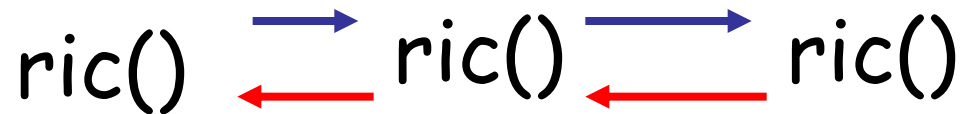


liste

concatenate

ricordare che nella ricorsione:

andata

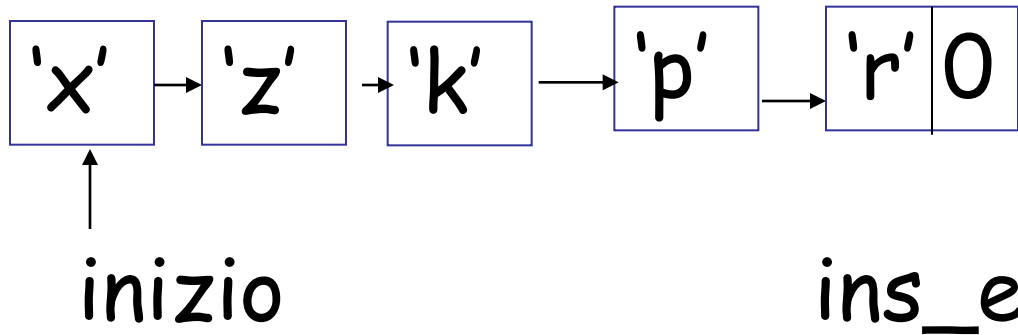


ritorno

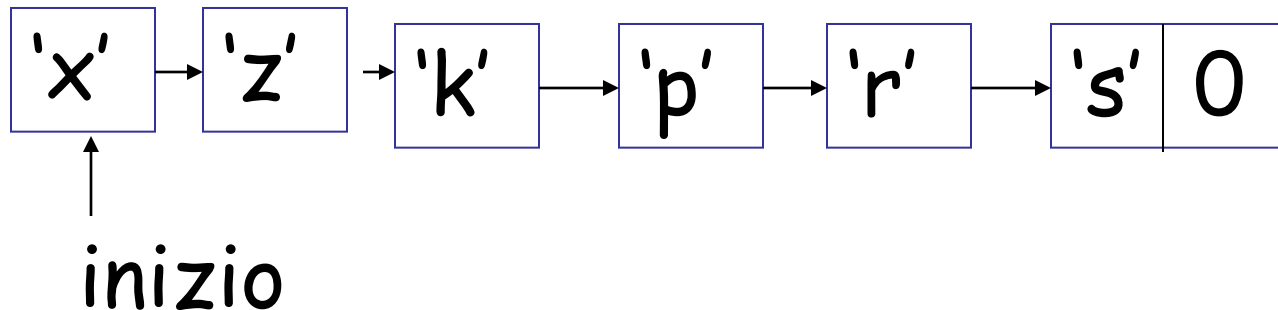
calcolare all'andata e/o al ritorno

se non si fa nulla al ritorno allora ricorsione  
terminale → facile trasformarla in WHILE

