

## Esercitazione n.3

---

Si consideri il tipo di dato lista di liste in cui:

- la lista principale memorizza elementi detti “chiavi” e viene detta appunto **lista delle chiavi**. Tale lista non ammette duplicazioni e non è necessariamente ordinata;
- ogni chiave ha associata una lista concatenata di elementi detta appunto **lista degli elementi**. Tale lista ammette duplicazioni e non è necessariamente ordinata.

Le operazioni definite sono:

- **insert(L,k,e)**: inserisce nella lista di liste L una coppia (k,e)
- **remove(L,k,e)**: rimuove dalla lista di liste L l'elemento e dalla lista associata alla chiave k
- **contains(L,k,e)**: ritorna true se la lista di liste L contiene la coppia (k,e)

## Esercitazione n.3

---

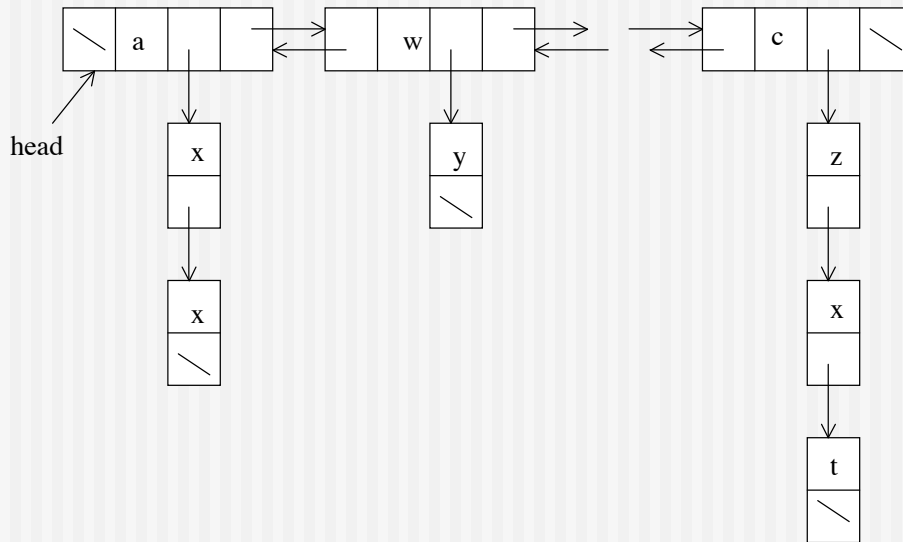
Si vuole realizzare il tipo di dato lista di liste mediante liste concatenate. In particolare:

- la **lista delle chiavi** è una lista doppia;
- la **lista degli elementi** è una lista semplice.

Si fa inoltre l'ulteriore ipotesi che non possa esserci una chiave senza alcun elemento associato, ovvero la lista degli elementi non può essere vuota.

## Esercitazione n.3

**Esempio:** le coppie (a,x) (c,z) (w,y) (c,t) (c,x) (a,x) vengono rappresentate dalla lista di liste:



## Esercitazione n.3

Il package LDL che realizza le liste di liste comprende:

- l'interfaccia LDL.java che dichiara tutte le operazioni della lista di liste
- la classe LKnodo.java per memorizzare una chiave
- la classe LEnodo.java per memorizzare un elemento
- la classe ListaDiListe.java che implementa tutte le operazioni dell'interfaccia LDL.java

Si richiede di implementare *in modo efficiente* la classe **ListaDiListe** e di consegnare:

- la classe ListaDiListe.java;
- la complessità dei metodi di ListaDiListe

**Consegna entro lunedì 10 Novembre ore 23.55**

# Interfaccia LDL.java (1)

```
package LDL;
public interface LDL {

    // post: ritorna true sse la lista di liste e' vuota
    public boolean isEmpty();

    // post: svuota la lista di liste
    public void clear();

    // post: ritorna il numero delle chiavi nella lista delle chiavi
    public int sizeKey();

    // pre: key non nullo
    // post: ritorna il numero di elementi della lista associata alla
    // chiave key; ritorna zero se key non e' presente nella
    // lista delle chiavi
    public int sizeElem(Object key);
```

