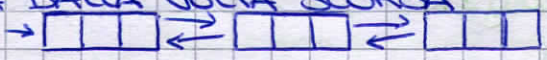
RIPRESA DALLA VOLTA SCORSA



6.11.13

```
search (List L, Elem k)
    x = L.head
    while x ≠ NULL AND x.key ≠ k
        x = x.next
    return x
```

Inv = gli elementi da L.head a x non compreso hanno chiavi ≠ k

(1) inizializzazioni
(2) conservazione — Inv ∧ Guardia → Inv (dopo l'esecuzione del corpo del ciclo)
(3) conclusione — Inv ∧ ¬Guardia → (asserzione finale)

N.B! Il ciclo termina per due ragioni:
(1) x = NULL in tal caso l'invariante assicura che k non è presente in tutta la lista
(2) x.key = k l'invariante assicura che k non è presente prima di x dunque x è la prima occorrenza dell'elemento con chiave k


```
insert (List L, Node x)
    x.next = L.head
    if L.head ≠ NULL
        L.head.prev = x
        L.head = x
        x.prev = NULL
```

```
delete (List L, Node x)
  /* Pre: x ∈ L */
    if x.prev ≠ NULL
        x.prev.next = x.next
    else
        L.head = x.next
        if x.next ≠ NULL
            x.next.prev = x.prev
rimuovi x
```

→ Distinguo tre casi
1) è testa
2) è coda
3) è in mezzo

REALIZZAZIONE CON LISTE DOPPIE E SENTINELLA

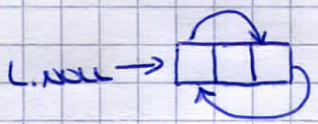
Sentinella: oggetto fittizio che ha tutti i campi degli altri elementi della lista

L.NULL rappresenta NULL

L.head è eliminata e sostituita con L.NULL.next, e L.NULL.prev punta alla coda

Una lista vuota è formata solo dalla sentinella.

La sentinella rende il codice più compatto.



```
search (List L, Elem k)
    x = L.NULL.next
    while x ≠ L.NULL AND x.key ≠ k
        x = x.next
    return x
```

L.NULL.key = k

Semplifica una delle condizioni con l'elemento fittizio L.NULL che esiste sempre

```
insert(List L, Node x)
    x.next = L.NULL.next
    L.NULL.next.prev = x
    L.NULL.next = x
    x.prev = L.NULL

delete (List L, Node x)
    x.prev.next = x.next
    x.next.prev = x.prev
rimuovi (x)

struct node{
        int info;
        struct node *next;
        struct node *prev;
}
typedef struct node *List;

List creaLista (){
        List e;
        e = malloc(sizeof (struct node));
        e->next = e;
        e->prev = e;
        return e;
```

UNION per gli insiemi dinamici richiede due insiemi DISGIUNTI S1 e S2 come input restituisce un insieme $S = S1 \cup S2$, che è formato da tutti gli elementi di S1 e S2

↳ COSTANTE, uso un puntatore alla testa e un puntatore alla coda ($\Theta(1)$)

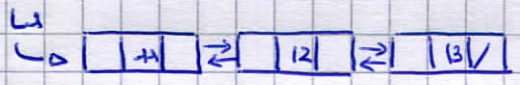
```
Union (List S1, List S2)
 if S1.head == NULL
        return S2
 else
    if S2.head == NULL
        return S1
    else
        S1.tail.next = S2.head
        S2.tail = S2.tail
```

Ho concatenato le due liste

Rappresentazione con più array

indice non presente nel nostro vettore

	1	2	3	4	5	6	7	8
next	0	0			3		6	
key		13			12		11	
prev	0	6			8			



La costante NULL viene rappresentata con l'indice $\emptyset$ (per lo pseudocodice) - 1 (in C).

freeLista → tutte le parti vuote

free: list concatenate singolarmente in cui mantengo gli oggetti liberi. È una variabile globale.

```
Allocate - Object ()
    if free = NULL
        errore "spazio terminato"
    else
        x = free
        free = next [x]
        return x
```

$\Theta(1)$ perché sto facendo degli assegnamenti

```
Free - Object (x)
    next [x] = free
    free = x
```

A = ARCHI

ASD 7.11.2013

Albero (RADICATO) è una coppia $T=(N,A)$. N è un insieme finito di nodi fra cui si distingue un nodo r detto RADICE

- $A \subseteq N \times N$ è un insieme di coppie di nodi, detti ARCHI

In un albero, ogni nodo v (eccetto la radice) ha esattamente un genitore (o padre) u tale che $(u,v) \in A$

Albero $\leftrightarrow$ GRAFO CONNESSO ACICLICO (NON) ORIENTATO

Un nodo v può avere $\emptyset$ o più figli v tali che $(u,v) \in A$
Il numero di figli di un nodo è detto grado di un nodo. Un nodo senza figli è detto foglia e un nodo non foglia (cioè che ha dei figli) è un nodo interno.

