Intelligenza Artificiale

Anno Accademico 2022 - 2023

Strutture Dati in Python:

Liste Concatenate

Norme di utilizzo dei materiali didattici

La visione e l'utilizzo del presente materiale didattico è riservato agli utenti iscritti al corso al solo fine di studio e approfondimento didattico.

L'utente che accede alla sezione e-learning e che ne consulta i contenuti è tenuto a rispettare le disposizioni legislative che tutelano il diritto d'autore e pertanto è fatto espresso divieto di riprodurre, pubblicare o distribuire i materiali tratti dal presente sito, anche in forma parziale, fatta salva la possibilità di realizzare un'unica copia del materiale, in formato cartaceo e/o digitale, all'esclusivo fine di studio.

L'utente è responsabile per l'uso improprio o non autorizzato del materiale pubblicato effettuato in violazione delle suddette disposizioni.

Sommario

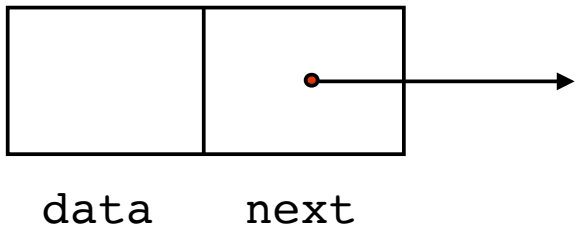
- Liste **Singolarmente Concatenate**
  - Creazione
  - Operazioni di inserimento, cancellazione, ricerca
- Liste **Doppiamente Concatenate**
  - Creazione
  - Operazioni di inserimento, cancellazione, ricerca

Liste singolarmente concatenate
Nodi, Liste e Operazioni

- Rappresentazione e creazione dei **nodi** di una lista concatenata
- Rappresentazione e creazione di una **lista**
- **Scansione** di una lista concatenata
- Operazioni di **Inserimento** di un nuovo elemento
- Operazioni di **cancellazione** di un elemento
- Operazioni di **ricerca** di un elemento

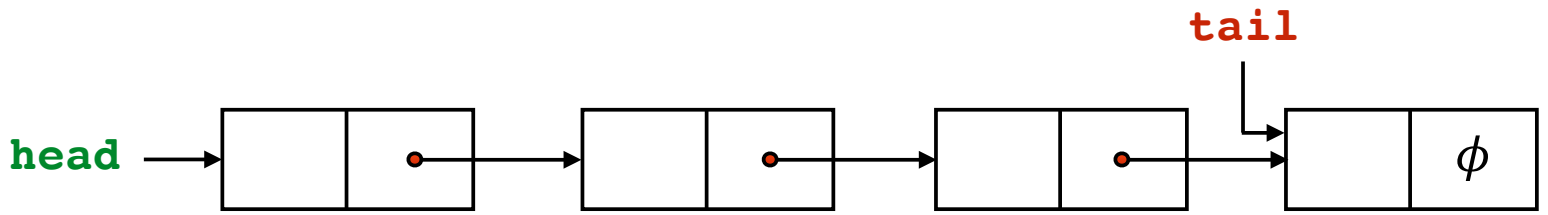
Liste singolarmente concatenate
Rappresentazione grafica nodo

In tali liste un nodo contiene, oltre al dato, un puntatore che punta al nodo successivo della lista:



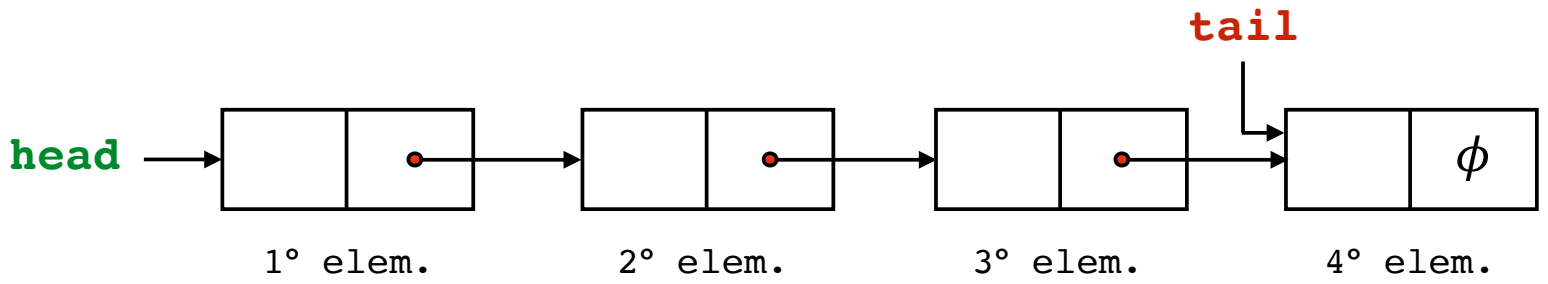
Liste singolarmente concatenate

Rappresentazione grafica lista



Liste singolarmente concatenate

Rappresentazione grafica lista



Liste singolarmente concatenate
Definizione della Classe Nodo

```
class Node:
    data = ''
    next = None

    def __init__(self, data, next):
        self.data = data
        self.next = next
```

Istanze di questa Classe sono oggetti **`Node`** con attributi **`data`** e **`next`** impostati dal costruttore con i valori dei parametri passati.

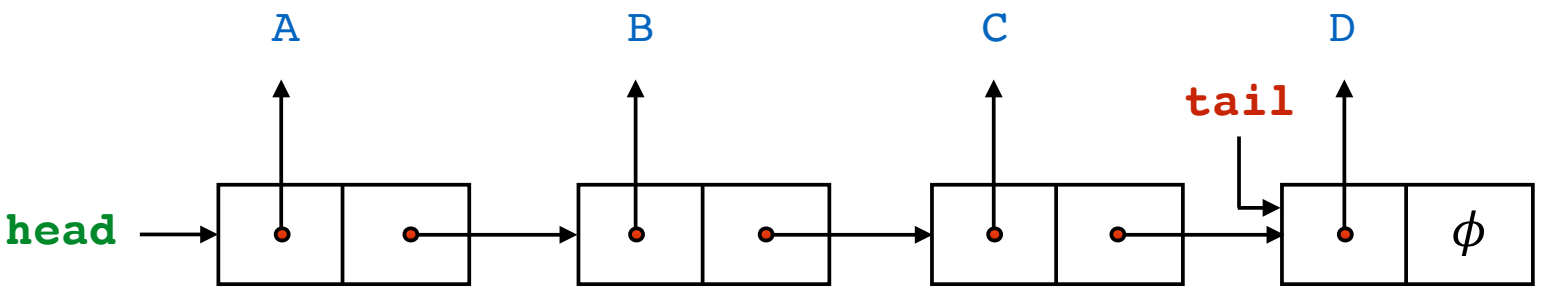
Liste singolarmente concatenate
Definizione della Classe Lista

```
class SinglyLinkedList:
#     __head = None
#     __tail = None

    def __init__(self):
        self.__head = None
        self.__tail = None
```

Istanze di questa Classe sono oggetti **`SinglyLinkedList`** con i puntatori **`head`** e **`tail`** impostati a **`None`**. Sono quindi inizializzati a liste vuote.

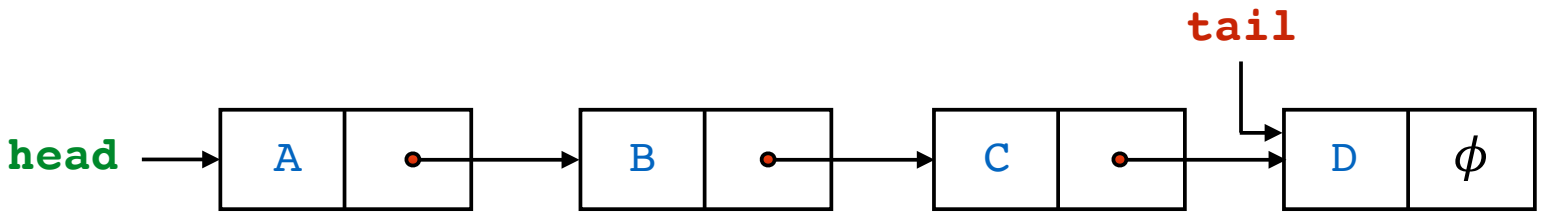
Liste singolarmente concatenate

Scansione della lista

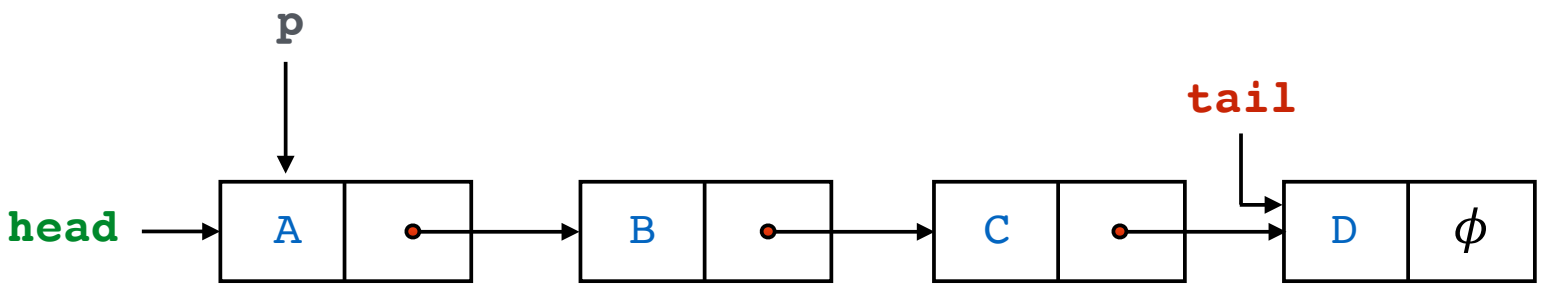


Liste singolarmente concatenate

Scansione della lista

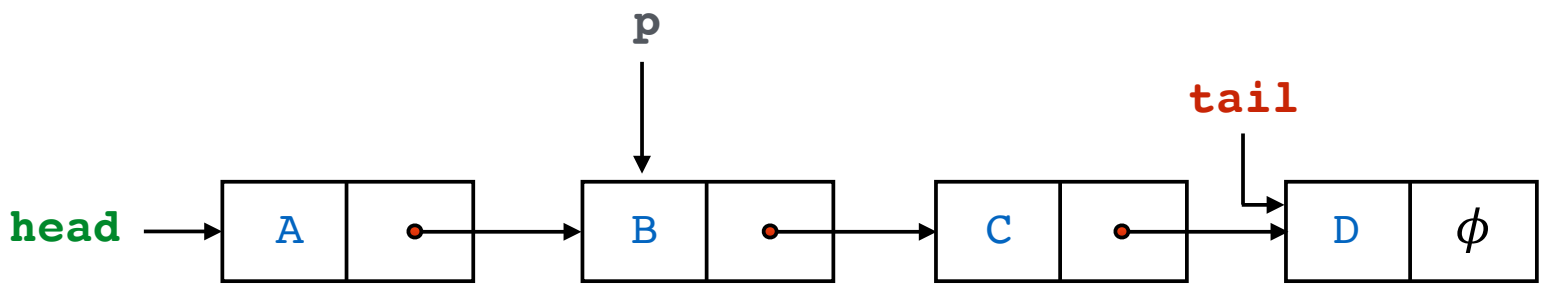


Liste singolarmente concatenate
Scansione della lista



Istruzione: `p = self.__head`

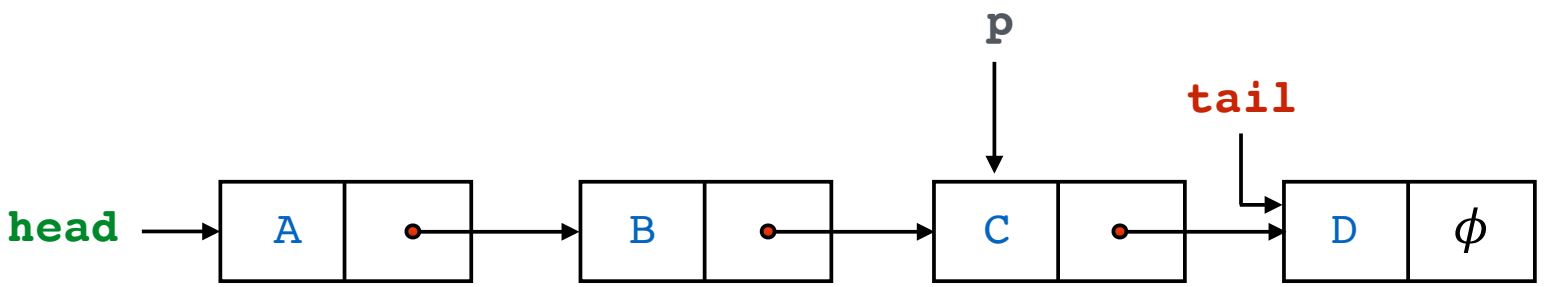
Liste singolarmente concatenate
Scansione della lista



Istruzione: `p = p.next`

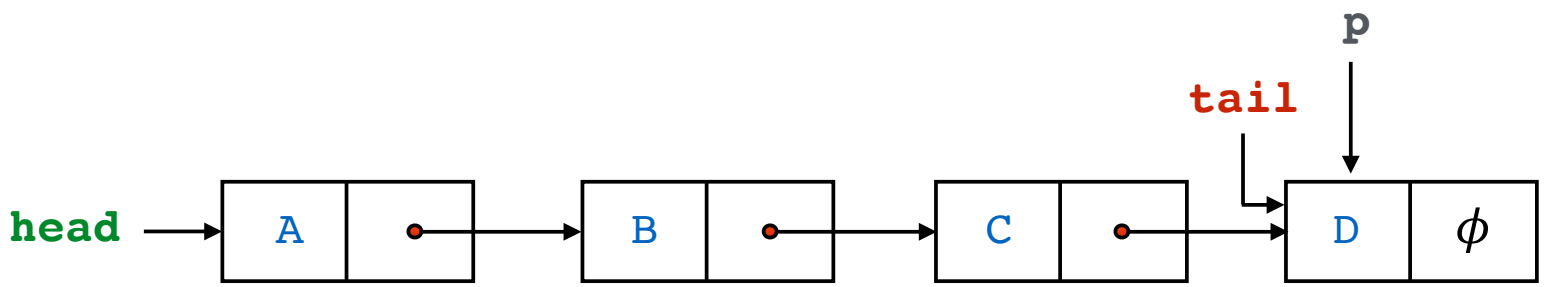
Si procede fino a raggiungere l'ultimo elemento della lista, ossia fino a quando: `p.next = None`.

