



# Corso di Introduzione agli algoritmi
# Prof.ssa Tiziana Calamoneri

## Strutture dati fondamentali: Insiemi dinamici ed operazioni su di essi con le strutture dati semplici

# Strutture dati (1)



- In un linguaggio di programmazione, un dato è un valore che una variabile può assumere.
- Un tipo di dato astratto è un modello matematico, dato da una collezione di valori e un insieme di operazioni ammesse su questi valori.
- Le strutture dati sono particolari tipi di dato, caratterizzate dall'organizzazione dei dati, più che dal tipo di dato stesso.

# Strutture dati (2)

- Una struttura dati è quindi composta da:
  - un modo sistematico di organizzare i dati
  - un insieme di operatori che permettono di manipolare la struttura
- Le strutture dati possono essere:
  - lineari o non lineari (a seconda che esista o no una sequenzializzazione)
  - statiche o dinamiche (a seconda che possano variare la dimensione nel tempo)
  - omogenee o disomogenee (rispetto ai dati contenuti).

# Insiemi dinamici (1)



- Una struttura dati serve a memorizzare e manipolare ***insiemi dinamici***, cioè insiemi i cui elementi possono variare nel tempo in funzione delle operazioni compiute dall'algoritmo che li utilizza.

# Insiemi dinamici (2)

- Gli elementi dell'insieme dinamico (spesso chiamati anche ***record***) possono essere piuttosto complessi e contenere ciascuno più di un dato "elementare". In tal caso è abbastanza comune che essi contengano:
  - una ***chiave***, utilizzata per distinguere un elemento da un altro nell'ambito delle operazioni di manipolazione dell'insieme dinamico; normalmente i valori delle chiavi fanno parte di un insieme totalmente ordinato (ad. es., sono numeri interi);
  - ulteriori dati, detti ***dati satellite***, che sono relativi all'elemento stesso ma non sono direttamente utilizzati nelle operazioni di manipolazione dell'insieme dinamico.
- In altri casi, ogni elemento contiene solo la chiave e quindi coincide con essa.
- Nel seguito ci riferiremo sempre a questa situazione semplificata, a meno che non venga esplicitamente specificato il contrario.

# Insiemi dinamici (3)

Le tipiche operazioni che si compiono su un insieme dinamico $S$ (e quindi sulla struttura dati che ne permette la gestione), che si suppone totalmente ordinabile, si dividono in due categorie:

- *operazioni di interrogazione*
- *operazioni di modifica*

## Insiemi dinamici (4)



Tipici esempi di operazioni di interrogazione, che non modificano la consistenza dell'insieme dinamico, sono:

- Search(S, k): recuperare l'elemento con chiave di valore *k*, se è presente in *S*, restituire un valore speciale nullo altrimenti;
- Min(S): recuperare il minimo valore presente in *S*;
- Max(S): recuperare il massimo valore presente in *S*;
- Predecessor(S, k): recuperare l'elemento presente in *S* che precederebbe quello di valore *k* (di cui supponiamo di conoscere la locazione) se *S* fosse ordinato;
- Successor(S, k): recuperare l'elemento presente in *S* che seguirebbe quello di valore *k* (di cui supponiamo di conoscere la locazione) se *S* fosse ordinato.

# Insiemi dinamici (5)

Tipici esempi di operazioni di manipolazione, che invece modificano la consistenza dell'insieme dinamico, sono:

- Insert(S, k): inserire un elemento di valore *k* in *S*;
- Delete(S, k): eliminare da *S* l'elemento di valore *k* (di cui supponiamo di conoscere la locazione).

# Insiemi dinamici (6)

Le differenti strutture dati che vedremo, se da un lato hanno in comune la capacità di memorizzare insiemi dinamici, dall'altro differiscono anche profondamente fra loro per le ***proprietà*** che le caratterizzano.

Sono proprio le proprietà della struttura dati ad essere l'elemento determinante nella scelta da effettuare quando si deve progettare un algoritmo per risolvere un problema.

