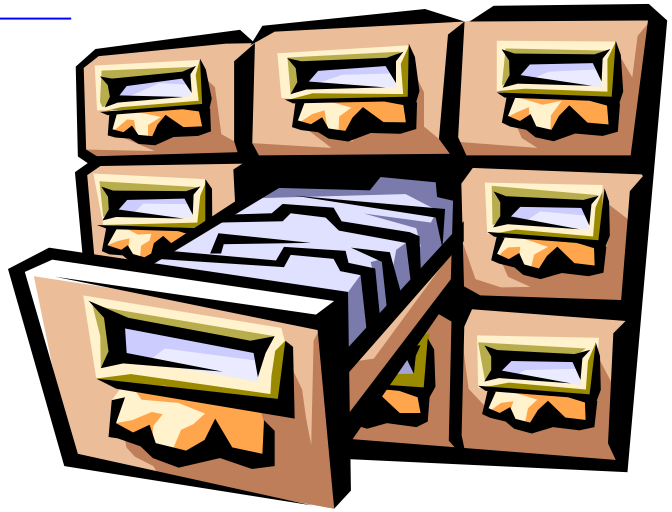
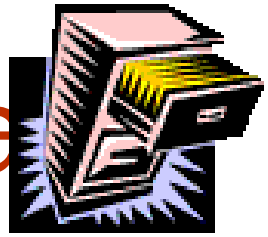
Mappe



Mappe

- Una mappa modella una collezione di elementi su cui facciamo ricerche usando una chiave di ricerca
- Gli elementi possono essere visti come coppie chiave-valore (k,v)
- Le operazioni fondamentali sono ricerca, inserimento e cancellazione di elementi
- Non è ammesso più di un elemento con la stessa chiave
- Esempi di applicazioni:
  - Agenda di indirizzi
  - Database dei dati di studenti

La struttura dati astratta



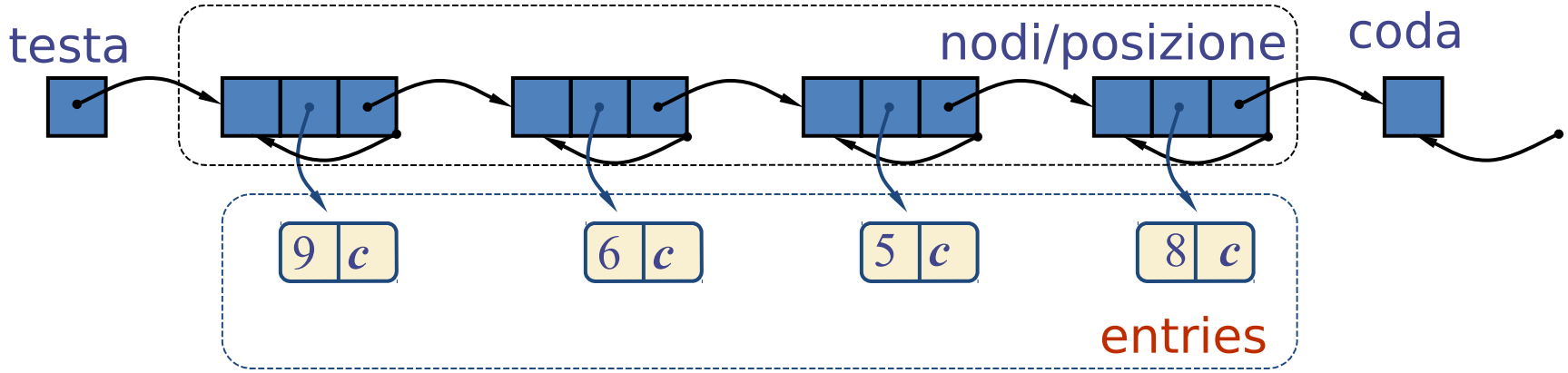
- get(k): se la mappa M ha un elemento con chiave k restituisce il valore associato; altrimenti restituisci null
- put(k, v): inserisci elemento (k, v) nella mappa M; se la chiave k non è presente in M allora ritorna null; altrimenti restituisci il vecchio valroe associato alla chiave k
- remove(k): se la mappa M ha un elemento con chiave k, rimuovilo da M e restituisci il valore associato; altrimenti restituisci null
- size(), isEmpty(): restituiscono rispettivamente la dimensione e true (false) se la la mappa M è non vuota (vuota)
- entrySet(): restituisci una collezione iterabile degli elementi di M
- keySet(): restituisci una collezione iterabile delle chiave in M
- values(): restituisci una collezione iterabile dei valori in M (con eventuali duplicati se un valore è associato a più chiavi)