# inserire un elemento alla fine della lista



`ins_end(inizio, 's')`

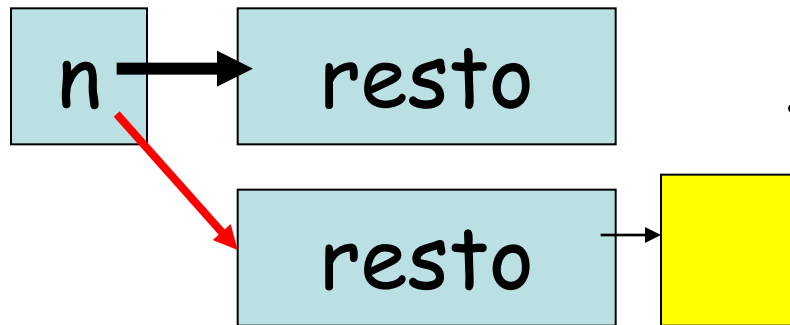


## I soluzione

**caso base:** lista vuota

*si tratta di creare il nuovo nodo e restituirlo*

**caso ricorsivo:** lista con un nodo almeno

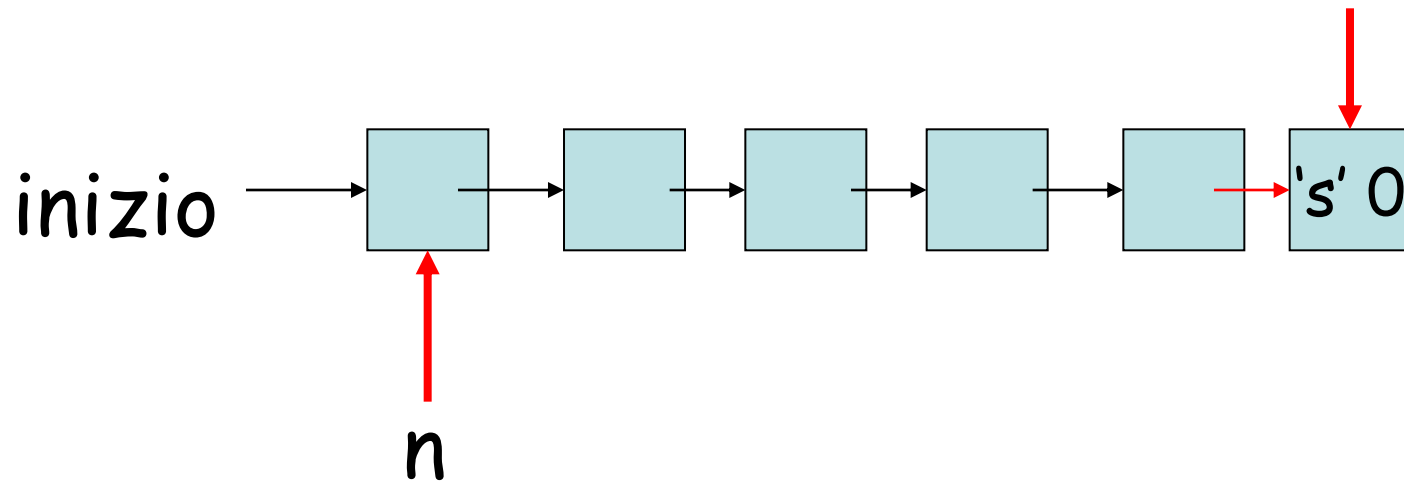


soluzione del resto

## Realizzazione:

.all'andata troviamo la fine della lista

.al ritorno costruiamo la lista allungata collegando ogni nodo con il nuovo resto



nella funzione che stiamo per  
descrivere


usiamo il fatto che :

puntatore == 0 = false

puntatore != 0 = true

```
nodo * ins_end(nodo *n, char c)
{ if(! n)
    {nodo *x=new nodo(c,0); return x;}
  else
    {n→next=ins_end(n→next,c); return n; }
}
```

fine  
ricorsione



invocazione:  
**inizio**=ins\_end(inizio,'s');

invocazioni  
intermedie



PRE=(n punta ad una lista event. vuota)

nodo \* ins\_end(nodo \*n, char c)

{ if(! n)

{nodo \*x=new nodo(c,0); return x;}

else

{n→next=ins\_end(n→next,c); return n; } }

POST=(restituisce n@nodo(c,0))



