Corso di laurea in Informatica
Introduzione agli Algoritmi

Esercizi su liste concatenate

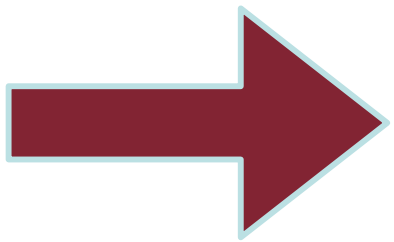
Angelo Monti

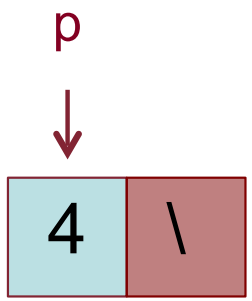
SAPIENZA
UNIVERSITÀ DI ROMA

Negli esercizi che seguono si assume sempre che i nodi delle liste coinvolte sono definiti tramite la classe seguente:

```
class Nodo:
    def __init__(self, key=None, next=None):
        self.key = key
        self.next = next
```

```
>>> p = Nodo(4)
```





Esercizio 1.

- Progettare una funzione ITERATIVA che, dato un intero n, restituisca il puntatore ad una lista concatenata di n nodi contenenti nell'ordine i valori $n, n-1, n-2, \ldots, 1$.

- Scrivere una versione RICORSIVA della funzione descritta al punto precedente

Le funzioni debbono avere costo computazionale: $\Theta(n)$

```
def es1(n):
    p= None
    for i in range(1,n+1):
        p = Nodo(i,p)
    return p


def es1b(n):
    if n == 0:
        return None
    return Nodo(n, es1b(n-1))
```

Costo computazionale: $\Theta(n)$

Esercizio 2.

Progettare una funzione ITERATIVA che, preso il puntatore alla testa di una lista concatenata e una chiave x, restituisca il numero di occorrenze di x della lista.

La funzione deve avere costo $\Theta(n)$.

Soluzione: si effettua una scansione della lista, contando le occorrenze di x via via individuate.

```
def es2(p,x):
    tot = 0
    while p:
        if p.key == x:
            tot += 1
        p = p.next
    return tot
```

Costo computazionale: $\Theta(n)$

Esercizio 2b.

Progettare una funzione RICORSIVA che, preso il puntatore alla testa di una lista concatenata e una chiave x, restituisca il numero di occorrenze di x della lista.

La funzione deve avere costo $\Theta(n)$.

IDEA:

- Se la lista è vuota, allora il numero di occorrenze è 0.
- Se la testa della lista non contiene x, allora il numero di occorrenze è dato dalle occorrenze di x nel resto della lista.
- Se la testa della lista contiene x, allora il numero di occorrenze è dato dalle occorrenze di x nel resto della lista $+1$.

```
def es2R(p,x):
    if p == None: return 0
    if p.key =! x : es2R(p.next)
    return 1 + es2R(p.next)
```

Costo computazionale: $\Theta(n)$

Esercizio 3.

- Progettare una funzione ITERATIVA che, preso il puntatore alla testa di una lista concatenata, stampi le chiavi di valore dispari presenti nella lista.

- Scrivere la versione Ricorsiva della funzione descritta nel punto precedente.

Il costo delle funzioni deve essere $\Theta(n)$.

```
def es3(p):
    while p:
        if p.key % 2 == 1:
            print(p.key, end = ' ')
        p=p.next
    print()


def es3R(p):
    if p==None:
        print()
        return
    if p.key %2:
        print(p.key, end = ' ')
    es3R(p.next)
```

Costo computazionale: $\Theta(n)$

Esercizio 4.

Progettare una funzione RICORSIVA che, preso il puntatore alla testa di una lista concatenata, stampi le chiavi dei nodi presenti nella lista in ordine inverso.

Il costo della funzione deve essere $\Theta(n)$.

Costo computazionale: $\Theta(n)$

```
def es4(p):
    if p == None:
        return
    es4(p.next)
    print(p.key)
```

Costo computazionale: $\Theta(n)$

Esercizio 5.

- Progettare una funzione ITERATIVA che, dato il puntatore alla testa di una lista concatenata, restituisca il valore dell'ultimo nodo della lista, restituisca *None* se la lista è vuota.