Liste singolarmente concatenate
Scansione della lista



Istruzione: `p = p.next`

Liste singolarmente concatenate
Scansione della lista



Istruzione: `p = p.next`

Liste singolarmente concatenate
Scansione e stampa della lista

Codice della funzione **`stampa_lista`**:

```
def stampa_lista(node):
    p = node
    while p != None:
        data = p.data
        print(data, '->', end = ' ')
        p = p.next
    print('Fine Lista')
```

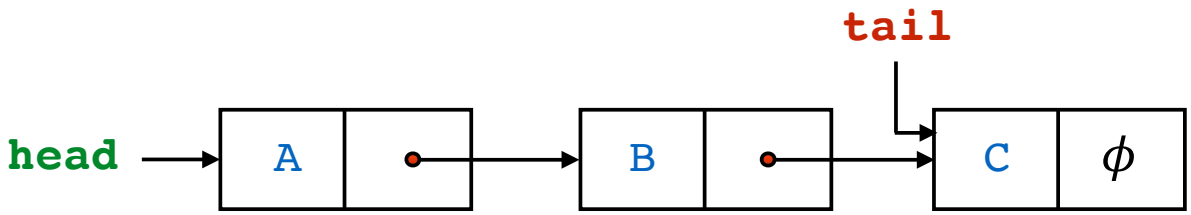
Liste semplici
Metodi per la Classe Lista

Alla Classe **`SinglyLinkedList`** vanno aggiunti dei metodi per consentire le varie operazioni richieste. Vediamo i metodi seguenti:

- `append` (inserimento in coda)
- `insert_head` (inserimento in testa)
- `insert_position` (inserimento in pos. intermedia)
- `delete` (cancellazione di un elemento)

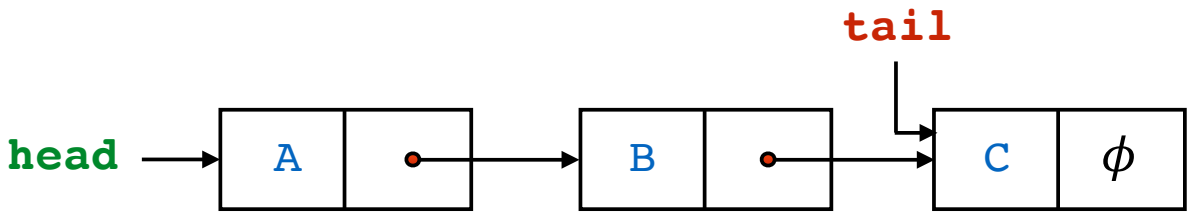
Liste singolarmente concatenate
Inserimento in coda (append)

Lista di partenza:



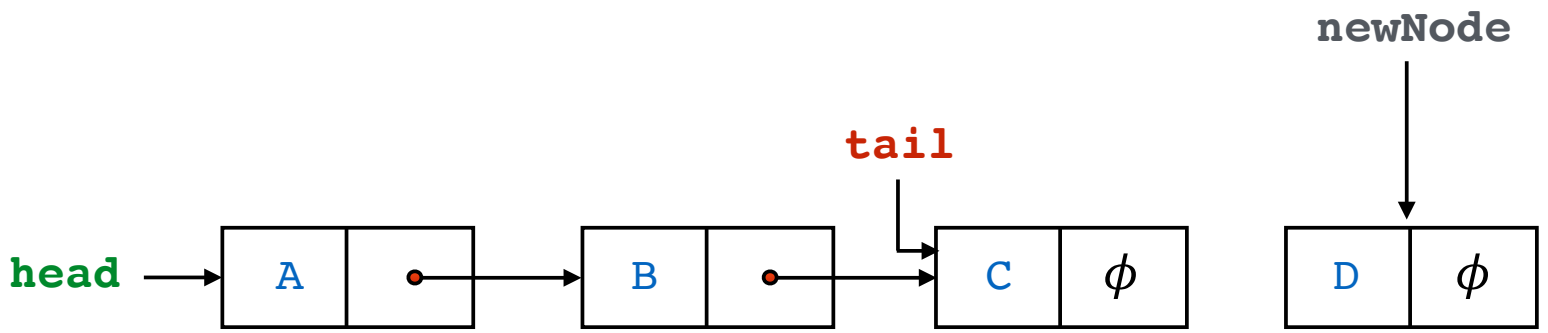
Liste singolarmente concatenate
Inserimento in coda (append)

Lista di partenza:



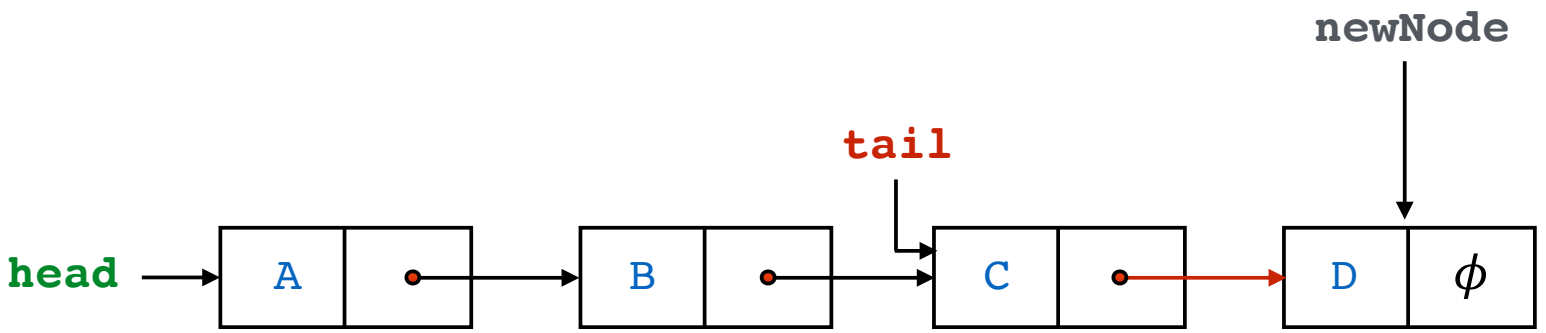
Supponiamo di creare un nuovo nodo (**`newNode`**) e di volerlo inserire in fondo alla lista (dopo l'elemento 'C')

Liste singolarmente concatenate
Inserimento in coda (append)



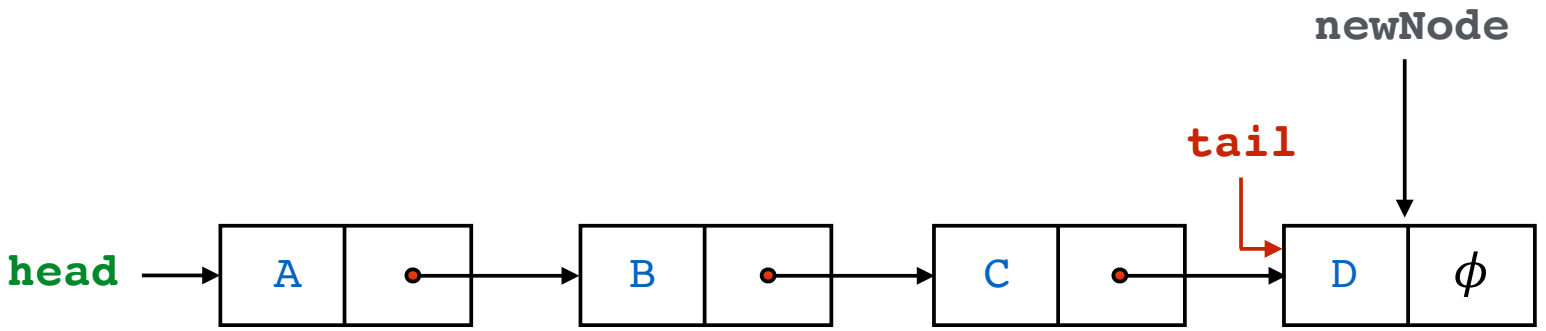
Istruzione: `Creazione dell'oggetto della classe Node da passare come newNode`

Liste singolarmente concatenate
Inserimento in coda (append)



Istruzione: `self.__tail.next = newNode`

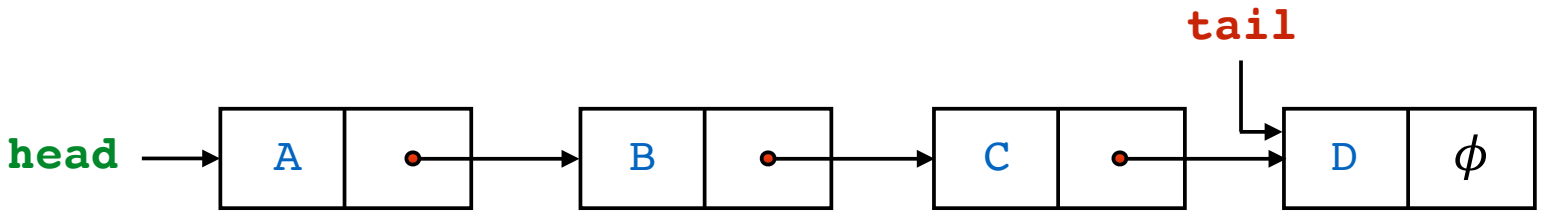
Liste singolarmente concatenate
Inserimento in coda (append)



Istruzione: `self.__tail = newNode`

Liste singolarmente concatenate
Inserimento in coda (append)

Risultato finale:



Liste singolarmente concatenate

Inserimento in coda (append)

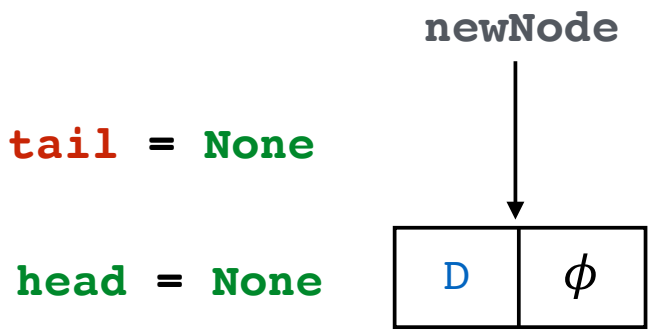
per lista vuota

```
tail = None

head = None
```

Liste singolarmente concatenate

Inserimento in coda (append)

per lista vuota

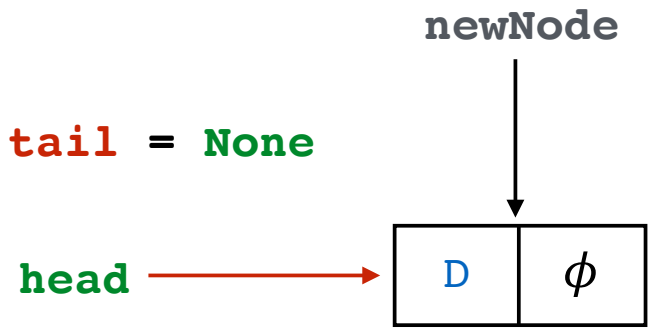


Istruzione: `Creazione dell'oggetto della classe Node da passare come newNode`

Liste singolarmente concatenate

Inserimento in coda (append)

per lista vuota

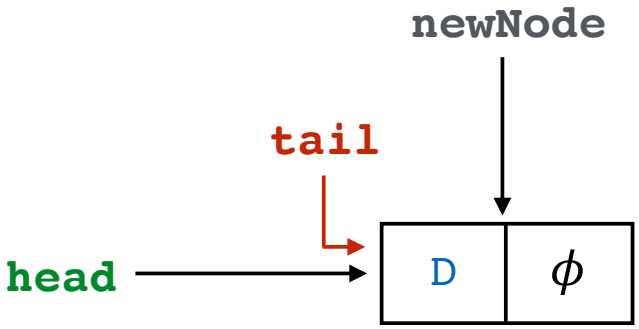


Istruzione: `self.__head = newNode`

Liste singolarmente concatenate

Inserimento in coda (append)

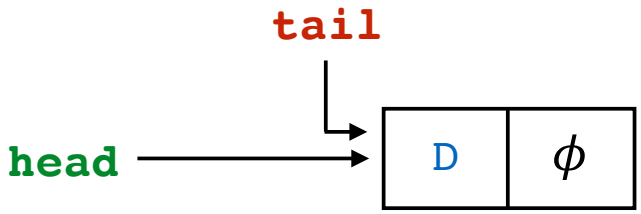
per lista vuota



Istruzione: `self.__tail = newNode`

Liste singolarmente concatenate
Inserimento in coda (append)
per lista vuota

Risultato finale:



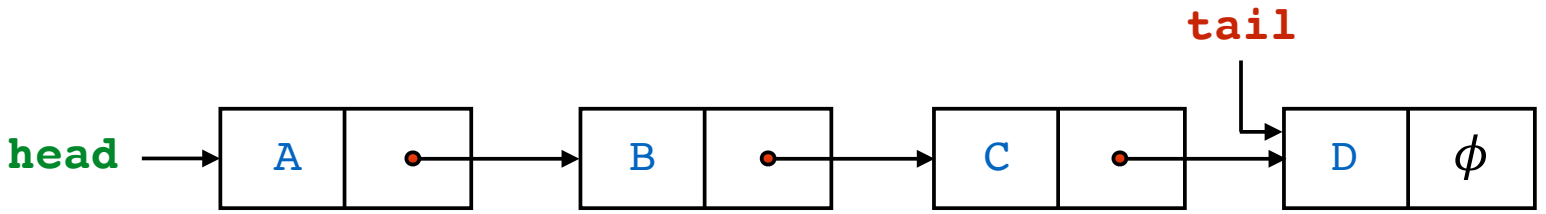
Liste singolarmente concatenate
Codice Inserimento in coda (append)

Codice del metodo **append**:

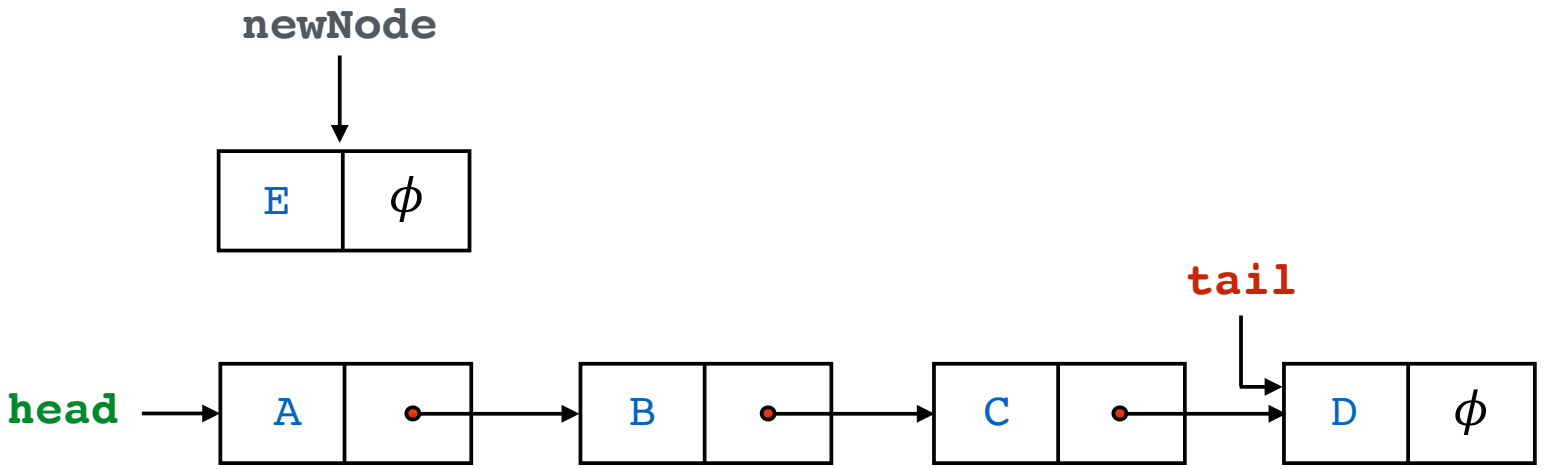
```
def append(self, newNode):
    if self.__head == None:
        self.__head = newNode
        self.__tail = newNode
        newNode.next = None
    else:
        self.__tail.next = newNode
        self.__tail = newNode
        newNode.next = None
```