## prova induttiva

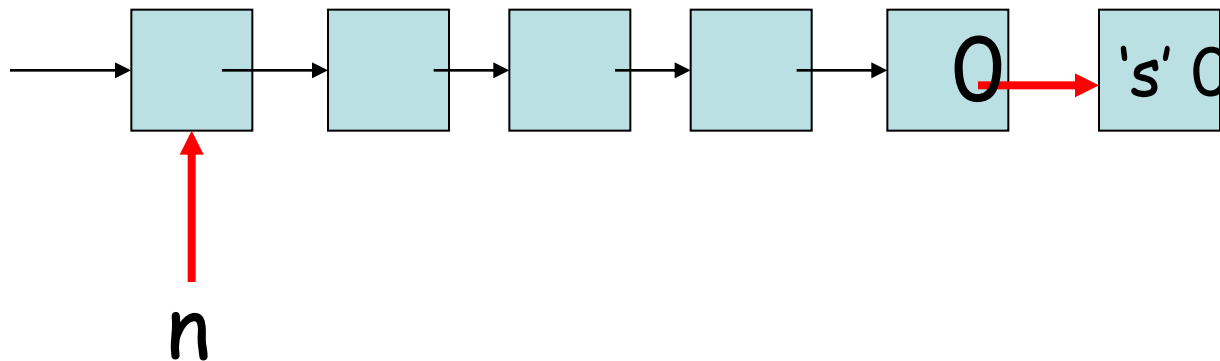
**base**=n è vuota  $n@nodo(c,0)$  è  $nodo(c,0)$

**passo ric**:  $n=a@n'$ ,

PRE\_ric vale  $\Rightarrow$   $ins\_end(n \rightarrow next, c)$   
restituisce  $n'@nodo(c,0)$

restituiamo  $a@n'@nodo(c,0)=n@nodo(c,0)$   
 $\Rightarrow$  POST

**II soluzione:** facciamo tutto all'andata,  
significa che la funzione restituisce void  
e quindi non possiamo modificare **inizio**  
che punta al primo nodo  
funziona solo per liste non vuote



```
void ins_end(nodo *n, char c)
{ if( ! n→next)
    n→next=new nodo(c,0);
else
    ins_end(n→next,c);
}
```

da chiamare  
solo con  $n \neq 0$

```
if(inizio) ins_end(inizio, 's');
else inizio=new nodo('s',0);
```

PRE=(n lista non vuota)

```
void ins_end(nodo *n, char c)
{ if( ! n→next)
    n→next=new nodo(c,0);
  else
    ins_end(n→next,c);
}
```

POST=(costruisce n@nodo(c,0))

**base:**  $n$  consiste di un solo nodo, la funzione costruisce  $n @ \text{nodo}(c, 0)$

**passo ric:**  $n = a @ a' @ n'$

PRE\_ric vale, cioè  $a' @ n'$  non è vuota

vale POST\_ric, cioè

$\text{ins\_end}(n \rightarrow \text{next}, c)$  costruisce

$a' @ n' @ \text{nodo}(c, 0)$

e quindi adesso abbiamo

$a @ a' @ n' @ \text{nodo}(c, 0)$

riassumiamo:

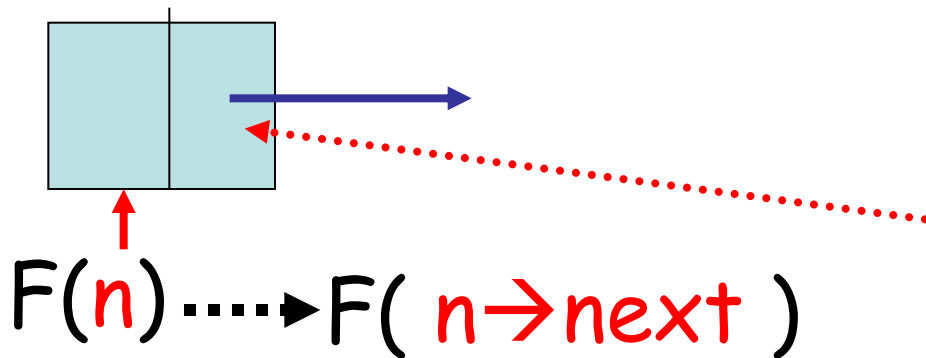
**soluzione I** : all'andata si passa l'ultimo nodo ( $n == 0$ ) e poi si costruisce la nuova lista al ritorno

