

Corso di laurea in Informatica
Algoritmi 1
A.A. 2025/26

Il problema della ricerca

Tiziana Calamoneri



SAPIENZA
UNIVERSITÀ DI ROMA



Slides realizzate sulla base di quelle preparate da T. Calamoneri e G. Bongiovanni per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

Sommario

Il problema della ricerca

Ricerca sequenziale e suo costo computazionale

Ricerca binaria e suo costo computazionale

Ricerca in tempo costante

Il problema della ricerca (1)

Nell'informatica esistono alcuni problemi particolarmente rilevanti, poiché essi:

- si incontrano in una grande varietà di situazioni reali;
- ricorrono come sottoproblemi da risolvere nell'ambito di problemi più complessi.

Uno di questi problemi è la **ricerca di un elemento in un insieme di dati** (ad es. numeri, cognomi, ecc.).

Il problema della ricerca (2)

Definiamo più formalmente tale problema descrivendone l'input e l'output:

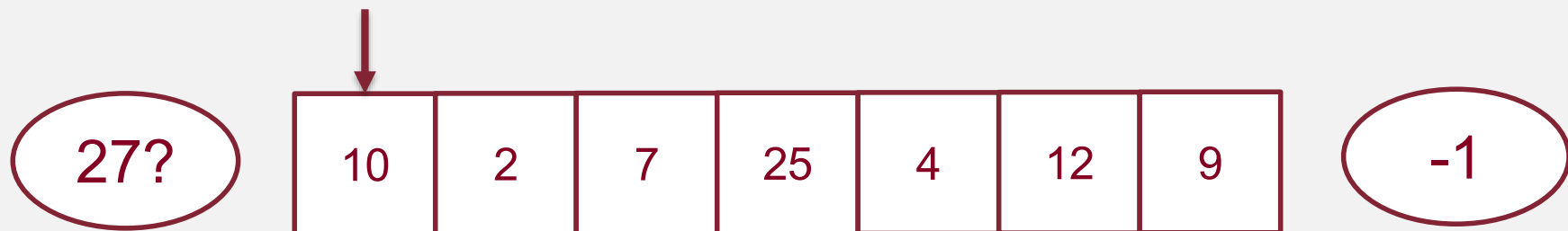
- Input: un **array** A di n valori (interi, stringhe, ecc.) ed un **valore** v ;
- Output: un **indice** i tale che $A[i] = v$, oppure un particolare valore **None** se il valore v non è presente nell'array.

Ricerca sequenziale (1)

Un primo semplice algoritmo è basato su questa idea:

- ispezioniamo uno alla volta gli elementi dell'array;
- confrontiamo ciascun elemento con v ;
- restituiamo il risultato, interrompendoci appena (non) trovato v

Vediamo il caso in cui v non è contenuto nell'array:



Ricerca sequenziale (2)

```
def Ricerca (A;v) :  
    i = 0  $\Theta(1)$   
    while ( (i < len(A)) and (A[i] ≠ v) )  $\Theta(1)$  + al più n volte:  
        i += 1  $\Theta(1)$   
    if (i < len(A)) :  $\Theta(1)$   
        return i  $\Theta(1)$   
    else:  
        return -1  $\Theta(1)$ 
```

Ricerca sequenziale (3)

```
Funzione Ricerca (A;v)
i = 1                                 $\Theta(1)$ 
while ((i < len(A)) and (A[i] ≠ v))  $\Theta(1)$  + al più n volte:
    i += 1                             $\Theta(1)$ 
if (i < len(A)): return i             $\Theta(1)$ 
else: return null                     $\Theta(1)$ 
```

Costo computazionale:

- $\Theta(n)$ nel caso peggiore (quando, cioè, v non è contenuto nell'array)
- $\Theta(1)$ nel caso migliore (quando v viene incontrato per primo),

Ricerca sequenziale (4)

Non avendo trovato una stima del costo che sia valida per tutti i casi, diremo che il costo computazionale dell'algoritmo (in generale, non nel caso peggiore) è un $O(n)$, per evidenziare che ci sono input in cui questo costo viene raggiunto, ma altri in cui il costo è minore.

Ricerca sequenziale (5)

Quando i casi migliore e peggiore si discostano, si può dare un valore stretto per il costo computazionale, ma possiamo domandarci quale sia il costo dell'algoritmo nel caso medio.

ipotesi: v può apparire con uguale probabilità in qualunque posizione, ossia:

$$P(v \text{ si trovi in } i\text{-esima posizione}) = \frac{1}{n}$$

Allora il numero medio di iterazioni è dato da:

$$\frac{1}{n} \sum_{i=1}^n i = \frac{1}{n} \frac{n(n+1)}{2} = \frac{n+1}{2}$$

Ricerca sequenziale (6)

ipotesi (alternativa): tutte le possibili $n!$ permutazioni della sequenza sono equiprobabili.