# Vettori (1)



Prima di procedere allo studio di alcune struture dati, vediamo il costo delle principali operazioni su insiemi dinamici quando questi vengano memorizzati su semplici vettori, disordinati o ordinati.

Oss. Il vettore è considerato come una struttura statica: in alcuni linguaggi di programmazione questi possono variare la loro dimensione, ma ciò viene consentito solo "simulando" la struttura dati vettore con una struttura dati dinamica…

# Vettori (2)

## Search(S,k)

- vettore disordinato: bisogna scorrere il vettore: $O(n)$.
- vettore ordinato: ricerca binaria: $O(\log n)$

## Min(S)-Max(S)

- vettore disordinato: bisogna scorrere il vettore: $\Theta(n)$.
- vettore ordinato: primo o ultimo elemento: $\Theta(1)$

## Predecessor(S,k)-Successor(S,k)

- vettore disordinato: bisogna scorrere il vettore: $\Theta(n)$.
- vettore ordinato: elemento precedente o seguente: $\Theta(1)$

# Vettori (3)

## Insert(S,k)

- vettore disordinato: inserimento in ultima pos.: $\Theta(1)$.
- vettore ordinato: ricerca della posizione, scorrimento a destra degli elementi maggiori ed inserimento: $O(n)$

## Delete(S,k)

- vettore disordinato: eliminazione e scambio con l'ultimo elemento: $\Theta(1)$.
- vettore ordinato: eliminazione e scorrimento per coprire lo spazio lasciato: $O(n)$.

# Vettori (4)

Riassumendo:

| Struttura dati | Search(S,k) | Minimum(S)<br>Maximum(S) | Predecessor(S,k)<br>Successor(S,k) | Insert(S,k) | Delete(S,k) |
|---|---|---|---|---|---|
| Vettore qualunque | $O(n)$ | $\Theta(n)$ | $\Theta(n)$ | $\Theta(1)$ | $O(1)$ |
| Vettore ordinato | $O(\log n)$ | $\Theta(1)$ | $\Theta(1)$ | $O(n)$ | $O(n)$ |

Già da questo semplice confronto delle due strutture dati più semplici in assoluto, si può dedurre come non abbia senso dire che una struttura è migliore di un'altra: le strutture dati possono essere più o meno adatte ad un certo algoritmo e sta al buon progettista disegnare un algoritmo efficiente usando la struttura dati più adatta.

# Liste semplici (1)

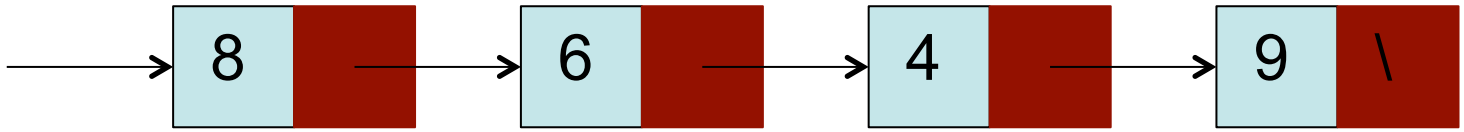
La ***lista semplice*** è una struttura dati nella quale gli elementi sono organizzati in successione.

Proprietà specifiche delle liste:

- l'accesso avviene sempre ad una estremità della lista, per mezzo di un puntatore alla testa della lista;
- è permesso solo un accesso sequenziale agli elementi.

Vettore: numero di elementi prefissato e successione degli elementi realizzata mediante gli indici degli elementi

Lista: può crescere di dimensioni e successione degli elementi implementata mediante un collegamento esplicito da un elemento ad un altro (ad es., tramite un puntatore).



# Liste semplici (2)

## Insert(S,k)

Funzione Insert_in_testa (p: puntatore alla lista; k: puntatore all'elemento da inserire)

```
if (k ≠ NULL)
        next[k] ← p
p ← k
return p
```

Il costo computazionale dell'inserimento è $\Theta(1)$.