Se due nodi hanno lo stesso padre sono fratelli.

Cammino da un ~~nodo~~ nodo u a un nodo u' in T è una sequenza di nodi $\langle n_0, n_1, \dots, n_k \rangle$ tali che soddisfa le seg. proprietà

- 1) $u = n_0$
  2) $u' = n_k$
  3) $\langle n_{i-1}, n_i \rangle \in A$ per $i = 1 \dots k$

La lunghezza di un cammino è il numero di archi oppure il numero di nodi che formano il cammino - 1

Sia x un nodo in un albero radicato T con radice r. Un qualsiasi y in un cammino da r a x è detto antenato di x. [anche il nodo stesso è antenato]

Se y è antenato di x, allora x è discendente di y

N.B. Ogni nodo è antenato e discendente di se stesso.

Se y è un antenato di x e $x \neq y$ allora y è un antenato proprio di x e x è un discendente PROPRIO di y.

- Il sottoalbero con radice in x è l'albero indotto dai discendenti di x con radice in x

La PROFONDITÀ di un nodo x è la lunghezza del cammino dalla radice a x

Un livello di un albero è costituito da tutti i nodi che stanno alla stessa profondità

L'altezza di un nodo x è la lunghezza del più lungo cammino che scende da x a una foglia

L'altezza di un albero è l'altezza della sua radice $\rightarrow$ Profondità massima di un nodo qualsiasi dell'albero

Albero binario $\rightarrow$ di natura ricorsiva

- un albero vuoto è un albero binario
- un albero costituito da un ① nodo radice, ② un albero binario detto sottoalbero sinistro della radice, e ③ da un albero ancora binario detto sottoalbero destro della radice, è un albero binario

Un albero k-ario è un albero in cui i figli di un nodo sono etichettati con interi positivi distinti e le etichette maggiori di k sono assenti

Un albero binario è un albero k-ario con $k=2$.

Un albero k-ario completo è un albero k-ario in cui tutte le foglie hanno la stessa profondità e tutti i nodi interni hanno grado k.

ESERCIZIO: Trovare il numero di foglie e di nodi interni di un albero k-ario completo di altezza h.

k^h — Perché la radice ha k figli che hanno k figli tutto per h volte

$\#\text{foglie}(h) = k^h \qquad h = 0$

$k^0 = 1$ l'albero è costituito dalla sola radice che è l'unica foglia

Assumiamo che per un albero di altezza h sia vero $\#\text{foglie}(h) = k^h$ e lo dimostro per alberi di altezza $h+1$.

Il numero di nodi di profondità h sono k^h per Ip. induttiva in quanto foglie di un albero di altezza h.

Poiché l'albero è completo, ognuno di questi nodi ha esattamente k figli:

$k^h \cdot k = k^{h+1}$ numero di foglie (C.V.D)

$\#\text{foglie}(h+1) = k^{h+1}$

- Numero nodi interni h

$$\sum_{i=0}^{h-1} k^i = \frac{k^h - 1}{k-1} = \frac{k^h - 1}{k-1}$$

ESERCIZIO: Trovare l'altezza di un albero k-ario completo con n foglie

Le foglie di un albero completo di altezza h

$n = k^h \qquad h = \log_k n$

$n = \frac{k^{h+1} - 1}{k-1}$ ⇝ COSÌ POSSO RICAVARE h

Tipo: Albero
dati: un insieme di nodi (di tipo Node) e un insieme di archi
operazioni:
newTree()
Post: restituisce un albero vuoto
treeEmpty(Tree t) → Bool
Post: restituisce true se è vuoto, false altrimenti
grado(Tree t, Node v) → int
Pre: v in t
Post: restituisce il numero di figli del nodo v.
numNode(Tree t) → int
Post: restituisce il numero di nodi dell'albero in t

padre(Tree t, Node v) -> Node
Pre : v in t
Post : restituisce il padre del nodo v nell'albero t oppure NULL se v è la radice

figli(Tree t, Node v) -> List of Node
Pre : v in t
Post : restituisce una lista contenente i figli di v

aggiungiNodo(Tree t, Node u, Elem k) -> Node
Pre: se il nodo u ≠ NULL allora u in t, altrimenti t è vuoto
Post : inserisce un nuovo nodo v con chiave k come figlio di u se u ≠ NULL, altrimenti v è la radice dell'albero e restituisce v

aggiungi sottoAlbero(Tree t, Node u, Tree tu)
Pre: u in t
Post: inserisce nell'albero il sottoalbero tu in modo che la radice di tu diventi figlia di u

rimuovi SottoAlbero(Tree t, Node u) -> Tree
Pre : u in t
Post: stacca e restituisce l'intero albero radicato in v. L'operazione cancella dall'albero t il nodo v e tutti i suoi discendenti