**soluzione II**: all'andata ci si ferma all'ultimo nodo ( $n \rightarrow next == 0$ ) e gli si attacca il nuovo nodo. Non si fa nulla al ritorno

la II è più semplice, ma non gestisce il caso della lista vuota

vorremmo contemporaneamente  
poter modificare il campo next  
dell'ultimo nodo ma fermarci con  $n == 0$   
possiamo ottenerlo passando il nodo  
per riferimento

$F(\text{nodo} * \& n) \{ \dots F(n \rightarrow \text{next}) \dots \}$



nella prossima  
F,  $n$  è un alias  
di questo  
puntatore

ins ultimo nodo con pass. per riferimento

PRE=(n punta a lista L event. vuota)

void ins(nodo\*&n,int x)

{

if(!n)

n=new nodo(x,0);

else

ins(n->next,x);

} POST=(n punta ad L@nodo(c,n))

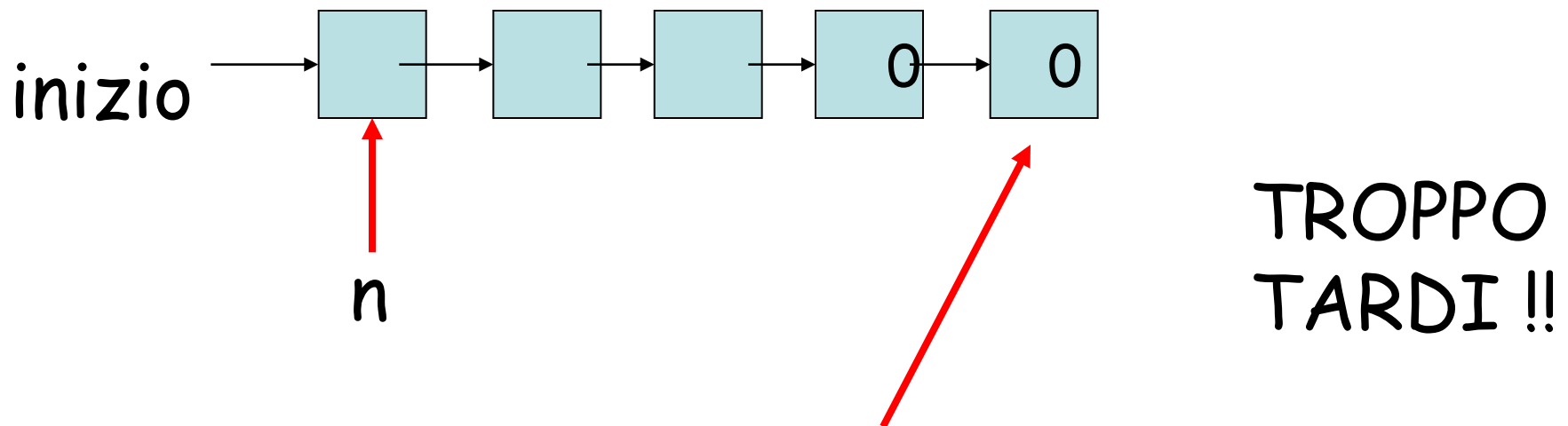


**base:**  $n=0 \Rightarrow L$  è vuota,  
 $n$  punta a  $\text{nodo}(c,0)$  cioè  $L@nodo(c,0)$

**passo ric:**  $n$  punta a  $L=a@L'$   
 $n \rightarrow \text{next}$  punta a lista  $L' \Rightarrow$  vale  $\text{PRE\_ric}$   
 $n \rightarrow \text{next}$  punta a  $L'@nodo(c,0)$   
 $n$  punta ad  $a@L'@nodo(c,0) = L@nodo(c,0)$

