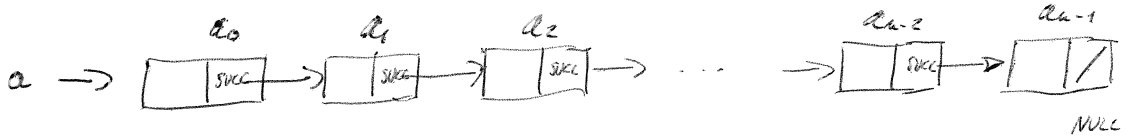
Liste

Sia dato un elemento x di una lista in posizione i

Indichiamo con $x.dato$ l'informazione associata a tale elemento e con $x.succ$ il riferimento all'elemento nella posizione $(i+1)$-esima.

Il valore NULL indica il riferimento a nessuna locazione.

Denoteremo con a il riferimento al primo elemento della lista



Come accedere all'elemento in posizione i?

```
p = a;
j = 0;
WHILE ( (p != NULL) && (j < i)){
    p = p.succ;
    j = j + 1;
}
```

Scorre la lista fino a raggiungere l'elemento i-esimo o la fine della lista

È facile modificare il codice di scorrimento per cercare una chiave K: basta modificare la condizione di terminazione

```
WHILE ( (p != NULL) && (p.dato != K))
```

A differenza degli array si prestano bene a gestire sequenze lineari dinamiche.

Inserimento e cancellazione

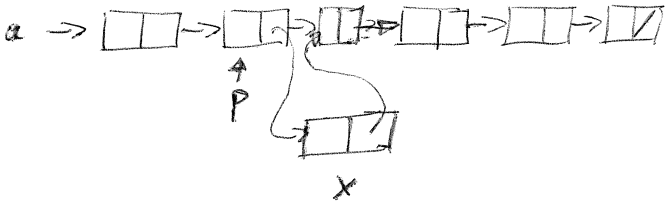
L'inserimento si realizza ponendo l'elemento in cima alla lista

```
x.succ = a;
a = x;
```



L'inserimento DOPO un elemento indicato da un riferimento p ≠ NULL è una semplice variazione: p.succ sostituisce la variabile a

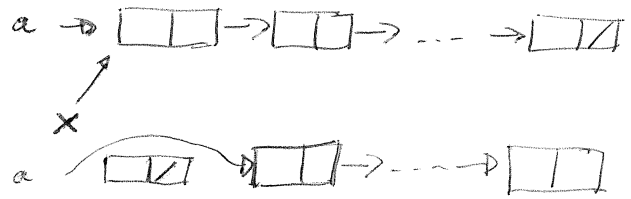
```
x.succ = p.succ;
p.succ = x
```



La cancellazione è leggermente più complicata. Avendo a disposizione il riferimento x all'elemento da cancellare si ha:

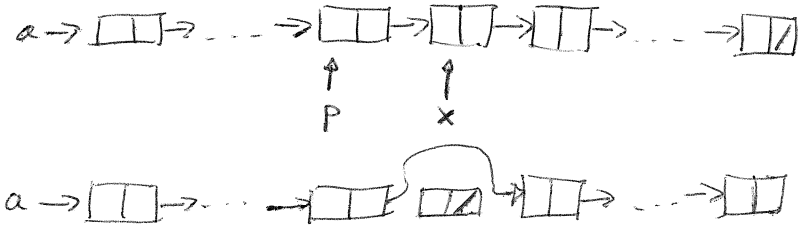
- se x coincide con a (primo elemento della lista)

```
a = x.succ
x.succ = NULL
```



- per elementi "nel mezzo" occorre anche il riferimento p all'elemento precedente

```
p.succ = x.succ
x.succ = NULL
```



Ignorando i costi di allocazione e deallocazione e per determinare i riferimenti p e x, le operazioni di inserimento e cancellazione hanno un costo pari a $O(1)$. Su un array hanno costo $\Theta(n)$.