# Liste semplici (3)

## Search(S,k)

Funzione Search (p: puntatore alla lista; k: valore)

```
    p_corr = p
    while ((p_corr ≠ NULL) and (key[p_corr] ≠ k))
          p_corr ← next[p_corr]
    return p_corr
```

Il costo computazionale della ricerca è *O(n)*.

# Liste semplici (4)

## Delete(S,k)

Funzione Delete (p: punt. alla lista; k: puntat. all'elem. da cancellare)

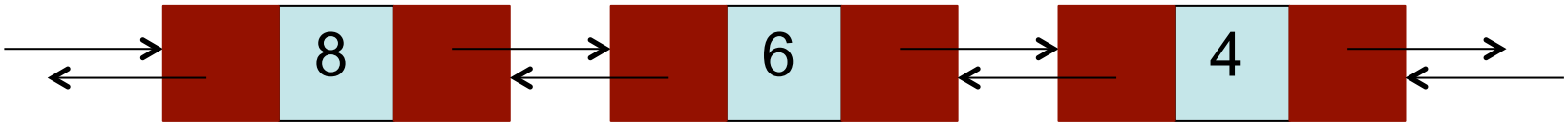
```
if (k ≠ NULL) // se k=NULL non c'è niente da cancellare
        if k = p // cancel. 1° elem
                p← next[p]; free(k); return(p)
        p_corr ←p
        while (next[p_corr] ≠ k)
                p_corr ← next[p_corr] // qui, p_corr punta all'elem.
                                      che precede k
        next[p_corr] ← next[k]; free(k);
return p
```

Il costo computazionale della cancellazione è *O(n)*.

# Liste doppiamente puntate (1)



Alcuni problemi riscontrati nella lista semplice (ad esempio complessità lineare nella cancellazione) possono essere risolti organizzando la struttura dati in modo che da ogni suo elemento si possa accedere sia all'elemento che lo segue che a quello che lo precede nella lista, quando essi esistono. Tale struttura dati si chiama *lista doppia* o *lista doppiamente concatenata* o *lista doppiamente puntata*.



Riassumendo:

| Struttura dati | Search(S,k) | Minimum(S) Maximum(S) | Predecessor(S,k) | Insert(S,k) | Delete(S,k) |
|---|---|---|---|---|---|
| Lista semplice | $O(n)$ | $\Theta(n)$ | $\Theta(n)$ | $\Theta(1)$ | $O(n)$ |
| Lista doppia | $O(n)$ | $\Theta(n)$ | $\Theta(n)$ | $\Theta(1)$ | $\Theta(1)$ |

# Esercizi (1)

Progettare degli algoritmi che risolvano i seguenti problemi; calcolare poi il costo computazionale:

- data in input una lista tramite il puntatore al primo elemento, restituire il puntatore all'ultimo elemento;
- data in input una lista tramite il puntatore al primo elemento, restituire il puntatore al penultimo elemento;
- data in input una lista tramite il puntatore al primo elemento, restituire il puntatore alla stessa lista da cui sia stato eliminato l'ultimo elemento;
- data in input una lista tramite il puntatore al primo elemento, restituire il puntatore di una lista che contenga gli stessi record della lista di partenza ma in ordine inverso (N.B. non deve essere creato alcun record, ma bisogna "smontare" e "rimontare" opportunamente i record iniziali).

# Esercizi (2)

(segue)

- data in input una lista tramite il puntatore al primo elemento, restituire tramite due variabili globali i puntatori a due liste, una con gli elementi di posto pari nella lista di partenza, ed una con gli elementi di posto dispari (anche qui, non bisogna creare nuovi record);
- data in input una lista di interi tramite il puntatore al primo elemento, stampare tutti i valori che compaiono almeno due volte nella lista;
- data in input una lista ordinata di interi tramite il puntatore al primo elemento, ed un elemento da inserire, aggiungere tale elemento alla lista in modo da rispettare l'ordinamento;
- data in input una lista di interi tramite il puntatore al primo elemento, restituire la lista ordinata (senza creare nuovi record).