altro esempio: eliminare ultimo  
nodo di una lista

# soluzioni col passaggio di n per valore



ci dobbiamo fermare qui, con  $n \rightarrow next == 0$

buttare via questo nodo e restituire 0

## costruiamo la lista al ritorno

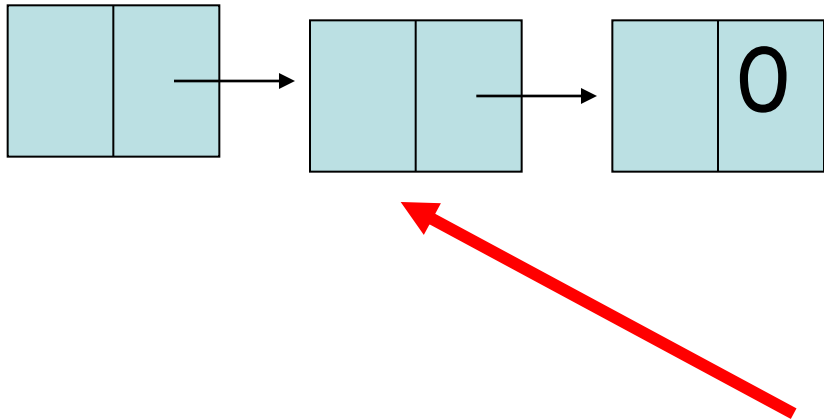
```
nodo * del_L(nodo *n)
{if( ! n→next )
{delete n; return 0; }
n→next=del_L(n→next);
return n;
}
```

invocazione:

```
if(inizio) inizio=del_L(inizio);
```

non ha senso se la lista è vuota

è possibile fare l'operazione solo all'andata ?  
a quale nodo deve fermarsi la ricorsione ?



poco generale  
BRUTTA !!

dobbiamo fermarci qui

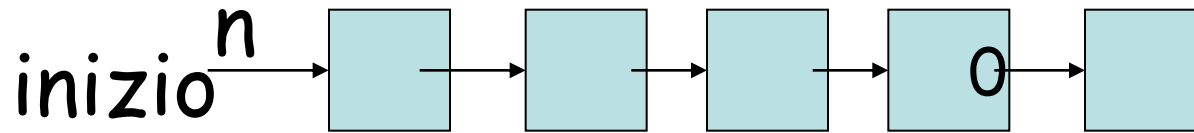
$n \rightarrow \text{next} \rightarrow \text{next} == 0$

funziona solo per liste con almeno 2 nodi !!!!!

eliminare ultimo nodo all'andata

```
void del_L(nodo *n)
{
    if( ! n->next->next )
    { delete n->next;
      n->next=0; }
    else
      del_L(n->next);
}
```

# eliminare ultimo nodo con passaggio per riferimento



```
void del_LR(nodo *& n)
{
    if(! n->next)
        delete n; n=0;
    else
        del_LR(n->next);
}
```

il passaggio per riferimento ci permette di arrivare all'ultimo nodo e di avere un alias del next del nodo precedente

invocazione: `if(inizio) del_LR(inizio);`

## ricorsione e passaggio per riferimento:

```
void f(... int & x ....)
```

```
{
```

```
....f(...x...)...
```

```
}
```

tutte le invocazioni di f condividono la variabile x : le modifiche di x si ripercuotono su tutte le invocazioni



negli esercizi visti prima era diverso:

```
void ins(nodo * & n....)
```

```
{
```

```
....ins( n->next...)
```

```
}
```

## altro esercizio distruggere l'intera lista

```
void del_all(nodo *x)
{if(x)
{del_all(x→next);
delete x;
} }
```

**invocazione:**

```
del_all(inizio);
```

```
inizio=0;
```

l'ordine delle  
cose è  
**MOLTO**  
importante !!

eliminare tutti i nodi con info= y

```
nodo* del(nodo*L,int y)
{
    if(L)
        if(L->info==y)
            {nodo*x=L->next; delete L;
             return del(x,y);}
        else
            {L->next=del(L->next,y); return L;}
    else
        return NULL;
}
```

Versione con passaggio per riferimento?