Di queste, ve ne saranno un certo numero in cui v appare in prima posizione, un certo numero nelle quali v appare in seconda posizione, ecc.

Il numero medio di iterazioni sarà quindi:

Numero medio di iterazioni =

$$= \sum_{i=1}^n i \frac{\text{numero di permutazioni in cui } v \text{ è in posizione } i}{\text{numero totale di permutazioni}}$$

Ricerca sequenziale (7)

Il numero di permutazioni nelle quali v appare nella i -esima posizione è uguale al numero delle permutazioni di $(n-1)$ elementi, cioè $(n-1)!$, dato che fissiamo solo la posizione di uno degli n elementi.

Quindi: Numero medio di iterazioni =

$$= \sum_{i=1}^n i \frac{(n-1)!}{n!} = \frac{1}{n} \sum_{i=1}^n i = \frac{n+1}{2}$$

💡 Stesso risultato con due ipotesi di equiprobabilità diverse. Dunque, è ragionevole che il costo nel caso medio sia un $\Theta(n)$.

nota: NON è una
dimostrazione generale!!!

Ricerca sequenziale (8)

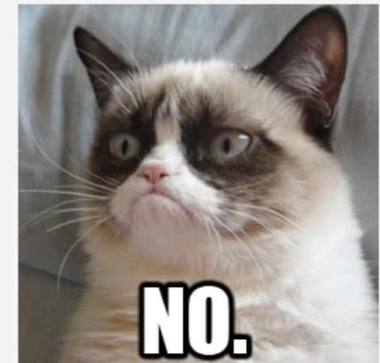
⚠ ATTENZIONE:

L'operatore *in* del Python che verifica se un elemento sia contenuto in un oggetto di tipo *list* utilizza una ricerca sequenziale, perciò ha costo computazionale $O(n)$.

Ad esempio, il seguente frammento di codice:

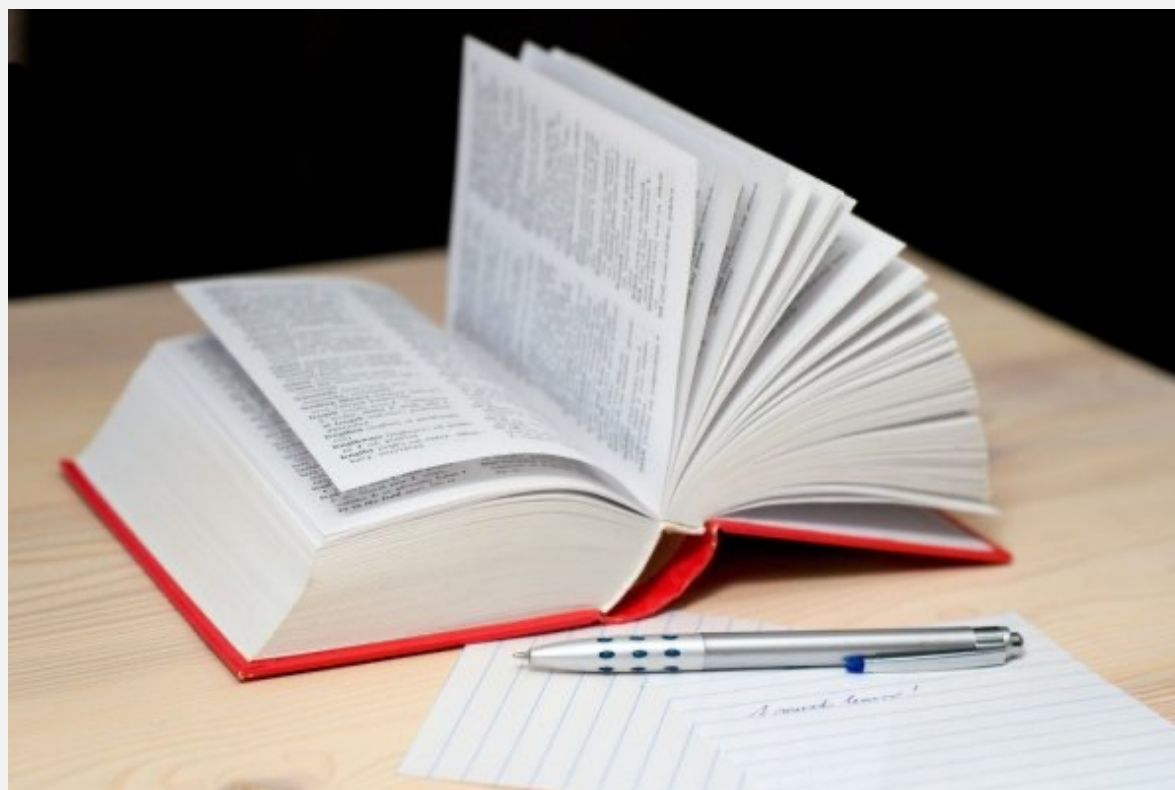
```
if v in A:
    print('presente')
else:
    print('assente')
```