# Interfaccia LDL.java (2)

```
// pre: key, at non nulli
// post: inserisce la coppia (key, at) nella lista di liste. Precisamente, se
// key e' gia' presente nella lista delle chiavi, aggiunge at alla
// lista degli elementi associati (in testa). Se key non e' presente,
// aggiunge un nodo per key nella lista delle chiavi (in testa) e un
// nodo per at nella corrispondente lista degli elementi (sempre in
// testa)
public void insert(Object key, Object at);

// pre: key, at non nulli
// post: cancella la prima occorrenza di at nella lista associata a key.
// Se la lista di elementi di key diventa vuota, cancella anche key
// dalla lista delle chiavi.
// Ritorna true se l'operazione e' andata a buon fine; false altrimenti
public boolean delete(Object key, Object at);

// pre: key non nulla
// post: cancella key dalla lista delle chiavi (e tutti gli elementi
// associati). Ritorna true se l'operazione e' andata a buon fine;
// ritorna false altrimenti
public boolean deleteKey(Object key);
```

## Interfaccia LDL.java (3)

```
// pre: at non nullo
// post: cancella dalla lista di liste TUTTE le occorrenze dell'elemento at
//       Cancellata anche tutte le chiavi che restano prive di elementi
public void deleteElem(Object at);

// pre: key, at non nulli
// post: ritorna true sse esiste un nodo della lista delle chiavi con
//       valore key e la cui lista degli elementi contiene at
public boolean search(Object key, Object at);

// pre: key non nullo
// post: ritorna true sse esiste un nodo della lista delle chiavi
//       con valore key
public boolean searchKey(Object key);

// pre: at non nullo
// post: ritorna true sse esiste un nodo nelle liste degli elementi
//       con valore at
public boolean searchElem(Object at);

// post: ritorna una stringa che rappresenta la lista di liste
//       come coppie (chiave, elemento) separate da uno spazio
public String toString();
}
```

## Classe LKnodo.java

```
package LDL;
class LKnodo {
    Object key;           // valore memorizzato nell'elemento
    LEnodo list;         // riferimento alla lista di attributi associata
    LKnodo next;         // riferimento al prossimo elemento
    LKnodo prev;         // riferimento all'elemento precedente

    // pre: ob diverso da null
    // post: costruisce un nuovo elemento con valore ob e lista l;
    //       connette il record in base a nextel e prevel
    LKnodo(Object ob, LEnodo l, LKnodo nextel, LKnodo prevel) {
        key = ob;
        list = l;
        next = nextel;
        if (next != null)
            next.prev = this;
        prev = prevel;
        if (prev != null)
            prev.next = this;
    }

    // pre: ob diverso da null
    // post: costruisce un record isolato con valore ob e lista nulla
    LKnodo(Object ob) { this(ob,null,null,null); }
}
```

# Classe LEnodo.java

```
package LDL;
class LEnodo {
    Object elem;           // elemibuto memorizzato nell'elemento
    LEnodo next;         // riferimento al prossimo elemento

    // pre: ob diverso da null
    // post: costruisce un nuovo nodo con valore ob e prossimo
    //        elemento nextel
    LEnodo(Object ob, LEnodo nextel) {
        elem = ob;
        next= nextel;
    }

    // pre: ob diverso da null
    // post: costruisce un nuovo nodo isolato con valore ob
    LEnodo(Object ob) { this(ob, null); }
}
}
```

# Classe ListaDiListe.java

```
package LDL;
public class ListaDiListe implements LDL {
    private LKnodo head = null; // riferimento alla lista di liste
    private int countKey = 0;   // totale chiavi memorizzate

    // implementare tutti i metodi dell'interfaccia LDL.java
    ...
    ...
}
}
```