Liste singolarmente concatenate
Inserimento in testa

Lista di partenza:

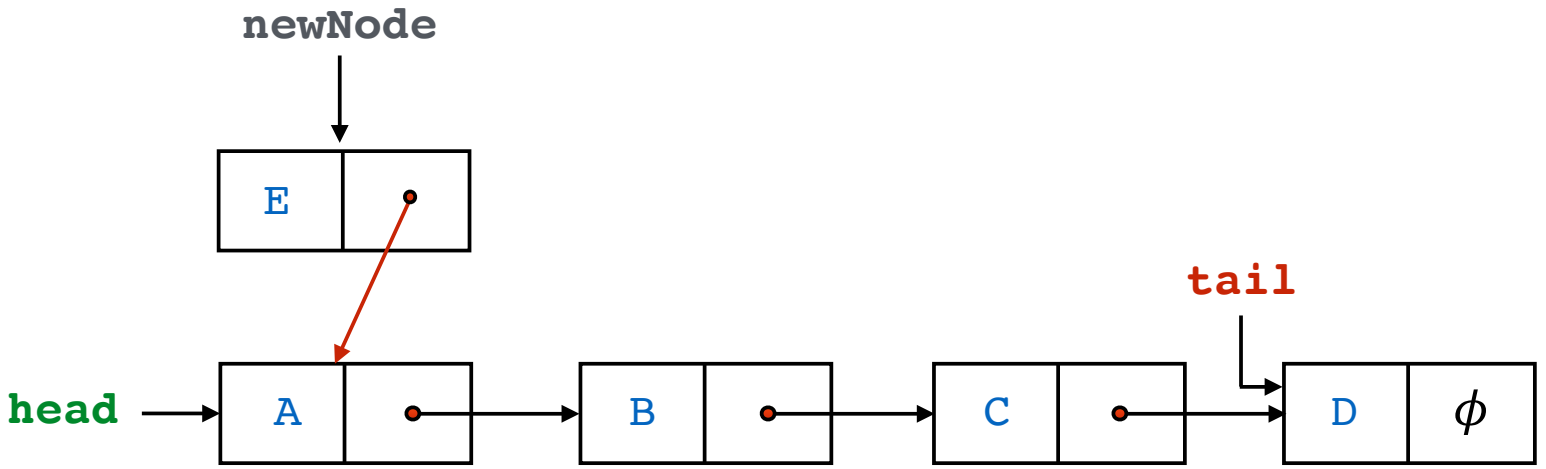


Liste singolarmente concatenate
Inserimento in testa



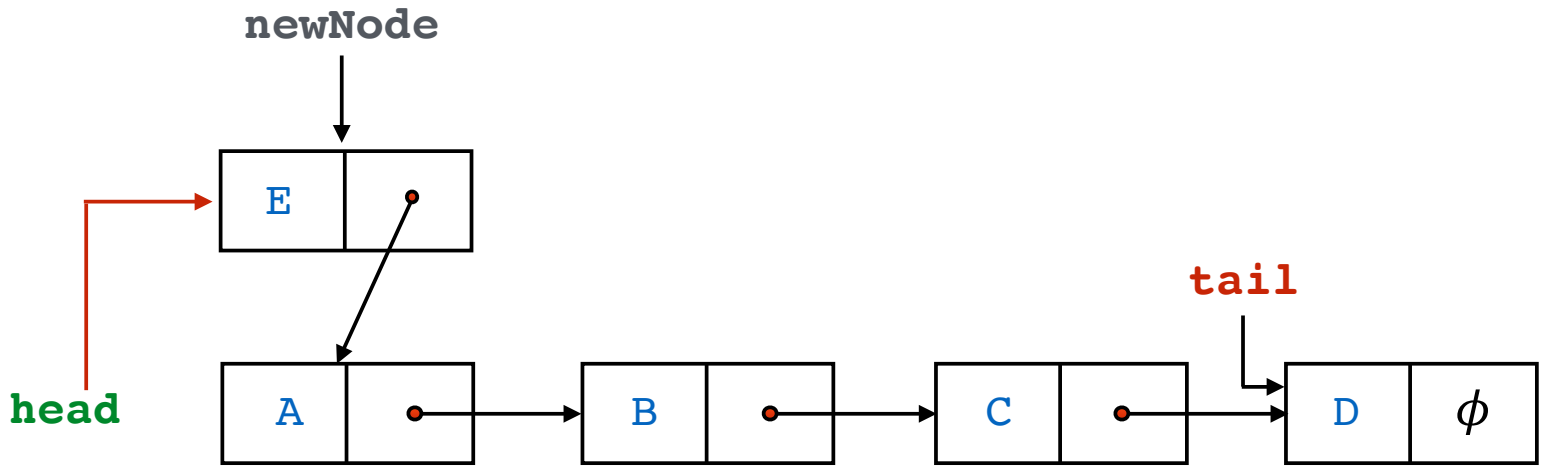
Istruzione: `Creazione dell'oggetto della classe Node da passare come newNode`

Liste singolarmente concatenate
Inserimento in testa



Istruzione: `newNode.next = self.__head`

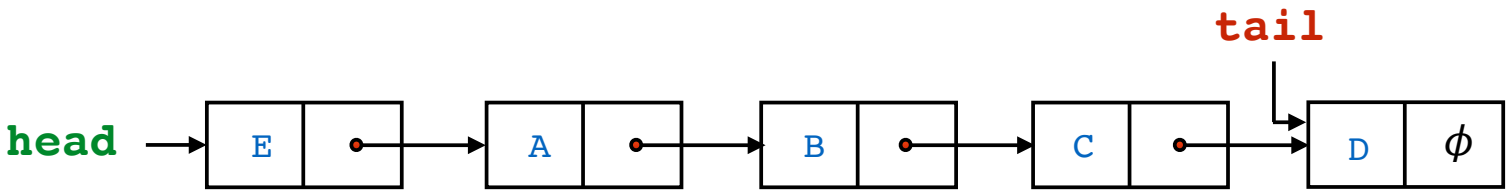
Liste singolarmente concatenate
Inserimento in testa



Istruzione: `self.__head = newNode`

Liste singolarmente concatenate
Inserimento in testa

Risultato finale:

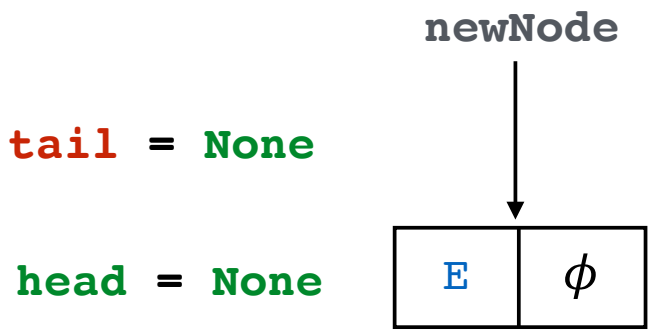


Liste singolarmente concatenate

Inserimento in testa per lista vuota

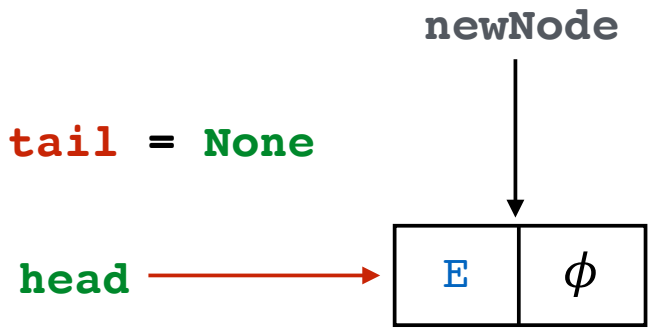
```
tail = None

head = None
```

Liste singolarmente concatenate
Inserimento in testa per lista vuota



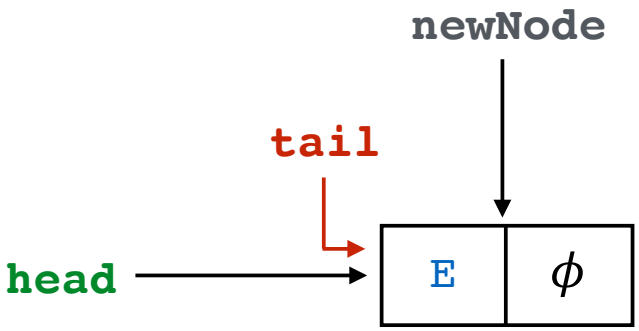
Istruzione: `Creazione dell'oggetto della classe Node da passare come newNode`

Liste singolarmente concatenate
Inserimento in testa per lista vuota



Istruzione: `self.__head = newNode`

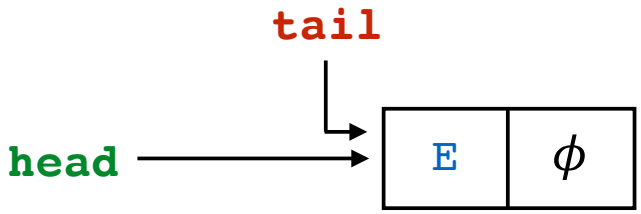
Liste singolarmente concatenate
Inserimento in testa per lista vuota



Istruzione: `self.__tail = newNode`

Liste singolarmente concatenate
Inserimento in testa per lista vuota

Risultato finale:



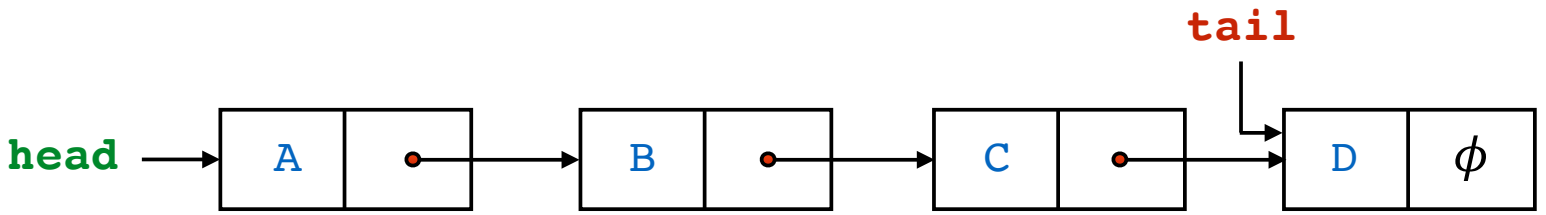
Liste singolarmente concatenate
Codice Inserimento in testa

Codice del metodo **`insert_head`**:

```
def insert_head(self, newNode):
    if self.__head == None:
        self.__head = newNode
        self.__tail = newNode
        newNode.next = None
    else:
        newNode.next = self.__head
        self.__head = newNode
```

Liste singolarmente concatenate
Inserimento di un elemento In posizione intermedia

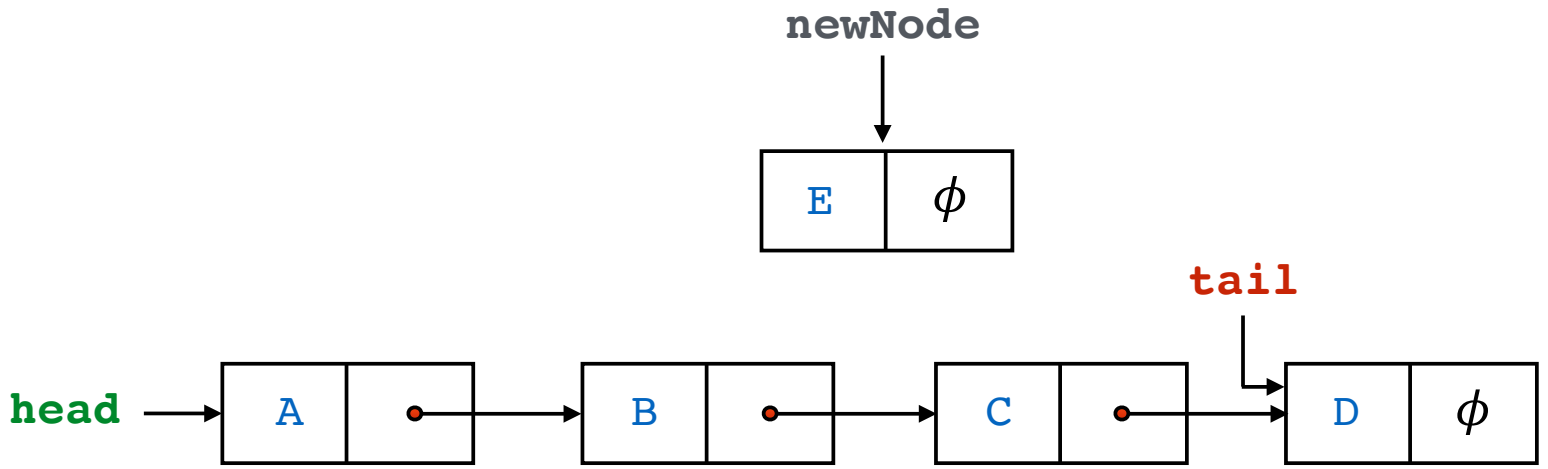
Lista di partenza:



Supponiamo di creare un nuovo nodo (**newNode**) e di volerlo inserire in posizione i = 2 (i parte da 0)

Liste singolarmente concatenate

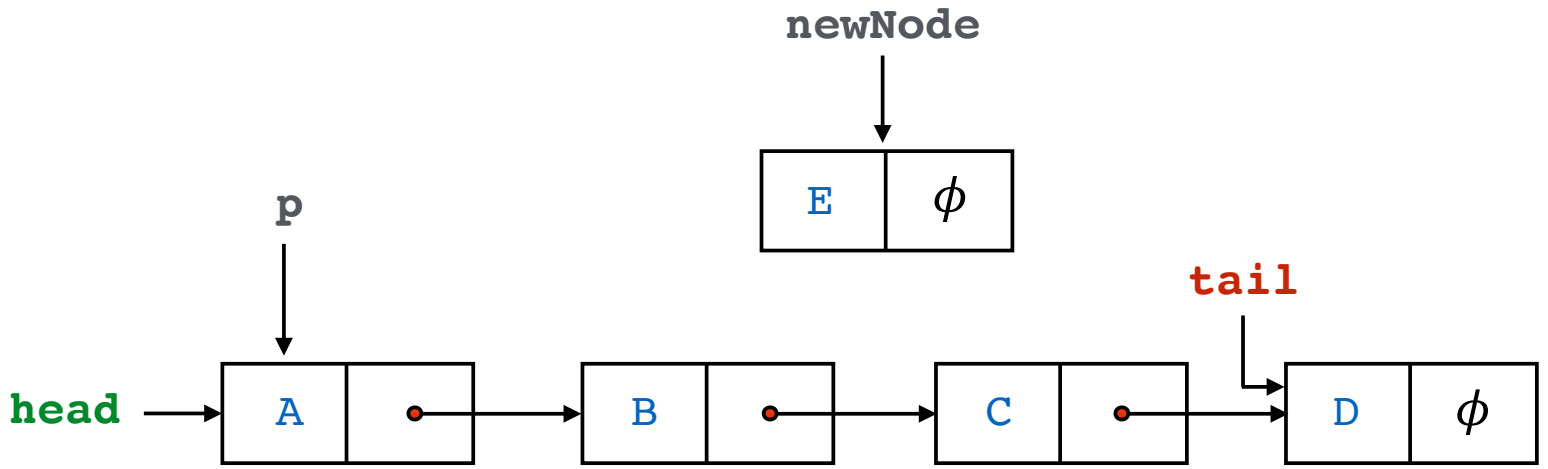
Inserimento di un elemento in posizione intermedia