Liste doppie

Per rendere più efficienti alcune operazioni, conviene modificare la struttura avendo "più riferimenti".

In una lista doppia, l'elemento x in posizione i, oltre al riferimento x.succ all'elemento in posizione i+1, ha un riferimento x.pred all'elemento in posizione i-1 (uguale a NULL se i = 0)

Pertanto, in tempo $O(1)$, ci si può muovere "avanti" e "indietro".



L'inserimento è leggermente più complicato.

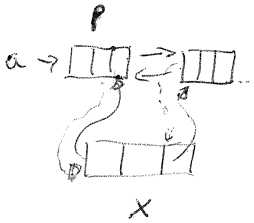
La cancellazione è semplificata. Non serve il riferimento precedente



INSERIMENTO

```
x.pred = x;
x.succ = a;
x.pred = NULL;
a = x;
```

(primo elemento)

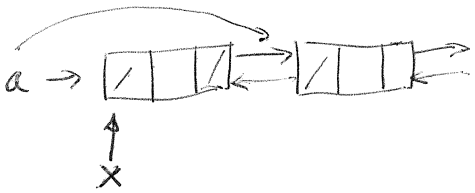



```
x.succ = p.succ;
x.pred = p;
p.succ.pred = x;
p.succ = x;
```

(DOPO un elemento indicato dal riferimento p)

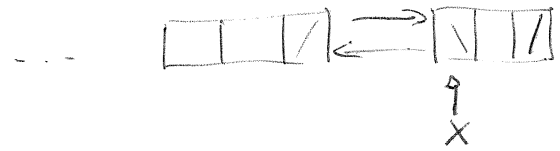
Detto x il riferimento all'elemento da cancellare si hanno 4 casi.

caso 1 ($x = a$) Primo elemento della lista




```
x.succ.pred = NULL
a = x.succ ;
x.succ = NULL;
```

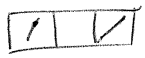
caso 2 (x fa riferimento all'ultimo elemento della lista)



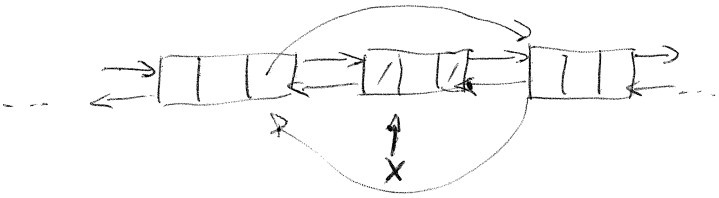

```
x.pred.succ = NULL ;
x.pred = NULL;
```

caso 3 (x è l'unico elemento della lista)

```
a = NULL
```



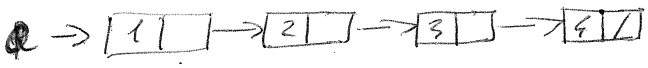
caso 4 (x fa riferimento a un elemento "interno" alla lista)




```
x.pred.succ = x.succ;
x.succ.pred = x.pred;
x.pred = NULL;
x.succ = NULL;
```

Tutti i casi garantiscono che x.succ = x.pred = NULL dopo una cancellazione. Non restano riferimenti pendenti (DANGLING POINTERS)