ESERCIZIO

# esercizio

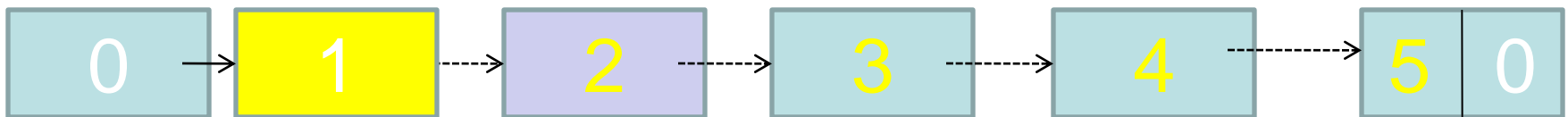
realizzare la cancellazione  
dell'ultimo nodo di una lista in  
modo iterativo

prima cosa da pensare: a quale nodo si  
deve fermare il ciclo ?

# esercizio



Inserire un nodo in posizione 1



Su quale nodo ci si deve fermare ?  
Se inseriamo in posizione  $k$ :

- sul nodo in posizione  $k$

- sul nodo precedente  $k-1$

conviene introdurre la seguente notazione:  
 $L(k)$  = lista che consiste dei primi  $k$  nodi di  $L$ , cioè dal nodo 0 al  $k-1$

il nodo finale di  $L(k)$  è in posizione  $k-1$

quindi se  $L = L(k)@R$ , fermarsi alla posizione  $k$  significa alla prima di  $R$   
fermarsi alla  $k-1$  significa all'ultima di  $L(k)$



-fermarsu sul nodo in posizione k:

PRE=(L corretta poss. vuota e  $k \geq 0$ )

nodo\* ins(nodo\*L, int k, int c)

{

if(!k)

return new nodo(c,L);

else

if(L)

{L->next=ins(L->next,k-1,c); return L;}

else

return 0;

} POST=(se  $L=L(k)@R$ , restituisce

$L(k)@nodo(c, \rightarrow)@R$ , altrimenti L)

-fermarsi sul nodo in posizione  $k-1$ :

$PRE = (L \text{ corretta e non vuota e } k > 0)$

void ins(nodo\*L, int k, int c)

{if( $k == 1$ )

$L \rightarrow next = \text{new nodo}(c, L \rightarrow next);$

else //  $k > 1$

    if( $L \rightarrow next$ )