Esempio

Operazion	Output	Mappa
isEmpty()	**true**	$\emptyset$
put(5,*A*)	**null**	(5,*A*)
put(7,*B*)	**null**	(5,*A*),(7,*B*)
put(2,*C*)	**null**	(5,*A*),(7,*B*),(2,*C*)
put(8,*D*)	**null**	(5,*A*),(7,*B*),(2,*C*),(8,*D*)
put(2,*E*)	*C*	(5,*A*),(7,*B*),(2,*E*),(8,*D*)
get(7)	*B*	(5,*A*),(7,*B*),(2,*E*),(8,*D*)
get(4)	**null**	(5,*A*),(7,*B*),(2,*E*),(8,*D*)
get(2)	*E*	(5,*A*),(7,*B*),(2,*E*),(8,*D*)
size()	4	(5,*A*),(7,*B*),(2,*E*),(8,*D*)
remove(5)	*A*	(7,*B*),(2,*E*),(8,*D*)
remove(2)	*E*	(7,*B*),(8,*D*)
get(2)	**null**	(7,*B*),(8,*D*)
isEmpty()	**false**	(7,*B*),(8,*D*)

Realizzazione con una lista

- Utilizzo di una lista non ordinata
  - Memorizza gli elementi in una lista S (ad esempio una lista doppia – doubly-linked list), in ordine arbitrario
  - Uso di una variabile ausiliaria n per memorizzare il numero di elementi



Algoritmo get(k)

```
Algorithm get(k):
    B = S.positions() {B è iteratore sulle posizioni di S}
    while B.hasNext() do
        p = B.next()     {posizione successiva in  B }
        if p.element().getKey() = k    then
            return p.element().getValue()
    return null {non esiste un elemento con chiave k}
```

Algortimo put(k,v)

```
Algorithm put(k,v):
B = S.positions()
while B.hasNext() do
    p = B.next()
    if p.element().getKey() = k  then
        t = p.element().getValue()
        S.set(p,(k,v))
        return t {ritorna il valore precedente}
S.addLast((k,v))
n = n + 1  {incrementa la variabile che
    memorizza il numero di elementi}
return null   {non ci sono elementi con chiave
    uguale a  k }
```

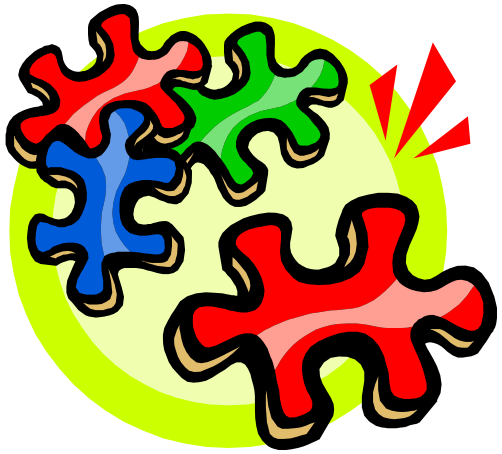
Algoritmo remove(k)

```
Algorithm remove(k):
B =S.positions()
while B.hasNext() do
    p = B.next()
    if p.element().getKey() = k  then
        t = p.element().getValue()
        S.remove(p)
        n = n – 1    {decrementa il numero di
    elementi}
        return t {restituisce il valore rimosso}
return null    {non ci sono elementi con chiave
    uguale a k }
```

