; 2^h]$ (invece che in $[2^{h-1}, \; 2^h - 1]$)