ha costo $O(n)$ nonostante non comprenda (apparentemente!) alcuna istruzione iterativa!



Ricerca binaria (1)

Ma nella vita di tutti i giorni, non cerchiamo così:



Ricerca binaria (2)

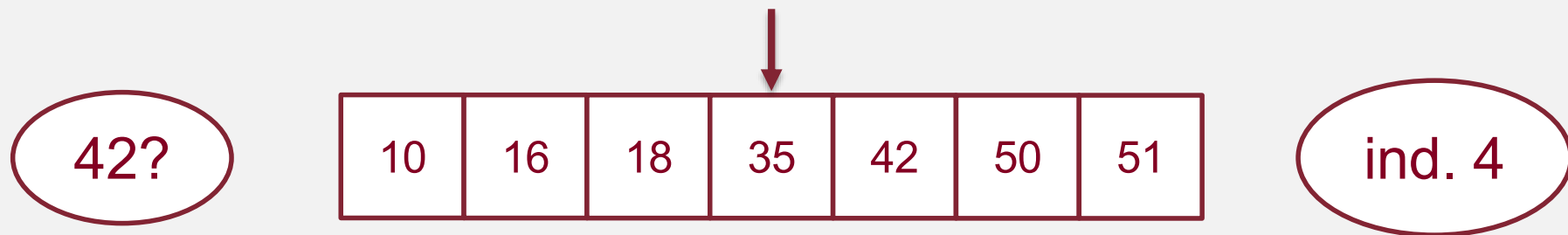
L'algoritmo della **ricerca binaria** funziona come segue:

Si ispeziona l'elemento centrale della sequenza:

- se esso è uguale al valore cercato ci si ferma;
- se il valore cercato è più piccolo si prosegue nella sola metà inferiore della sequenza, altrimenti nella sola metà superiore.

Ricerca binaria (3)

Vediamo un esempio nel quale l'elemento cercato (42) è presente.



Ricerca binaria (4)

```
def Ricerca_binaria(A,v):  
    a,b=0, len(A)-1  
    m =(a+b) //2  
    while (A[m] !=v):  
        if (A[m] > v):  
            b=m - 1  
        else:  
            a=m + 1  
        if a > b:  
            return -1  
        m=(a+b) //2  
    return m
```


Ricerca binaria (5)

Costo computazionale?

Informalmente:

ad ogni iterazione si dimezza il numero degli elementi su cui lavorare, per cui il numero di iterazioni cresce come $\log n$.

Questo risultato è molto forte:

per trovare un elemento in una sequenza ordinata di un miliardo di valori bastano circa 30 iterazioni!

Ricerca binaria (6)

```
def Ricerca_binaria (A, v):  
    a,b= 0, len(A)  
    m= (a+b)//2  
    while (A[m] != v):  
        if (A[m] > v): b=m-1  
        else: a=m+1  
        if (a > b): return -1  
        m= (a+b)//2  
    return m
```

Più formalmente:

all'i-esima iterazione la dimensione dell'input è $\frac{n}{2^i}$ quindi

l'ultima i è $\log n$

In altre parole, il ciclo **while** viene eseguito al più $\log n$ volte; pertanto il costo è:

- $\Theta(\log n)$ nel caso peggiore (l'elem. non c'è);
- $\Theta(1)$ nel caso migliore (l'elem. viene trovato al primo colpo).

Ricerca binaria (7)

Poiché i casi migliore e peggiore non hanno lo stesso costo, valutiamo il costo del caso medio...

Assunzioni:

- il numero di elementi è una potenza di 2 (per semplicità di calcolo, ma è facile vedere che ciò non modifica il risultato finale);
- v è presente nella sequenza (altrimenti si ricade nel caso peggiore);
- tutte le posizioni di v fra 1 ed n sono equiprobabili.