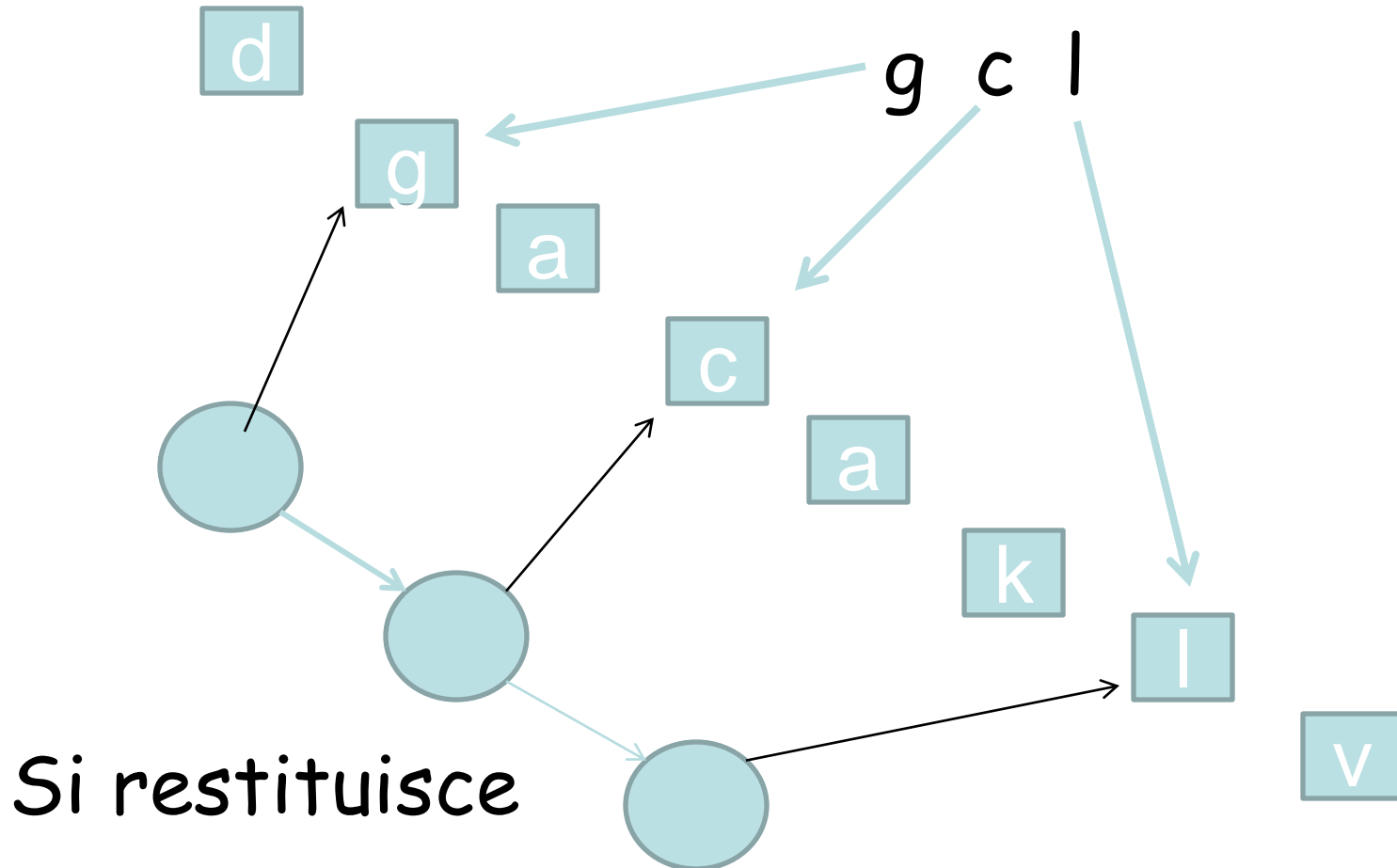
        ins( $L \rightarrow next$ ,  $k-1$ ,  $c$ );