Prestazioni di Map con Liste

- Prestazioni:
  - Put richiede tempo $O(n)$ poiché possiamo inserire un nuovo elemento all'inizio o alla fine della sequenza
  - get e remove richiede tempo $O(n)$ poiché nel caso peggiore (elemento cercato non è presente) si attraversa l'intera sequenza alla ricerca dell'elemento
- La lista non ordinata è efficiente solo per mappe di piccole dimensioni o per mappe in cui le operazioni più costose (ricerca – search e rimuovi – removal) sono eseguite raramente (ad esempio nel caso di una mappa per memorizzare la storia dei login ad un elaboratore)

Insiemi e Multimappe



Definizioni

- Un **insieme** è una lista non ordinata di elementi senza duplicati con l'obiettivo di realizzare efficientemente l'operazione di appartenenza
  - Elementi di un insieme sono come le chiavi di una mappa ma senza valori associati alla chiave
- Un **multiinsime** è un insieme che permette duplicati
- Una **multimappa** è simile a una mappa tradizionale (poiché associa valori a chiavi); però in una multimappa la stessa chiave può essere associata a molteplici valori
  - Per esempio, l'indice analitico di un libro mappa un termine ad uan o più punti del libro in cui quel termine occorre)

Struttura dati astratta Insieme

add(e): Adds the element e to S (if not already present).

remove(e): Removes the element e from S (if it is present).

contains(e): Returns whether e is an element of S.

iterator( ): Returns an iterator of the elements of S.

There is also support for the traditional mathematical set operations of ***union***, ***intersection***, and ***subtraction*** of two sets S and T:

$$\begin{aligned} S \cup T &= \{e:\ e \text{ is in } S \text{ or } e \text{ is in } T\}, \\ S \cap T &= \{e:\ e \text{ is in } S \text{ and } e \text{ is in } T\}, \\ S - T &= \{e:\ e \text{ is in } S \text{ and } e \text{ is not in } T\}. \end{aligned}$$

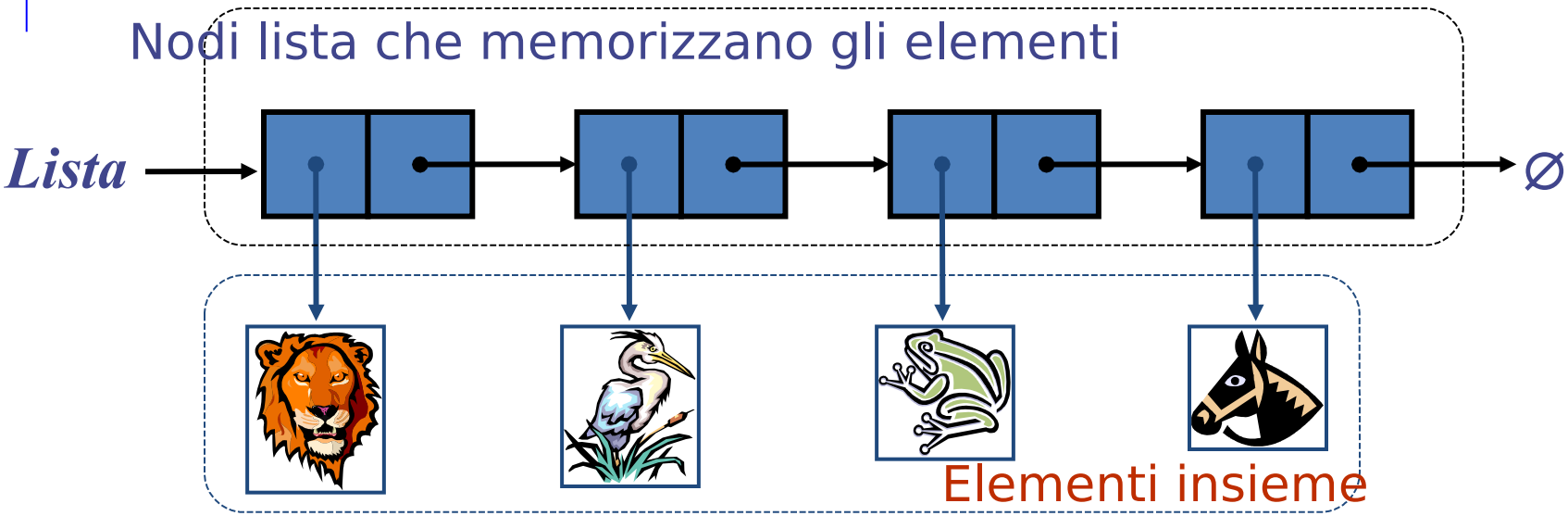
addAll(T): Updates S to also include all elements of set T, effectively replacing S by $S \cup T$.