Istruzione: `Creazione dell'oggetto della classe Node da passare come newNode`

Liste singolarmente concatenate

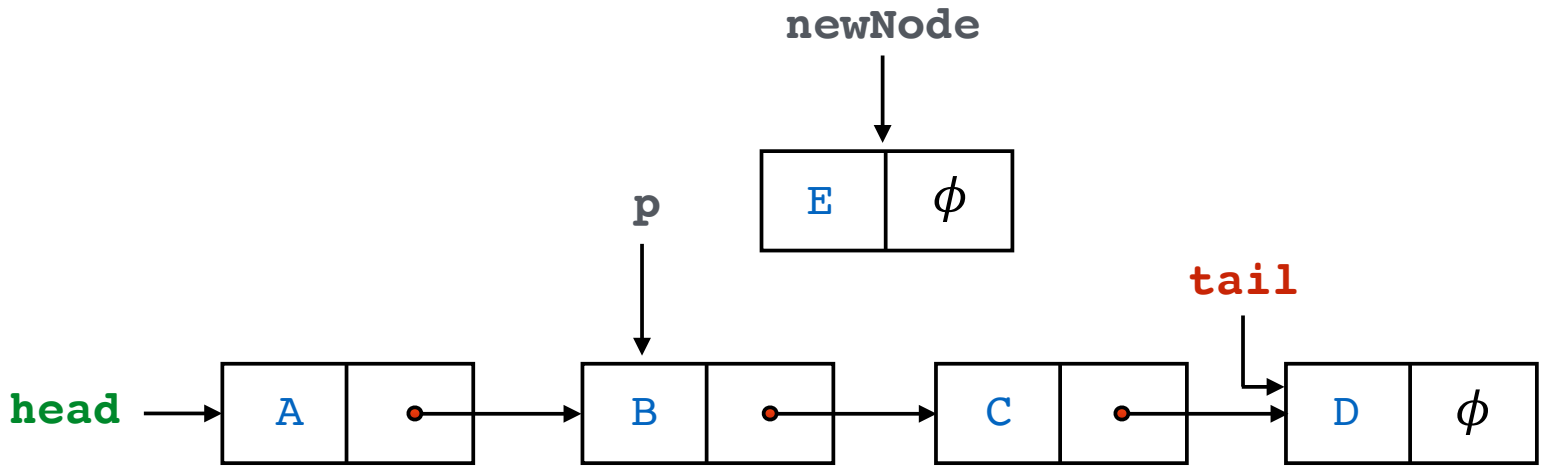
Inserimento di un elemento in posizione intermedia



Istruzione: `p = self.__head`

Liste singolarmente concatenate

Inserimento di un elemento in posizione intermedia

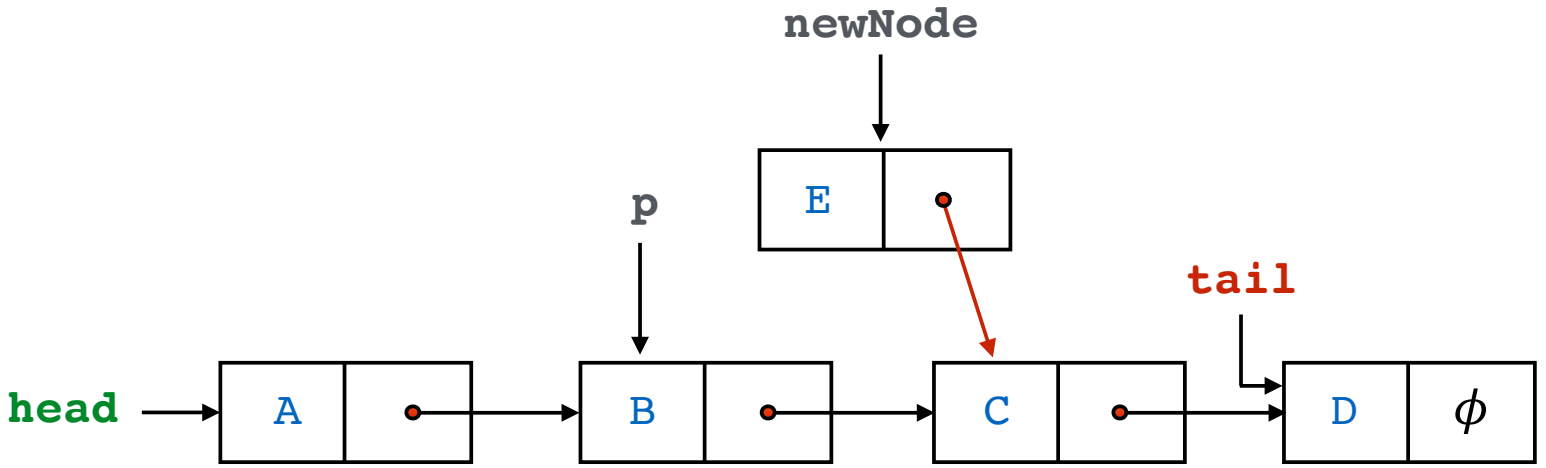


Istruzione: `p = p.next`

Si procede fino a raggiungere la posizione precedente quella dell'inserimento.

Liste singolarmente concatenate

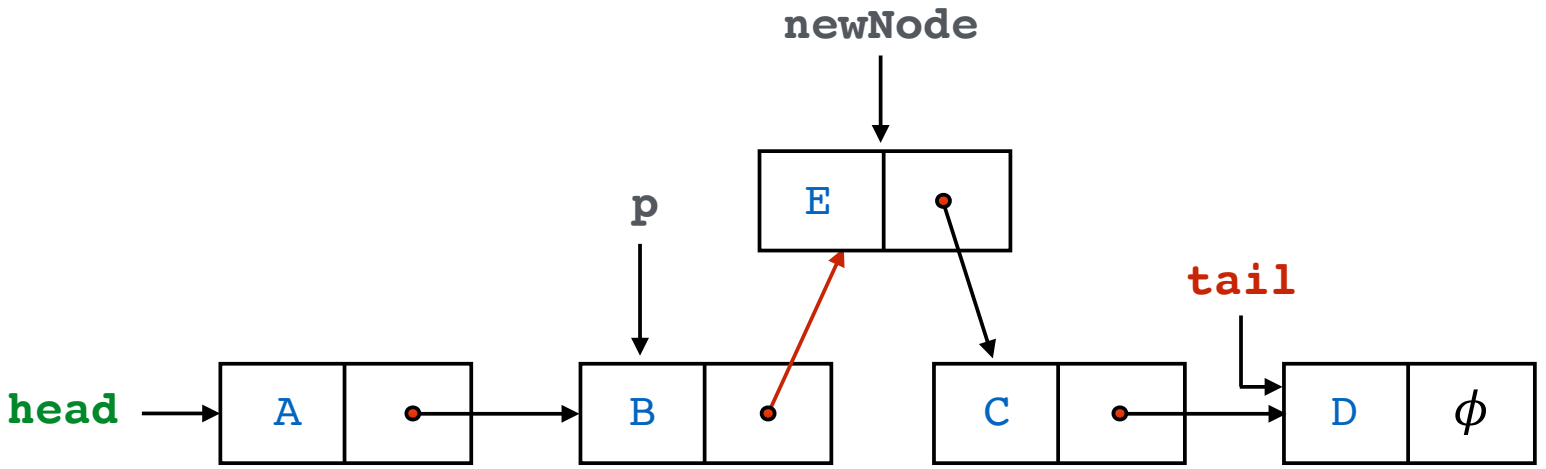
Inserimento di un elemento in posizione intermedia



Istruzione: `newNode.next = p.next`

Liste singolarmente concatenate

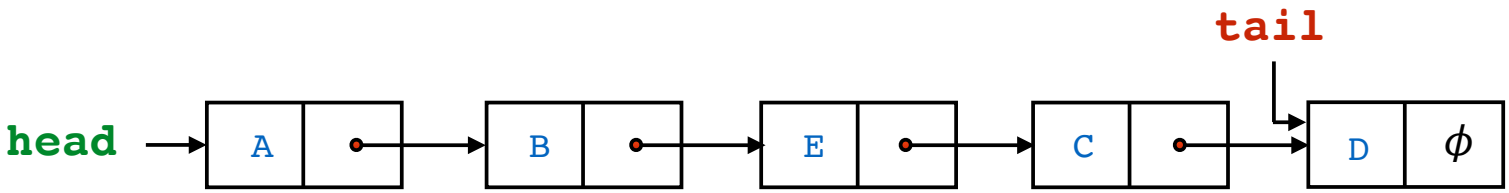
Inserimento di un elemento in posizione intermedia



Istruzione: `p.next = newNode`

Liste singolarmente concatenate
Inserimento di un elemento
In posizione intermedia

Risultato finale:



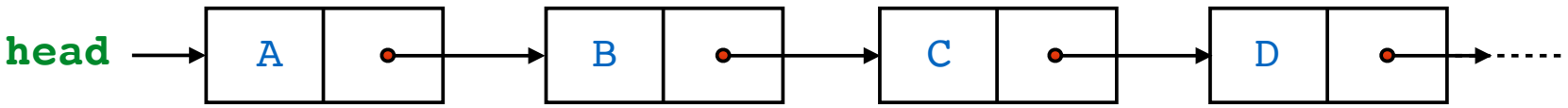
Liste singolarmente concatenate
Inserimento di un elemento in posizione intermedia

Codice del metodo **`insert_position`**:

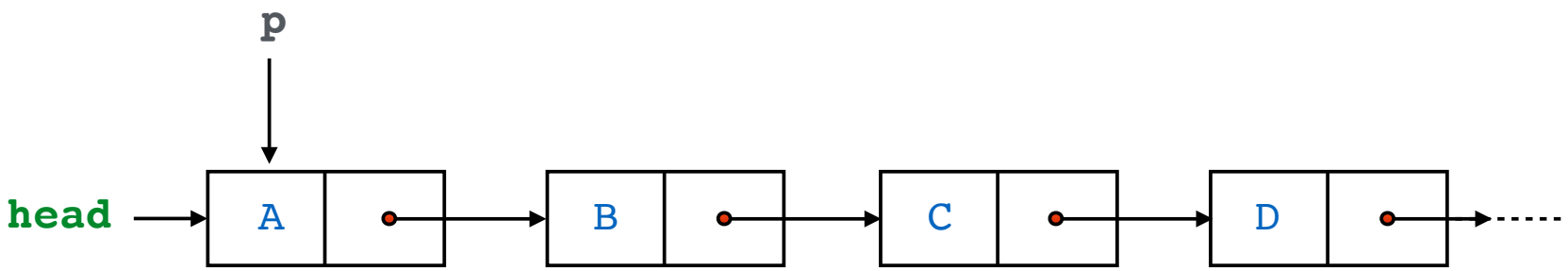
```
def insert_position(self, insertPosition, newNode):
    p = self.__head
    i = 0
    # Spostiamo il puntatore p nella posizione di inserimento
    while p != None and i < insertPosition - 1:
        p = p.next
        i = i +1
    newNode.next = p.next
    p.next = newNode
```

Liste singolarmente concatenate
Cancellazione dell'elemento i-esimo

Lista di partenza:

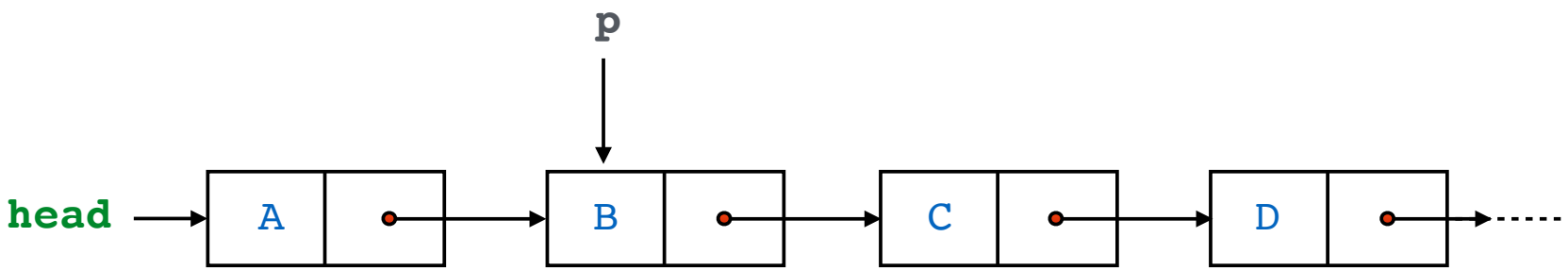


Liste singolarmente concatenate
Cancellazione dell'elemento i-esimo



Istruzione: `p = self.__head`

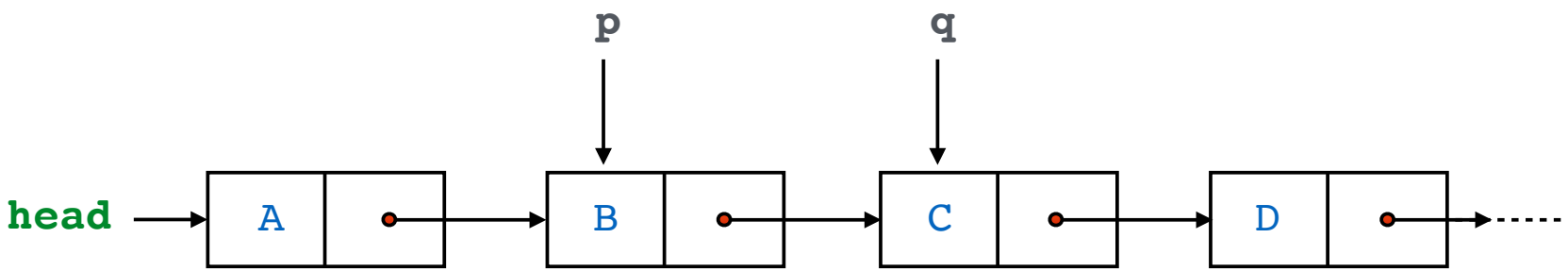
Liste singolarmente concatenate
Cancellazione dell'elemento i-esimo



Istruzione: `p = p.next`

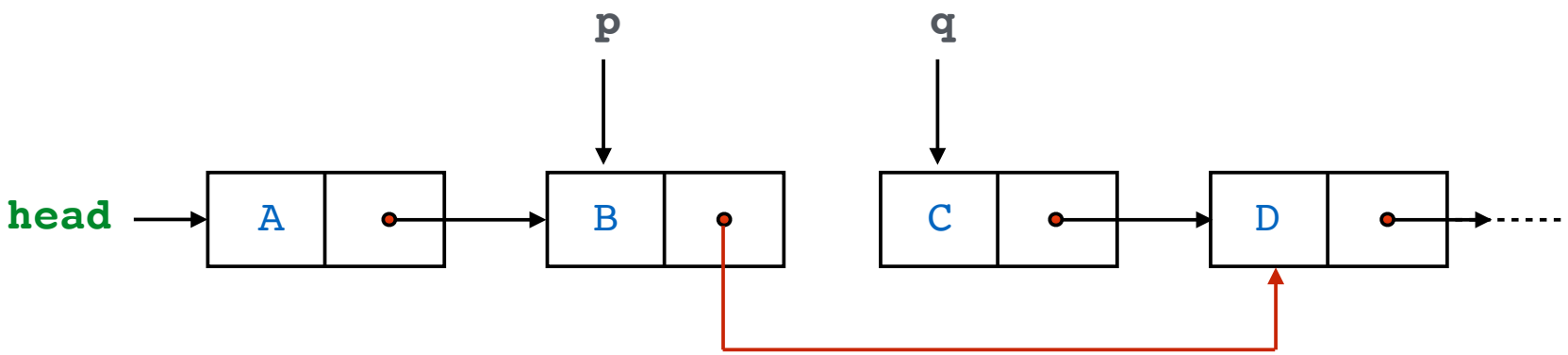
Si procede fino a raggiungere la posizione precedente quella della cancellazione. Nell'esempio: i = 2 (i parte da 0)