Versione iterativa

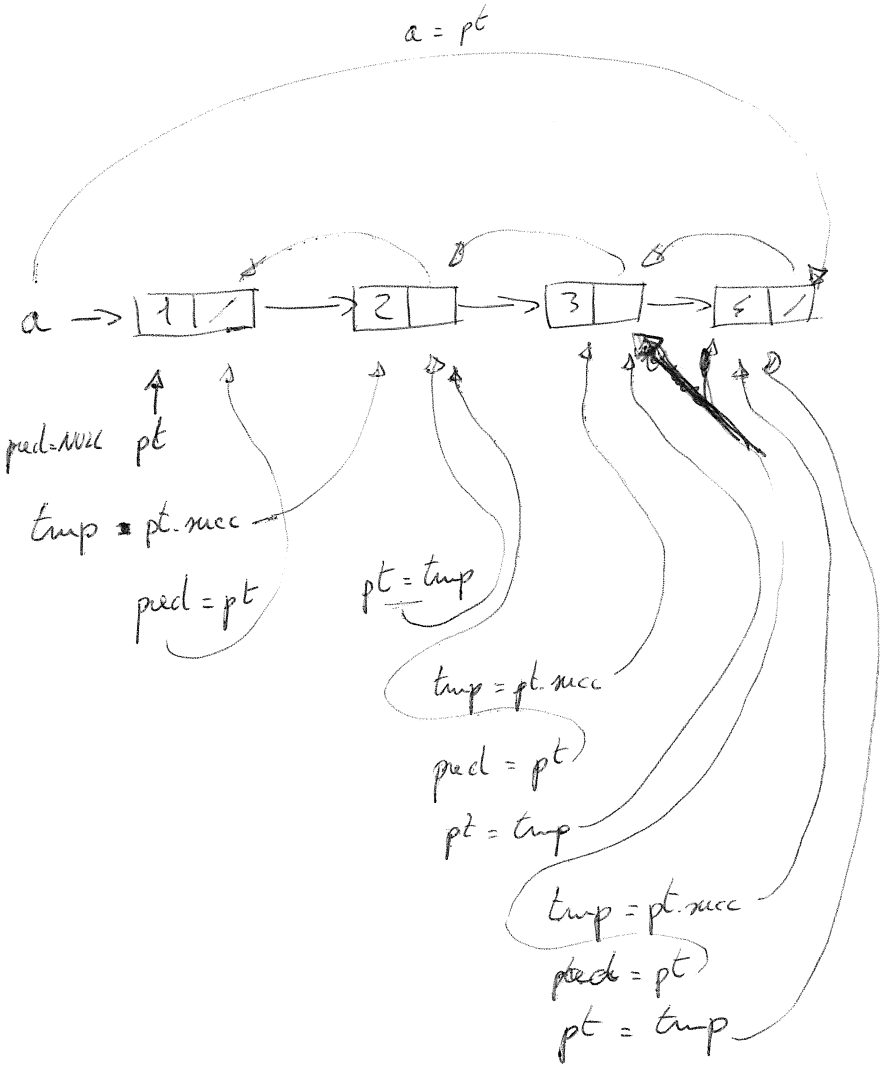



```
InvertiLista (p)
pred = NULL;
pt = p;
WHILE (pt.succ != NULL){
    tmp = pt.succ;
    pt.succ = pred;
    pred = pt;
    pt = tmp;
}
pt.succ = pred;
a = pt;
RETURN a;
```

InvertiLista (a)

```
pred = NULL;
pt = a;
```





I Garbage collector, nei linguaggi in cui sono presenti, recuperano le celle di memoria di elementi cancellati.

Esercizio. Si descriva un algoritmo che inverte una lista semplice. (La lista 1, 2, 3, 4 diventa 4, 3, 2, 1).

Soluzione.

```
InvertiLista (p, x)
  IF ( x. succ != NULL)
     a = InvertiLista (x, x. succ);
  ELSE
     a = x ;

  x. succ = p ;
  RETURN a ;
```



Invocazione iniziale: InvertiLista (NULL, a).

Caso semplice: a → 

Poiché a. succ = NULL risulta a = a; a. succ = NULL; e viene restituito a.

Caso "generale":

InvertiLista (NULL, a) → InvertiLista (a, a. succ) → InvertiLista (a. succ, a. succ. succ)
0 1 1 2 2 3

3. succ = 2 → (a = 4)

InvertiLista (3, 4)

Poiché 4. succ = NULL

(a = 4)

2. succ = 1 ← (a = 4)

1. succ = NULL ← (a = 4)

RETURN a

4. succ = 3