retainAll(T): Updates S so that it only keeps those elements that are also elements of set T, effectively replacing S by $S \cap T$.

removeAll(T): Updates S by removing any of its elements that also occur in set T, effectively replacing S by $S - T$.

Memorizzazione con una Lista

- Possiamo implementare insiemi con una lista
- Elementi sono memorizzati secondo un ordine canonico
- Spazio usato è $O(n)$



Fusione Generica

- Fusione di due liste ordinate A e B
- Metodo genericMerge
- Metodi ausiliari
  - aIsLess
  - bIsLess
  - bothAreEqual
- Tempo esecuzione $O(n_A + n_B)$ assumendo che i metodi ausiliari abbiano tempo di esecuzione $O(1)$

```
Algoritmo genericMerge(A, B)
    S ← sequenza vuota
while ¬A.isEmpty() ∧ ¬B.isEmpty()
    a ← A.first().element(); b ←
    B.first().element()
        if a < b
    aIsLess(a, S); A.remove(A.first())
        else if b < a
    bIsLess(b, S); B.remove(B.first())
        else { b = a }
    bothAreEqual(a, b, S)
A.remove(A.first()); B.remove(B.first())
 while ¬A.isEmpty()
 aIsLess(a, S); A.remove(A.first())
    while ¬B.isEmpty()
 bIsLess(b, S); B.remove(B.first())
    return S
```

Utilizzo di Fusione Generica per altre Operazioni

- Una qualunque operazione può essere realizzata utilizzando Fusione generica
- Ad esempio:
  - intersezione: memorizza una sola copia degli elenti presente in ambedue le liste
  - Unione: copia ciascun elemento da ambedue le liste eccetto I duplicati
- Tutte le operazioni richiedono tempo lineare

Multimappe

- Una **multimappa** è simile a una mappa eccetto che permette di memorizzare più elementi con la stessa chiave
- Possiamo implementare una multimappa M con una mappa M'
  - Per ciascuna chiave k in M sia E(k) l'insieme degli elementi con chiave k
  - Gli elementi di M' sono le coppie (k, E(k))

Multimappe

get(k):	Returns a collection of all values associated with key k in the multimap.
put(k, v):	Adds a new entry to the multimap associating key k with value v, without overwriting any existing mappings for key k.
remove(k, v):	Removes an entry mapping key k to value v from the multimap (if one exists).
removeAll(k):	Removes all entries having key equal to k from the multimap.
size( ):	Returns the number of entries of the multiset (including multiple associations).
entries( ):	Returns a collection of all entries in the multimap.
keys( ):	Returns a collection of keys for all entries in the multimap (including duplicates for keys with multiple bindings).
keySet( ):	Returns a nonduplicative collection of keys in the multimap.
values( ):	Returns a collection of values for all entries in the multimap.