Liste singolarmente concatenate
Cancellazione dell'elemento i-esimo



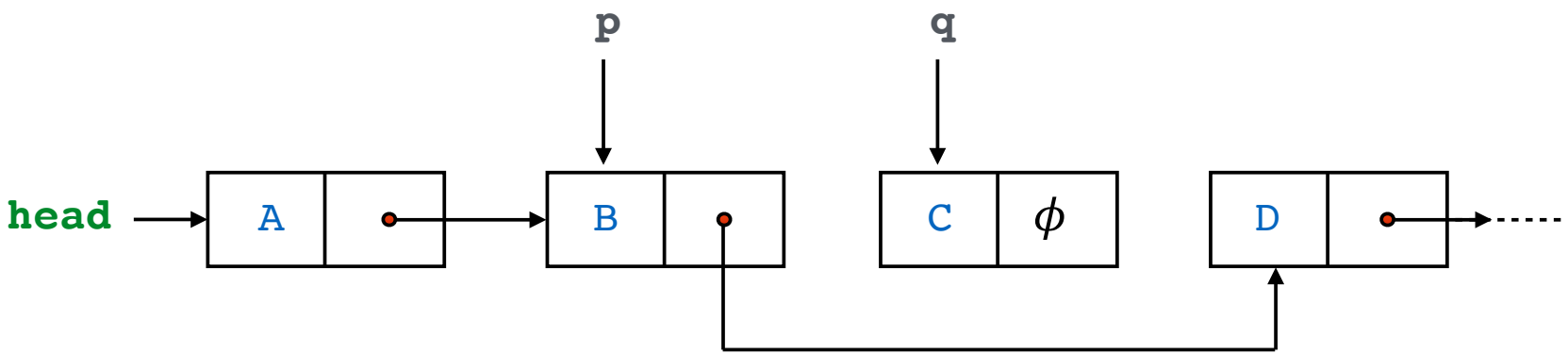
Istruzione: `q = p.next`

Liste singolarmente concatenate
Cancellazione dell'elemento i-esimo



Istruzione: `p.next = q.next`

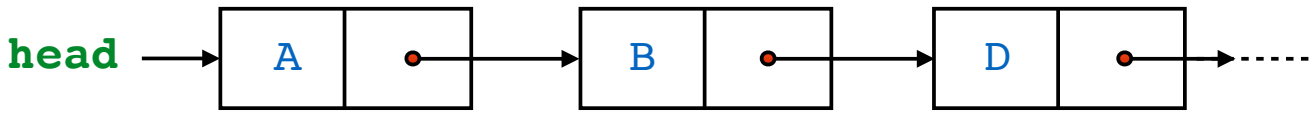
Liste singolarmente concatenate
Cancellazione dell'elemento i-esimo



Istruzione: `q.next = None`

Liste singolarmente concatenate
Cancellazione dell'elemento i-esimo

Risultato finale:



Liste singolarmente concatenate
Cancellazione dell'elemento i-esimo

Codice del metodo **`delete`**:

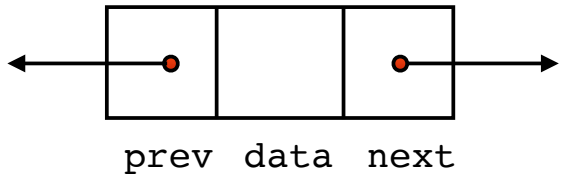
```
def delete(self, deletePosition):
    p = self.__head
    if p != None:
        if deletePosition == 0:
            self.__head = p.next
            p.next = None
        else:
            i = 0
            # Spostiamo il puntatore sul nodo immediatamente precedente
            # a quello che vogliamo rimuovere
            while p.next != None and i < (deletePosition - 1):
                p = p.next
                i += 1
            q = p.next
            p.next = q.next
            q.next = None
    else:
        print('Lista vuota')
```

Liste doppiamente concatenate
Nodi, Liste e Operazioni

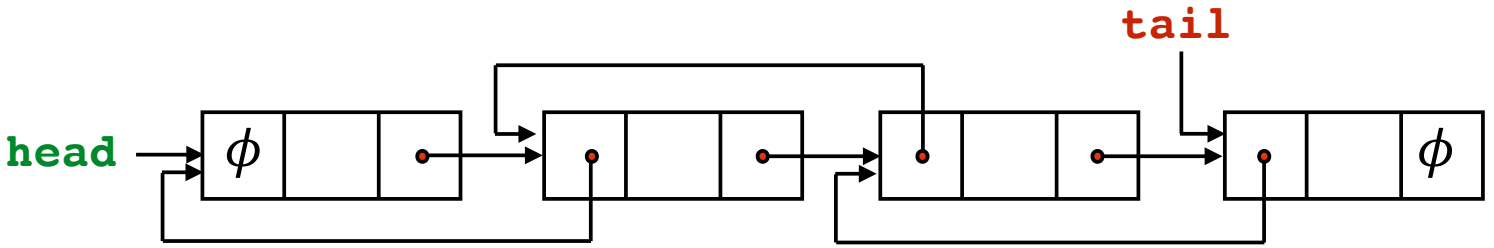
- Rappresentazione e creazione dei **nodi** di una lista
- Rappresentazione e creazione di una **lista**
- **Scansione** di una lista doppiamente concatenata
- Operazioni di **Inserimento** di un nuovo elemento
- Operazioni di **cancellazione** di un elemento
- Operazioni di **ricerca** di un elemento

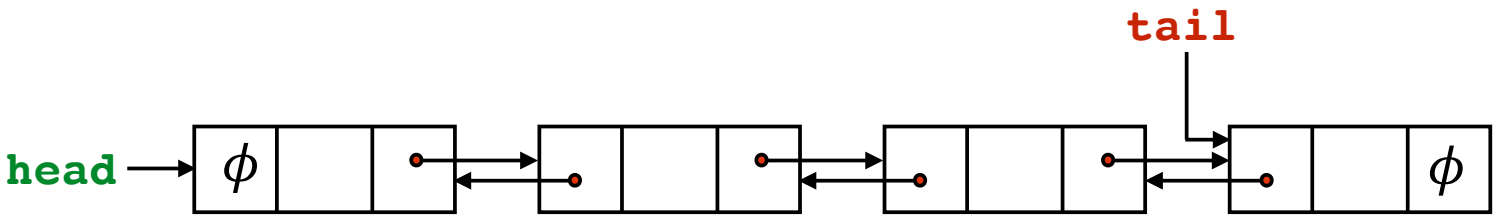
Liste doppie
Rappresentazione nodo

In tali liste un nodo contiene, oltre al dato, due puntatori, uno che punta al nodo successivo e uno al nodo precedente:



Liste doppie
Rappresentazione grafica





Liste doppie
Definizione della Classe Node

```
class Node:
    data = ''
    prev = None
    next = None

    def __init__(self, data):
        self.data = data
```

Istanze di questa Classe sono oggetti **`Node`** con attributi **`data`**, **`prev`** e **`next`**. L'attributo **`data`** è impostato dal costruttore con il valore del parametro passato.

Liste doppie
Definizione della Classe Lista

```
class DoublyLinkedList:
#     __head = None
#     __tail = None

    def __init__(self):
        self.__head = None
        self.__tail = None
```

Istanze di questa Classe sono oggetti **`DoublyLinkedList`** con i puntatori **`head`** e **`tail`** impostati a **`None`**. Sono quindi inizializzati a liste vuote.

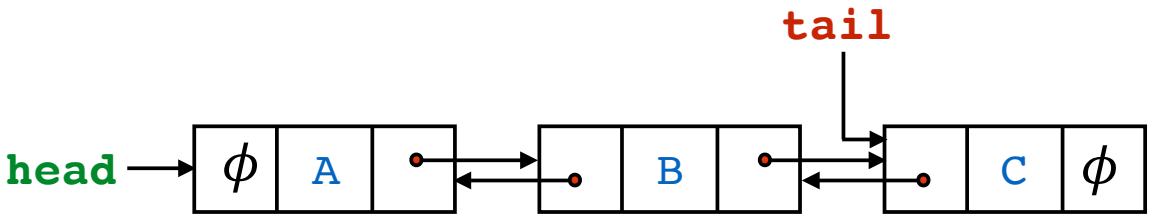
Liste doppie
Metodi per la Classe Lista

Alla Classe **`DoublyLinkedList`** vanno aggiunti dei metodi per consentire le varie operazioni richieste. Vediamo ora i metodi seguenti:

- `append` (inserimento in coda)
- `insert_head` (inserimento in testa)
- `insert_position` (inserimento in pos. intermedia)
- `delete` (cancellazione di un elemento)

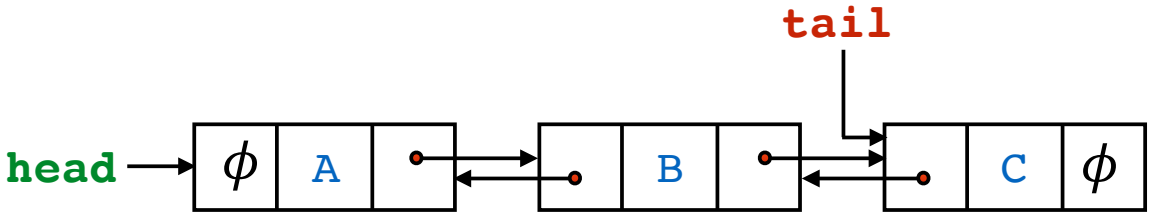
Liste doppie
Inserimento in coda (append)

Lista di partenza:



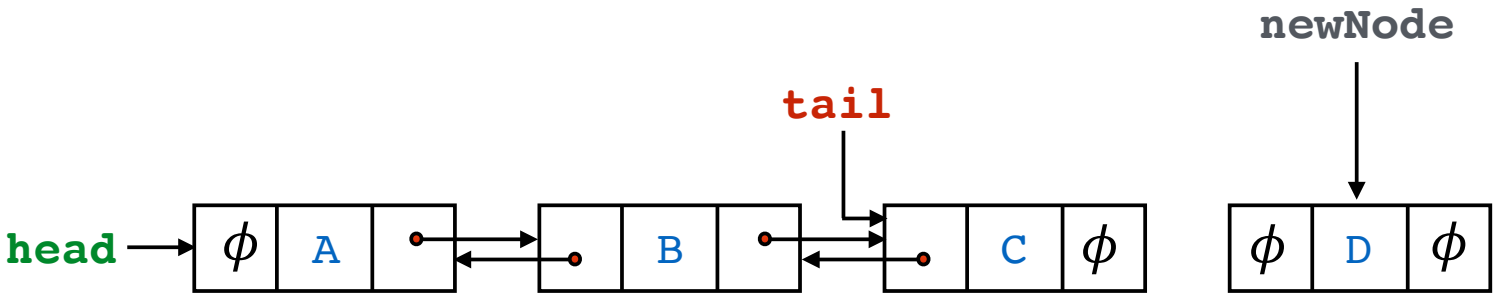
Liste doppie
Inserimento in coda (append)

Lista di partenza:



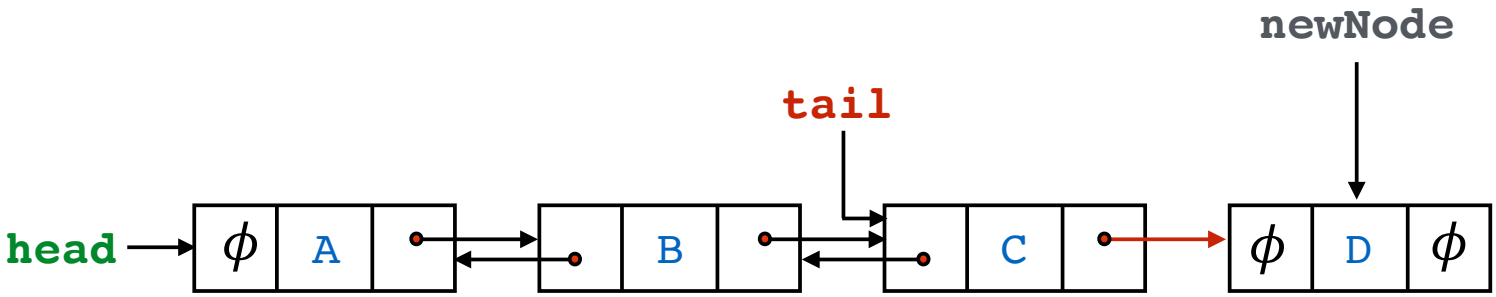
Supponiamo di creare un nuovo nodo (**newNode**) e di volerlo inserire in fondo alla lista (dopo l'elemento 'C')

Liste doppie
Inserimento in coda (append)



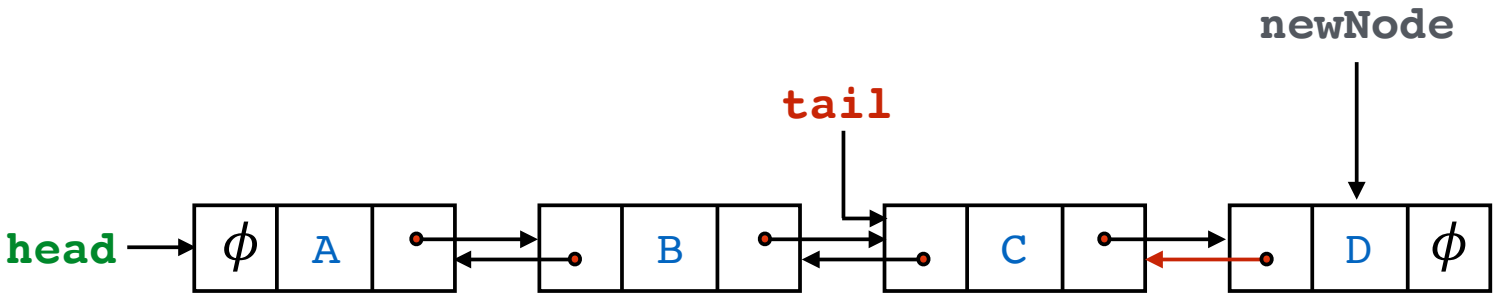
Istruzione: `Creazione dell'oggetto della classe Node da passare come newNode`

Liste doppie
Inserimento in coda (append)