Ricerca binaria (8)

Quante sono le posizioni $n(i)$ che possono essere controllate alla i -esima iterazione?

- con una iterazione si raggiunge la sola posizione $n/2$, cioè $n(1)=2^0=1$;
 - con esattamente due iterazioni si raggiungono le due posizioni $\frac{n}{4}$ e $3\frac{n}{4}$, cioè $n(2)=2^1=2$;
 - con esattamente tre iterazioni si raggiungono le quattro posizioni $\frac{n}{8}$, $3\frac{n}{8}$, $5\frac{n}{8}$, $7\frac{n}{8}$, cioè $n(3)=2^2=4$;
- e così via...

Ricerca binaria (9)

...

In generale, la ricerca binaria esegue i iterazioni se v si trova in una delle $n(i)=2^{i-1}$ posizioni raggiungibili con esattamente tale numero di iterazioni.

La probabilità che l'elemento da trovare si trovi su una delle $n(i)$ posizioni toccate dalla i -esima iterazione è $n(i)/n$ (ossia $2^{i-1}/n$), perciò il numero medio di iterazioni è:

...

Ricerca binaria (10)

$$\text{numero medio di iterazioni} = \sum_{i=1}^{\log n} i \frac{n(i)}{n} = \frac{1}{n} \sum_{i=1}^{\log n} i 2^{i-1}$$

$$\text{dove } \sum_{i=1}^k i 2^{i-1} = (k-1)2^k + 1$$

per cui (sostituendo $k=\log n$):

$$\frac{1}{n} \left((\log n - 1) 2^{\log n} + 1 \right) = \log n - 1 + \frac{1}{n}$$

Ossia, il numero medio di iterazioni si discosta per meno di un confronto dal numero massimo di iterazioni!

Ricerca costante (1)

Problema: Data una sequenza di DNA **S** ed un certo frammento **F**, **F** è presente in **S**?

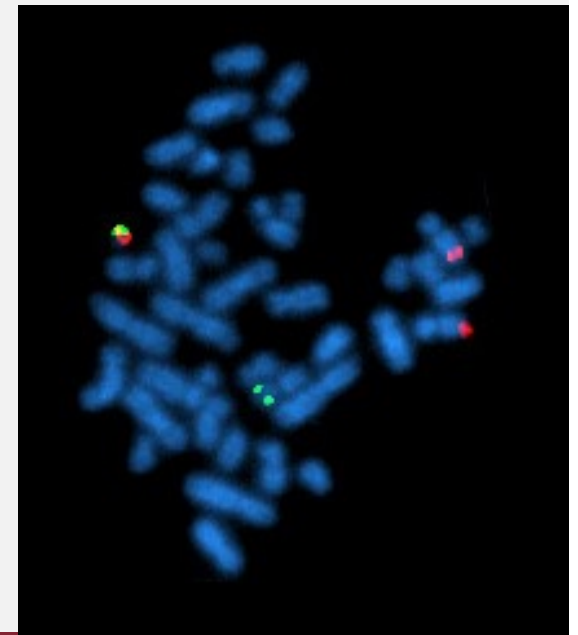
Ibridizzazione: processo che unisce due eliche singole di DNA in un'unica elica doppia (frammenti di singole eliche si attraggono e formano un'elica doppia se essi sono uno il complementare dell'altro).

Sonda: piccolo frammento di elica singola di DNA che ha una sequenza nota ed è fluorescente.

Ricerca costante (2)

L' ibridizzazione di una sonda in un frammento di DNA sconosciuto (= un singolo passo computazionale su una sorta di ipotetica macchina molecolare) evidenzia la presenza della sequenza complementare a quella della sonda nel frammento.

Cioè: algoritmo di ricerca con costo costante se consideriamo un diverso modello computazionale...



Esercizi per casa



SAPIENZA
UNIVERSITÀ DI ROMA



Esercizio per casa

(dal compito d'esame del 18/2/2021)

Sia dato un array A di interi e due valori a e b con $a \leq b$, il problema è quello di sapere quanti elementi di A sono compresi nell'intervallo chiuso $[a, b]$.

- Si progetti un algoritmo per risolvere tale problema su qualsiasi array A .
- Si progetti un algoritmo più efficiente del precedente assumendo che A sia già ordinato e che contenga solo valori distinti.

Per ciascun algoritmo si scriva lo pseudocodice e si analizzi il tempo di esecuzione asintotica.