

# Il problema del dizionario

Oggetto

k	...	...	...	...
---	-----	-----	-----	-----

chiave

dati satellite

$x$  = puntatore all'oggetto

**Dizionario** = insieme dinamico che consente di effettuare le Operazioni di ricerca, inserimento, rimozione.

**Search( $S, k$ )** - dato un insieme  $S$  ed un valore chiave  $k$  restituisce un puntatore  $x$  ad un elemento in  $S$  tale che  $key[x] = k$

**Insert( $S, x$ )** - inserisce in  $S$  l'elemento puntato da  $x$

**Delete( $S, x$ )** - rimuove da  $S$  l'oggetto puntato da  $x$

Ci chiediamo qual e' la migliore realizzazione possibile per un dizionario.

## Esempio

Dizionario per la gestione del corso di algoritmi

N.ordine (puntatore)	Nome (chiave)	Voto Parziale (dati satellite)
1)	Panozaqi	????
2)	Facciolo	????
3)	Businaro	?????
4)	...	
5)	Destro	
6)	...	
7)	Destro	
8)	...	
9)	Cestari	
...		

Metodo piu' semplice → **array non ordinato**

In questo caso:

Insert → costa  $O(1)$  - inserisco dopo Cestari  
Search → **costa  $O(n)$**  - devo scorrere la tabella  
Delete → costa  $O(n)$  - delete = search + trasferimento

Possiamo ridurre il tempo di esecuzione della operazione di ricerca?

Il problema della ricerca di un elemento in un array non ordinato ha **delimitazione inferiore  $\Omega(n)$**

-----

Ogni algoritmo deve, nel caso peggiore, guardare tutti gli elementi dell'insieme per accertarsi o meno della presenza dell'elemento cercato.

-----

L'operazione di ricerca costa di meno in un **array ordinato**.  
Posso usare il metodo di ricerca per dimezzamenti successivi

```
Cerca2(A,k,p,r)
if (k<A[p]) or (k>A[r])
    then return 0
if (k=A[p])
    Then return p
if (k=A[r])
    Then return r
if (p < r)
    Then q =  $\lfloor (p+r)/2 \rfloor$ 
        if (k ≤ A[q])
            Then Cerca2(A,k,p,q)
            Else Cerca2(A,k,q+1,r)
        Else return 0
```

**Esercizio** - Scrivere la versione iterativa di cerca2.

La ricerca per dimezzamenti successivi ha un costo  $O(\log(n))$ . Posso fare meglio?

NO

Ogni algoritmo per la ricerca di un elemento in un insieme di  $n$  elementi richiede  $\Omega(\log(n))$  confronti.

Infatti -

- Ogni algoritmo deve restituire la posizione dell'elemento tra le  $n$  possibili;
- Ad ogni algoritmo posso associare un albero di decisione;
- L'albero di decisione deve avere  $n$  foglie;
- L'altezza dell'albero fornisce un lower bound alla complessita' di un generico algoritmo;
- Il numero di foglie di un albero di altezza  $h$  e' al piu'  $2^h$

Quindi:

$$n = \text{N. Foglie} < 2^h \rightarrow h > \log_2(n)$$

Per un generico algoritmo

$$T(n) = \Omega(\log_2(n))$$

Il metodo di ricerca per dimezzamenti successivi e' ottimale.

## Array Ordinato

Search -  $O(\log(n))$

Insert -  $O(n)$

Ho bisogno di:

$O(\log(n))$  confronti → per trovare la giusta posizione in cui inserire l'elemento

$O(n)$  trasferimenti → per mantenere l'array ordinato

(Ricordo che  $T(n) = O(n) + O(\log(n)) = O(n)$ )

Delete -  $O(n)$

## Array non Ordinato

Search -  $O(n)$

Insert -  $O(1)$

Delete -  $O(n)$

## Lista non Ordinata

Search -  $O(n)$

Insert -  $O(1)$

Delete -  $O(n)$

## Lista Ordinata

Search -  $O(n)$  Costerebbe comunque  $n$  (non posso fare dim. successivi)

Insert -  $O(n)$  Devo mantenere ordinata la lista

Delete -  $O(n)$