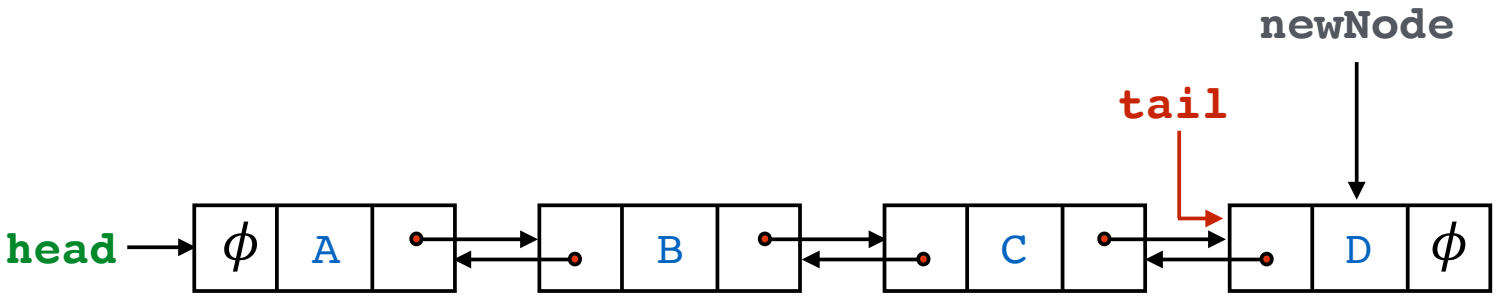
Istruzione: `self.__tail.next = newNode`

Liste doppie
Inserimento in coda (append)



Istruzione: `newNode.prev = self.__tail`

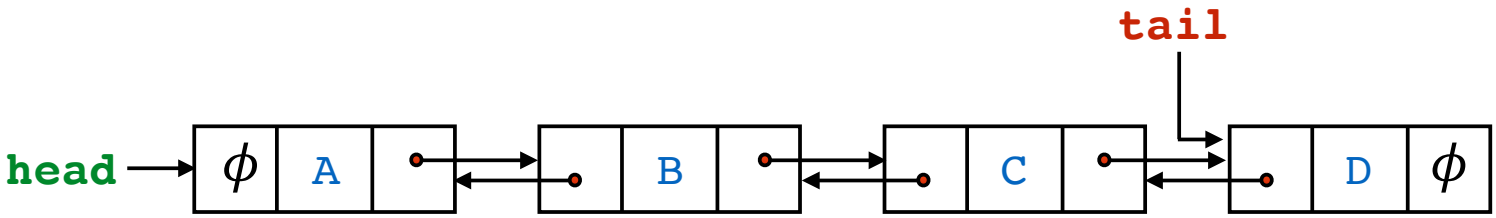
Liste doppie
Inserimento in coda (append)



Istruzione: `self.__tail = newNode`

Liste doppie
Inserimento in coda (append)

Risultato finale:



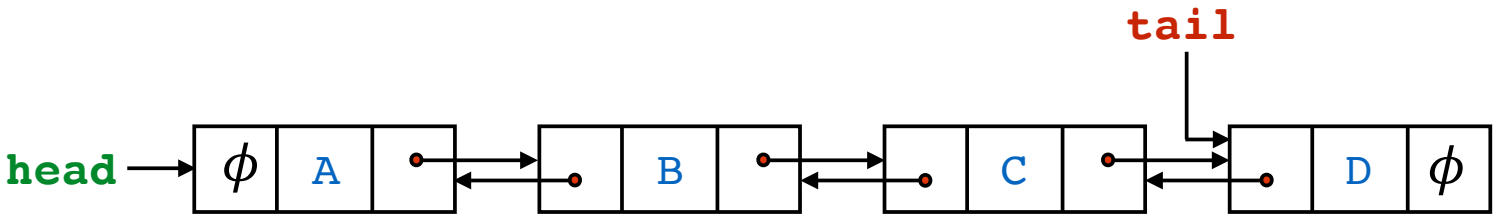
Liste doppie
Inserimento in coda (append)

Codice del metodo **append**:

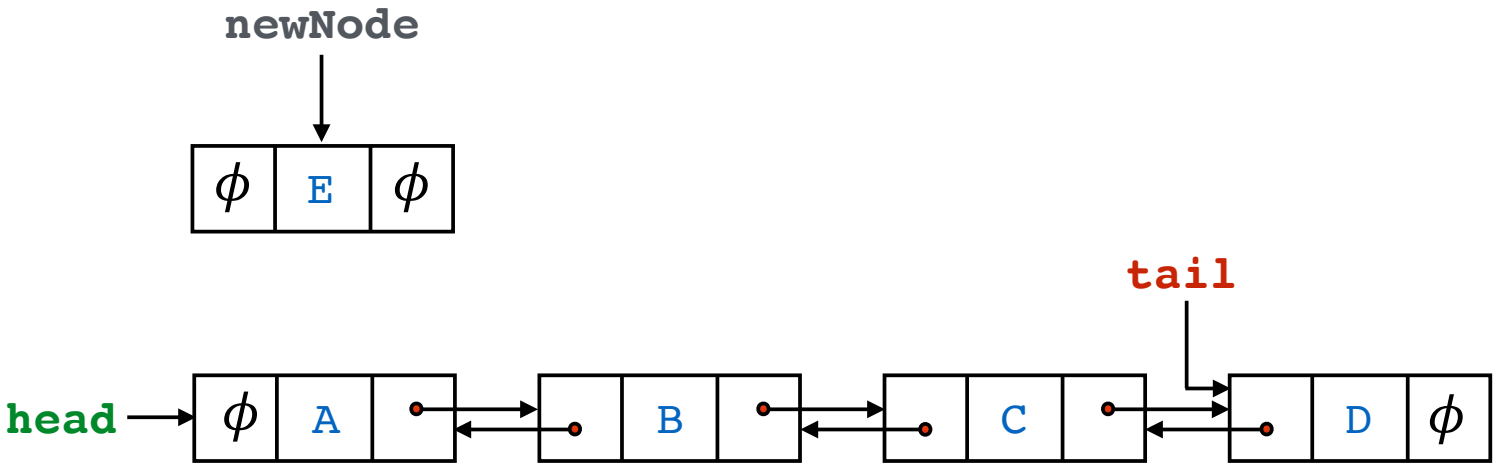
```
def append(self, newNode):
    if self.__head == None:
        self.__head = newNode
        self.__tail = newNode
    else:
        self.__tail.next = newNode
        newNode.prev = self.__tail
        self.__tail = newNode
```

Liste doppie
Inserimento in testa

Lista di partenza:

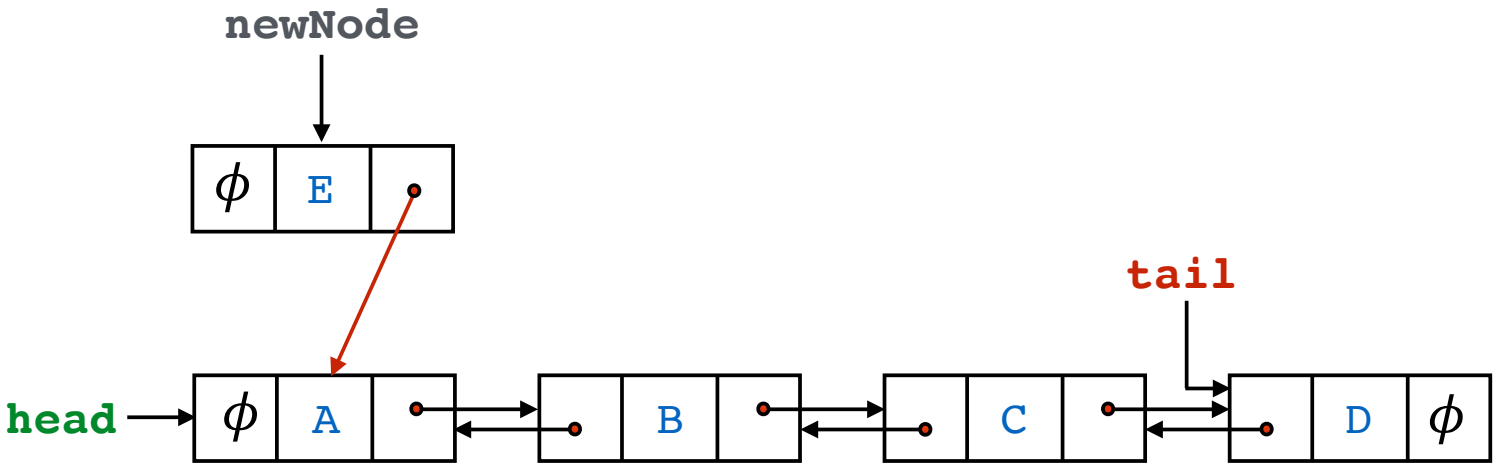


Liste doppie
Inserimento in testa



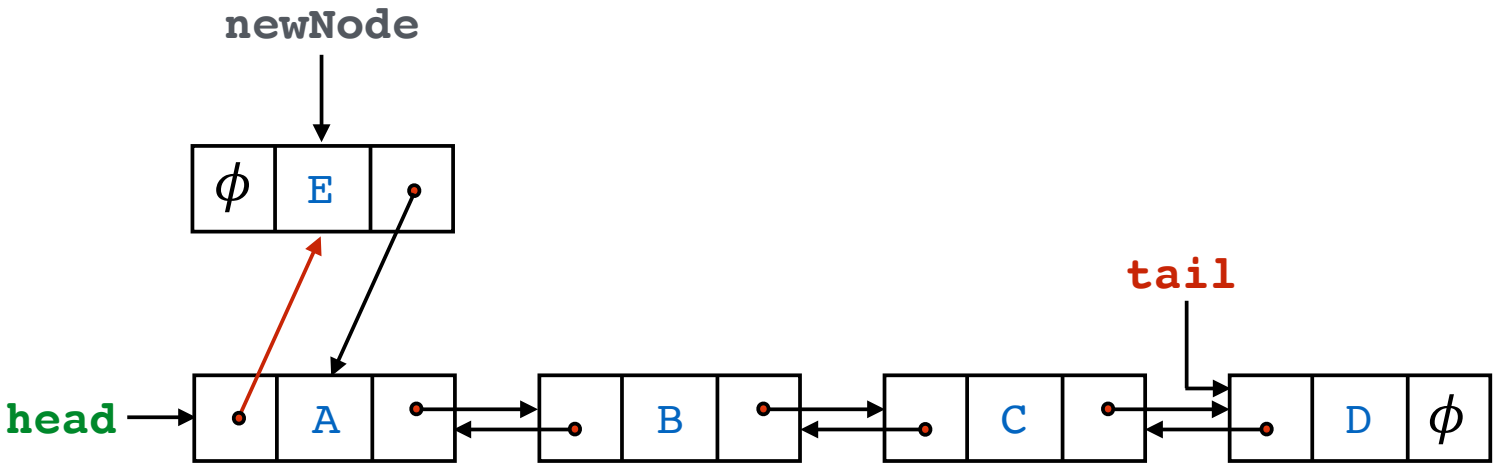
Istruzione: **Creazione dell'oggetto della classe Node da passare come newNode**

Liste doppie
Inserimento in testa



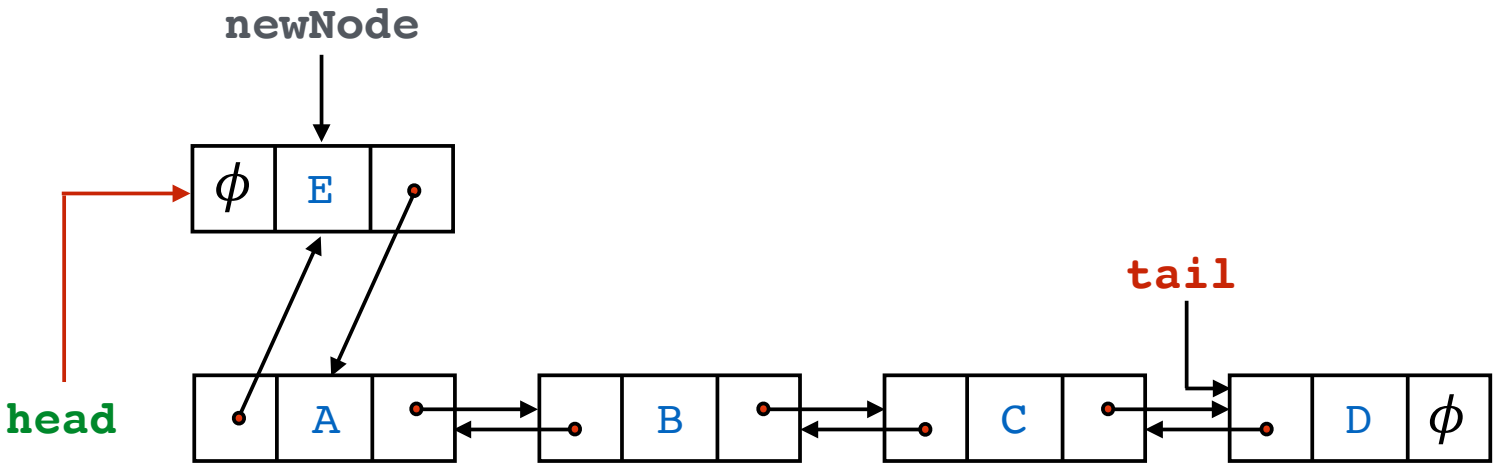
Istruzione: `newNode.next = self.__head`

Liste doppie
Inserimento in testa



Istruzione: `self.__head.prev = newNode`

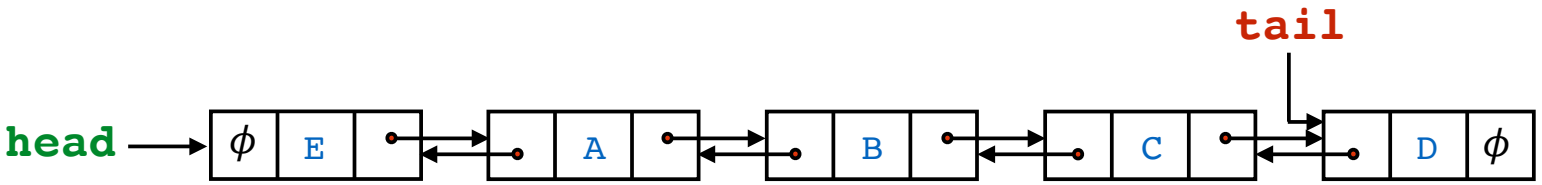
Liste doppie
Inserimento in testa



Istruzione: `self.__head = newNode`

Liste doppie
Inserimento in testa

Risultato finale:



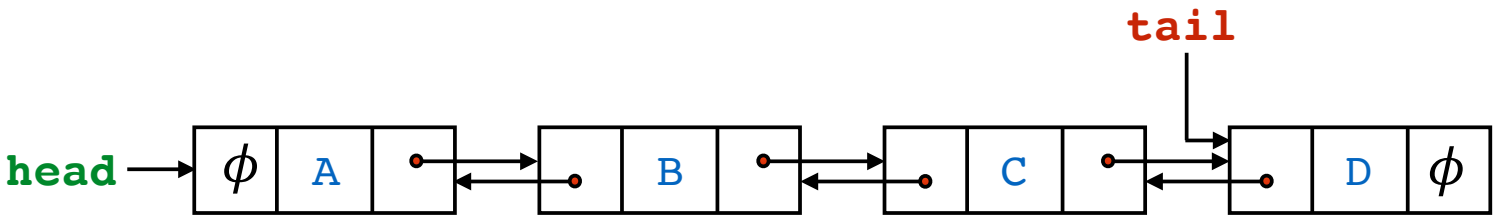
Liste doppie
Inserimento in testa

Esercizio proposto:

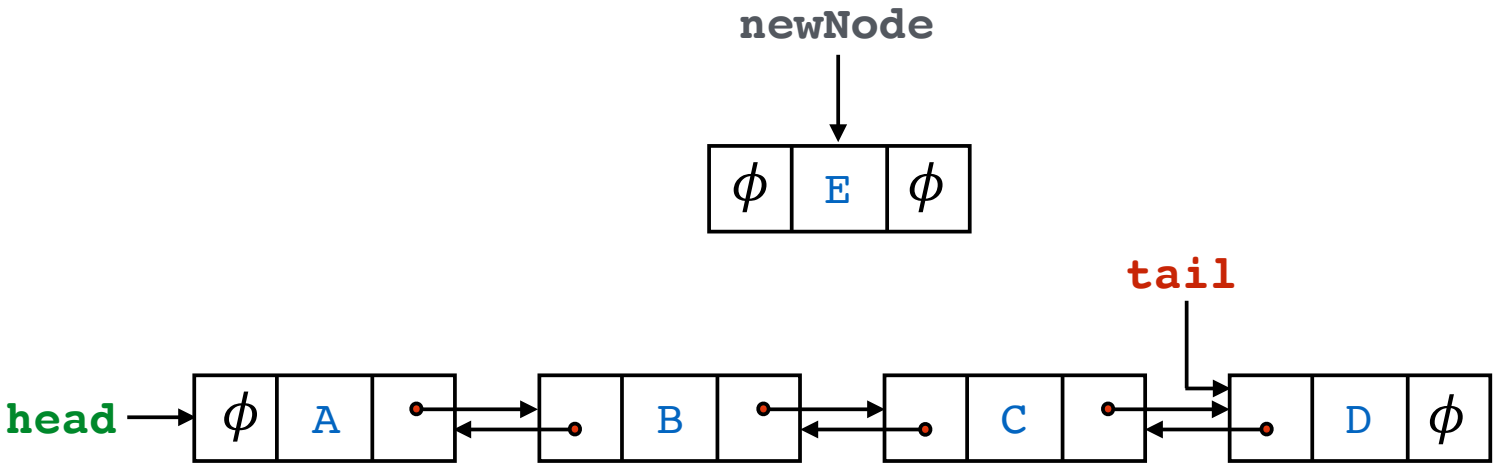
Scrivere il codice Python per il metodo **`insert_head`** che realizza l'operazione di inserimento in testa, prevedendo anche il caso di lista vuota.