}

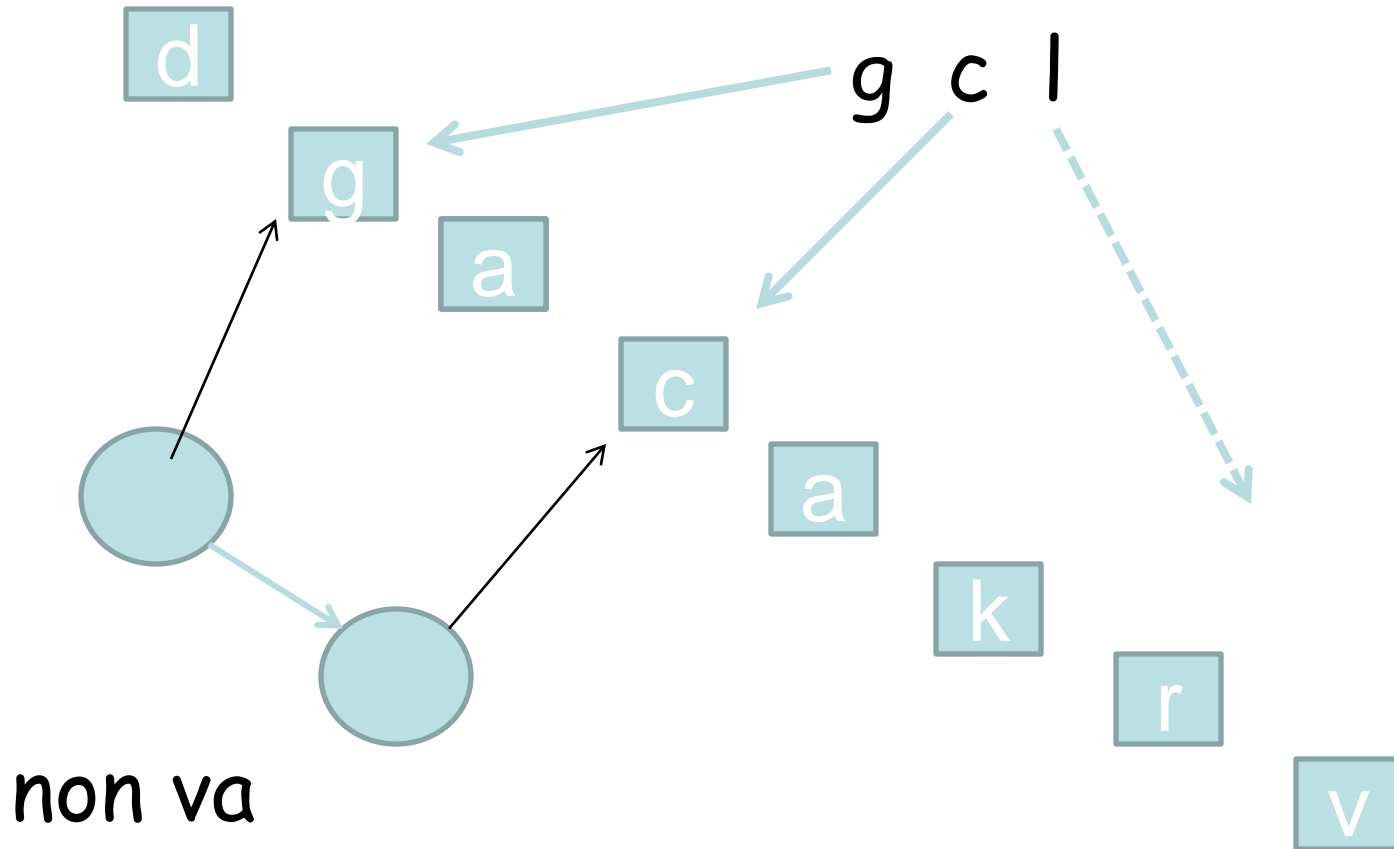
$POST = (\text{se } L = L(k) @ R, \text{ costruisce}$

$L(k) @ \text{nodo}(c, \rightarrow) @ R, \text{ altrimenti } L)$

# Pattern matching su una lista :



Ma potrebbe anche non funzionare



All'**andata** percorriamo la lista e  
facciamo i match possibili  
Al **ritorno** dobbiamo fare due cose:

- far sapere se match fallisce o no
- se c'è match allora si deve  
costruire la lista risultato

Così lo costruiamo solo se c'è match

3 possibili soluzioni:

Soluzione 1) Usiamo un bool passato per riferimento (visto che deve comunicare alle invocazioni che "stanno sopra" se è stato trovato match o no)

caso base match finito

soluzione 2:

Comunichiamo il  
successo/insuccesso con la  
stessa lista

- se ritorno 0 fallimento
- se ritorno != 0 successo

complica il caso base

soluzione 3: sfruttando le  
eccezioni in caso di fallimento  
del match → throw

variazioni dell'esercizio: match contigui

- calcolare numero di match contigui e completi esistenti (anche sovrapposti)
- n. match contigui e completi e non sovrapposti
- lunghezza massima dei match contigui ma possibilmente incompleti (anche sovrapposti)