Liste doppie
Inserimento di un elemento in posizione intermedia

Vediamo come si può inserire un elemento (**newNode**) in posizione i-esima. Lista di partenza:

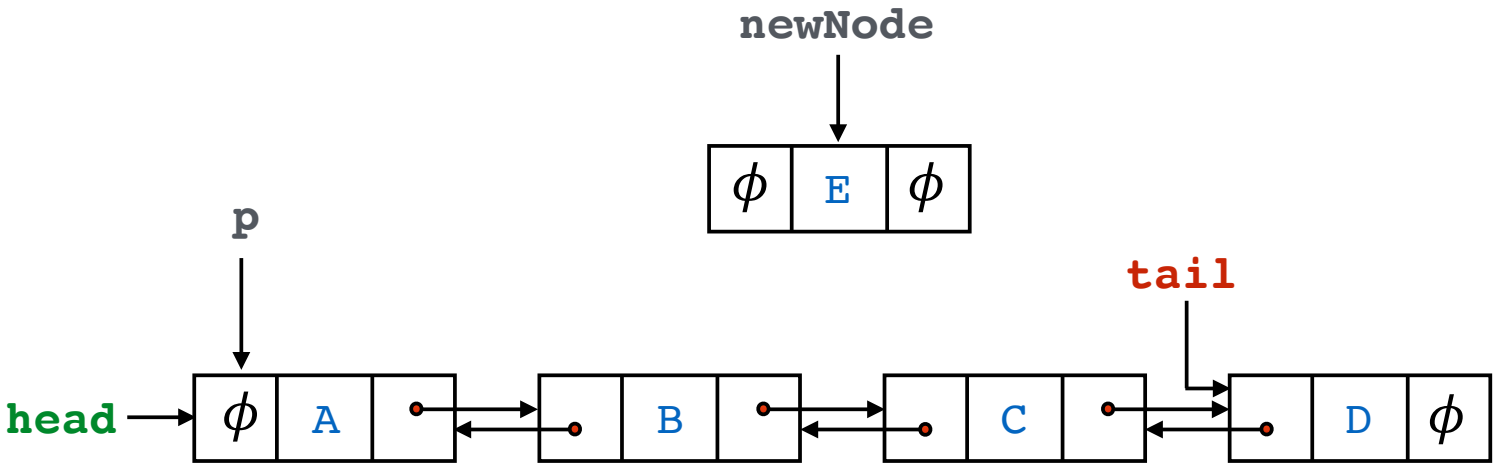


Liste doppie
Inserimento di un elemento In posizione intermedia



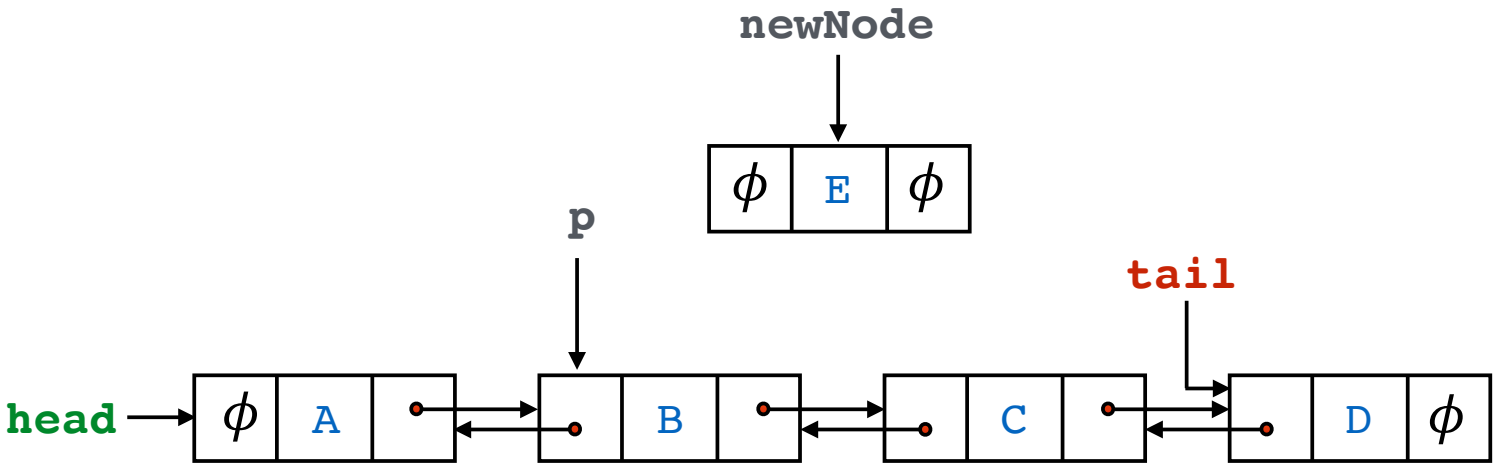
Istruzione: **Creazione dell'oggetto della classe Node da passare come newNode**

Liste doppie
Inserimento di un elemento In posizione intermedia



Istruzione: `p = self.__head`

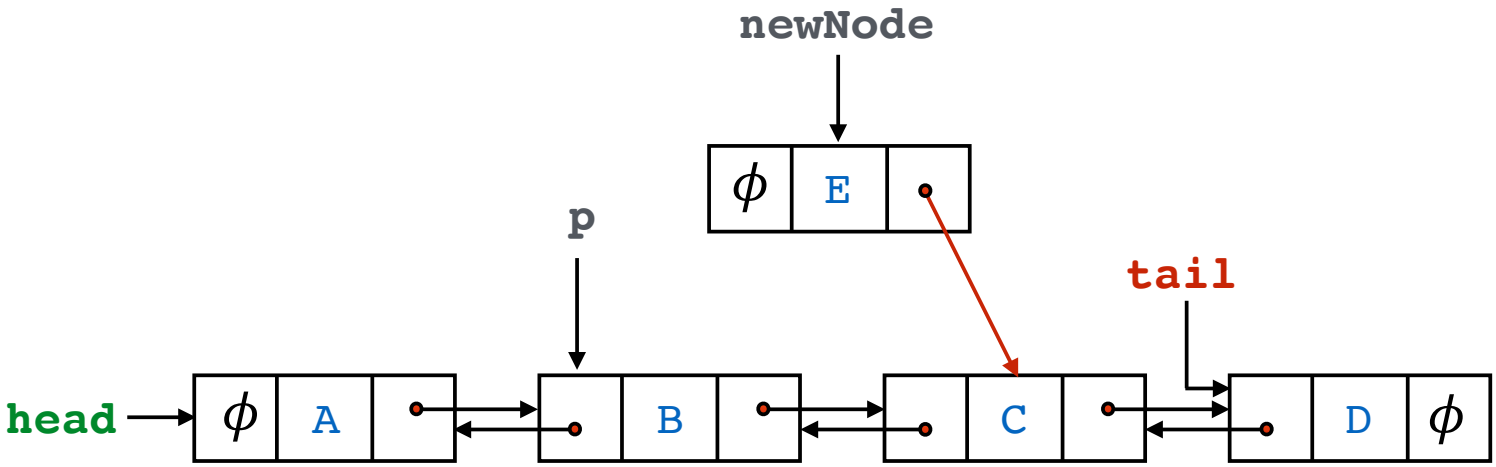
Liste doppie
Inserimento di un elemento In posizione intermedia



Istruzione: `p = p.next`

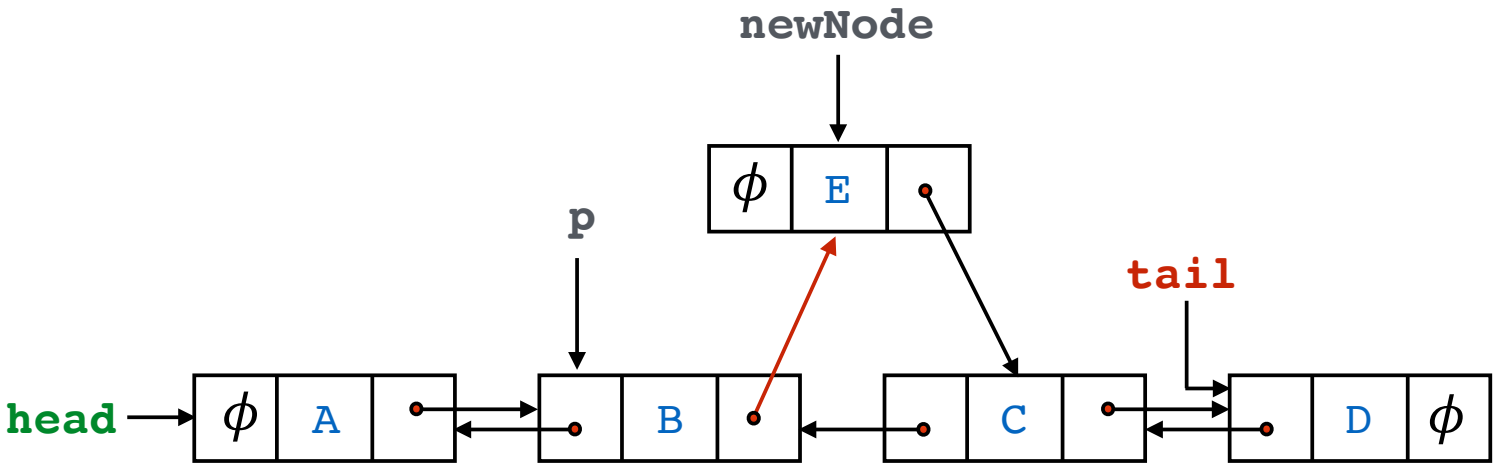
Si procede fino a raggiungere la posizione precedente quella dell'inserimento. Nell'esempio: i = 2 (i parte da 0)

Liste doppie
Inserimento di un elemento in posizione intermedia



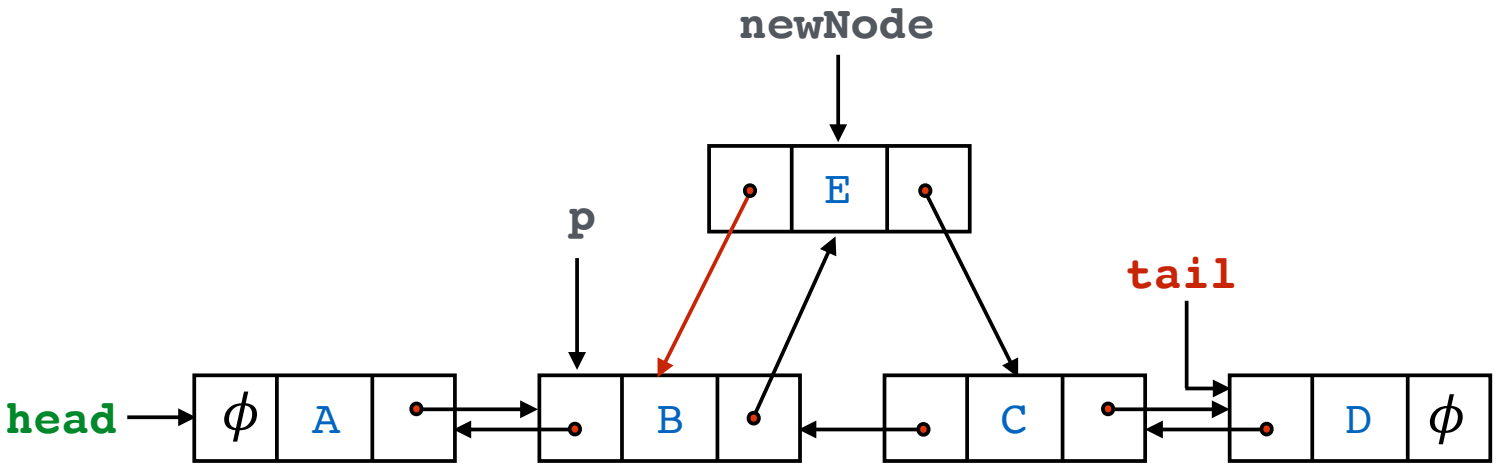
Istruzione: `newNode.next = p.next`

Liste doppie
Inserimento di un elemento In posizione intermedia



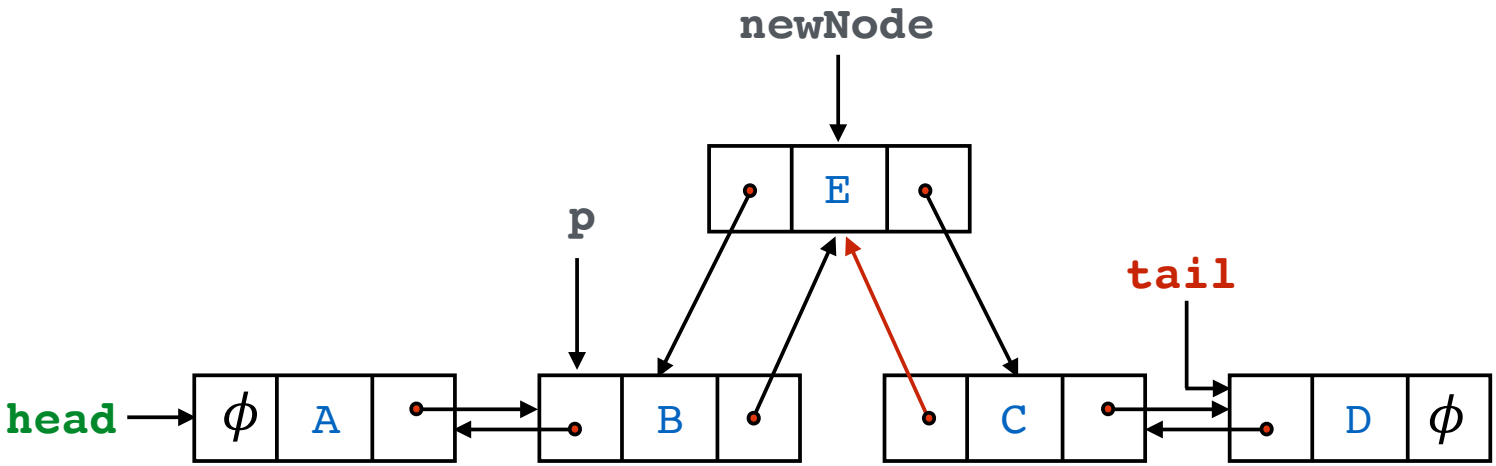
Istruzione: `p.next = newNode`

Liste doppie
Inserimento di un elemento In posizione intermedia



Istruzione: `newNode.prev = p`

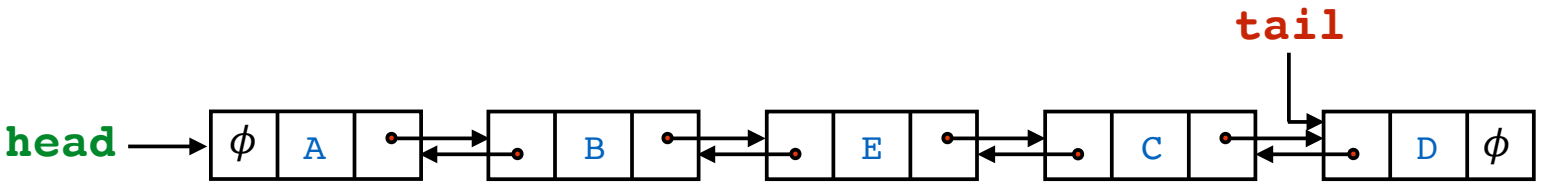
Liste doppie
Inserimento di un elemento In posizione intermedia



Istruzione: `newNode.next.prev = newNode`

Liste doppie
Inserimento di un elemento In posizione intermedia

Risultato finale:



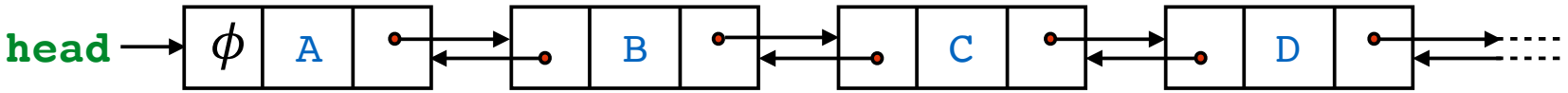
Liste doppie
Inserimento di un elemento In posizione intermedia

Codice del metodo **`insert_position`**:

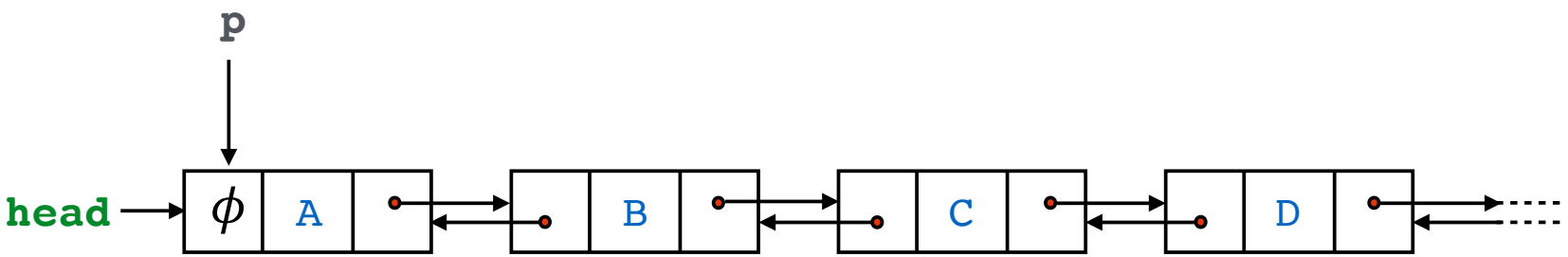
```
def insert_position(self, insertPosition, newNode):
    p = self.__head
    i = 0
    # Spostiamo il puntatore p nella posizione di inserimento
    while(p != None and (i < insertPosition - 1)):
        p = p.next
        i += 1
    newNode.next = p.next
    p.next = newNode
    newNode.prev = p
    newNode.next.prev = newNode
```

Liste doppie
Cancellazione dell'elemento i-esimo

Lista di partenza:

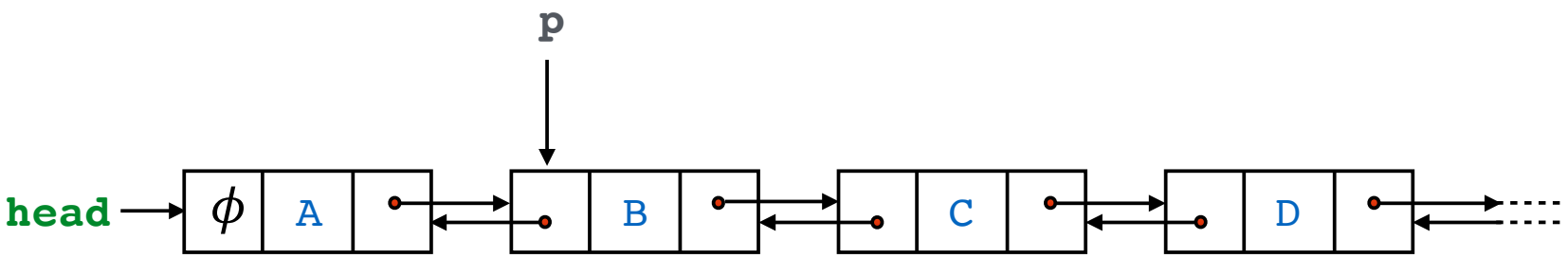


Liste doppie
Cancellazione dell'elemento i-esimo



Istruzione: `p = self.__head`

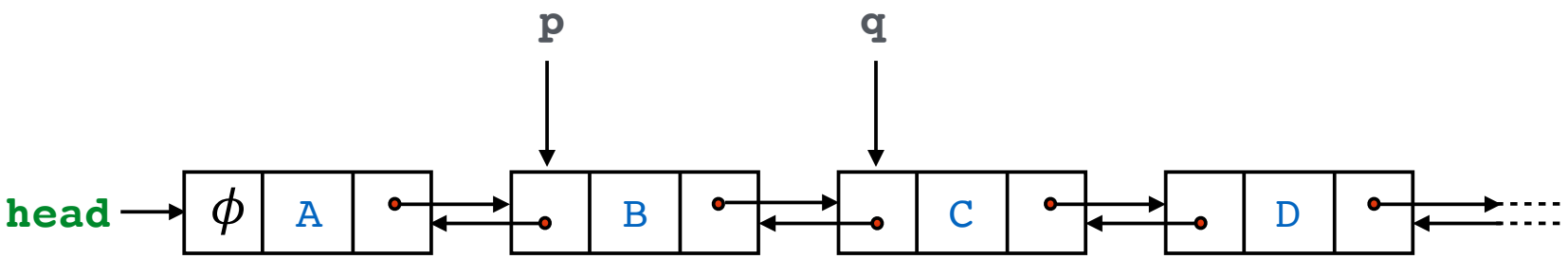
Liste doppie
Cancellazione dell'elemento i-esimo



Istruzione: `p = p.next`

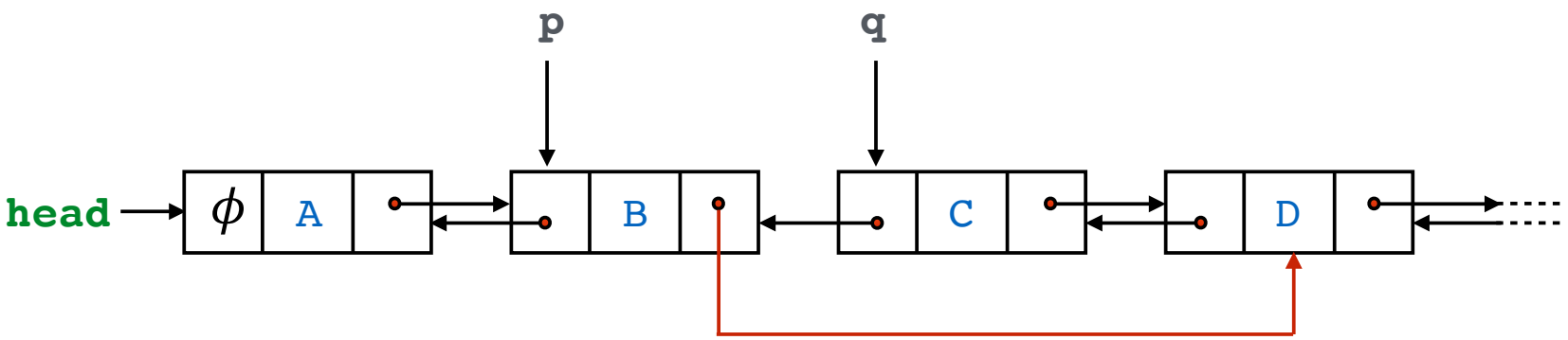
Si procede fino a raggiungere la posizione precedente quella della cancellazione. Nell'esempio: i = 2 (i parte da 0)

Liste doppie
Cancellazione dell'elemento i-esimo



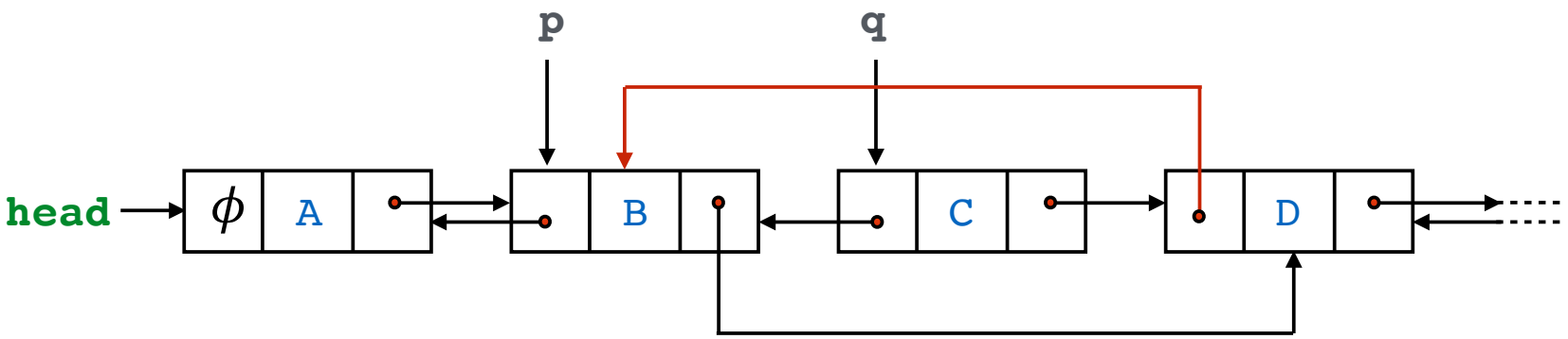
Istruzione: `q = p.next`

Liste doppie
Cancellazione dell'elemento i-esimo



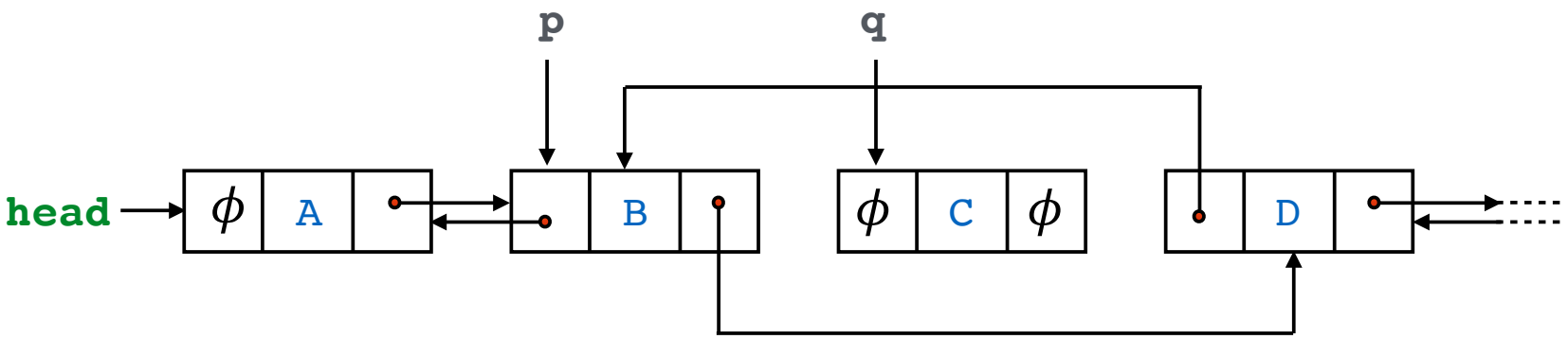
Istruzione: `p.next = q.next`

Liste doppie
Cancellazione dell'elemento i-esimo



Istruzione: `q.next.prev = p`

Liste doppie
Cancellazione dell'elemento i-esimo

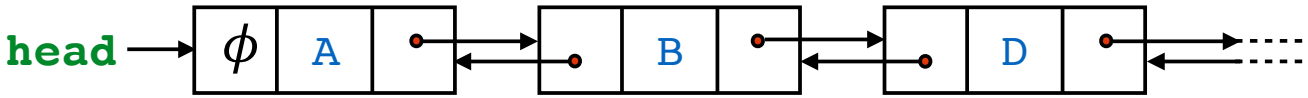


Istruzioni:

```
q.next = None
q.prev = None
```

Liste doppie
Cancellazione dell'elemento i-esimo

Risultato finale:



Liste doppie
Cancellazione dell'elemento i-esimo

Codice del metodo **`delete`**:

```
def delete(self, deletePosition):
    p = self.__head
    if p != None:
        if deletePosition == 0:
            self.__head = p.next
            p.next.prev = None
            p.next = None
            p.prev = None
        else:
            i = 0
            # Spostiamo il puntatore sul nodo immediatamente precedente
            # a quello che vogliamo rimuovere
            while(p != None and (i < deletePosition - 1)):
                p = p.next
                i += 1
            q = p.next
            p.next = q.next
            p.next.prev = p
            q.next = None
            q.prev = None
    else:
        print('Lista vuota')
```

Riferimenti

Agarwal, B., Baka, B. *Data Structures and Algorithms with Python*, Packt, 2018.

Lubanovic, B. *Introducing Python*, O'Reilly, 2020.

Horstmann, C., Necaise, R.D. *Python for Everyone*, John Wiley & Sons, 2014.

Hu, Y. *Easy learning Data Structures & Algorithms Python*, second edition, 2021.

Aho, A.V., Ullman, J.D. *Fondamenti di Informatica*, Zanichelli, 1994.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. *Introduzione agli algoritmi e strutture dati*, terza edizione, McGraw-Hill, 2010.