- Progettare la versione RICORSIVA della funzione descritta al punto precedente.

La complessità delle funzioni deve essere $\Theta(n)$.

Costo computazionale: $\Theta(n)$

```
def es5(p):
    if p == None:
        return None
    while p.next:
        p = p.next
    return p.key
```

```
def es5R(p):
    if p == None:
        return None
    if p.next:
        return es5R(p.next)
    return p.key
```

__sto computazionale: $\Theta(n)$

Esercizio 6.

- Progettare una funzione ITERATIVA che, preso il puntatore alla testa di una lista concatenata, elimini da questa lista l'ultimo nodo e restituisca il puntatore alla testa (che potrebbe essere cambiato).

- Scrivere la versione RICORSIVA della funzione al punto precedente

Le funzioni debbono avere complessità $\Theta(n)$.

Costo computazionale: $\Theta(n)$

```
def es6(p):
    if p == None or p.next==None:
        return None
    q = p
    while q.next.next != None:
        #q non è il penultimo nodo
        q = q.next
    q.next = None
    return p

def es6R(p):
    if p == None or p.next == None:
        return None
    p.next = es6R(p.next)
    return p
```

costo computazionale: $\Theta(n)$

Esercizio 7.

Progettare una funzione RICORSIVA che, preso il puntatore alla testa di una lista concatenata ed un intero x, rimuova dalla lista tutte le occorrenze di x e restituisca il puntatore alla testa della lista.

La funzione deve avere costo computazionale $\Theta(n)$.

Costo computazionale: $\Theta(n)$

```
def es7R(p, x):
    if p == None:
        return None
    if p.key == x:
        return es7R(p.next, x)
    p.next = es7R(p.next, x)
    return p
```

Costo computazionale: $\Theta(n)$

Esercizio 8.

Progettare una funzione RICORSIVA che, dato il puntatore ad una lista concatenata ed un intero x, inserisca in coda alla lista un nuovo nodo con chiave x e restituire il puntatore alla testa della lista.

La funzione deve avere complessità $\Theta(n)$.

Costo computazionale: $\Theta(n)$

```
def es8R(p, x):
    if p == None:
        nuovo = Nodo(x)
        return nuovo
    p.next = es8R(p.next, x)
    return p
```

Costo computazionale: $\Theta(n)$

Esercizio 9.

Progettare una funzione che, preso il puntatore alla testa di una lista concatenata, restituisca i puntatori alle teste di due liste concatenate, la prima con gli elementi dei nodi di posto pari nella lista di partenza, e la seconda con gli elementi dei nodi di posto dispari.

Ad esempio per la lista concatenata contenente nell'ordine i valori $1,2,3,4,5$ verranno restituiti i puntatori $p1$ e $p2$ dove $p1$ punta alla lista concatenata contenente i valori $\{1,3,5\}$ (non importa in che ordine) mentre $p2$ punta alla lista concatenata contenente i valori $\{2,4\}$ (non importa in che ordine).

La funzione non deve creare nuovi nodi e deve avere complessità $\Theta(n)$.

IDEA:

- Usiamo una variabile booleana per alternare le operazioni sui posti pari e dispari.
- Scorriamo la lista di partenza ed inseriamo ogni suo elemento alternatamente nella lista dei posti pari e in quella dei posti dispari.

```
def es9(p):
    p1, p2 = None, None
    dispari=True
    while p != None:
        q = p.next
        if dispari:
            p.next=p1
            p1 = p
        else:
            p.next=p2
            p2 = p
        dispari = not dispari
        p = q
    return p1, p2
```

Costo computazionale: $\Theta(n)$

Esercizio 10.

Progettare una funzione che, preso il puntatore alla testa di una lista concatenata, stampi tutti i valori che compaiono come chiave in almeno due nodi della lista.

Ad esempio se i nodi della lista contengono nell'ordine i valori :
4 20 10 20 5 3 8 4 20
verranno stampati i valori 4 e 20.

IDEA:

utilizziamo una funzione di servizio $conta(p, x)$ che, data una lista p ed un valore x, restituisce il numero di occorrenze di x nella lista.

- scorriamo la *lista* e stampiamo la chiave x del nodo p se e solo se $conta(p, x) == 2$

  nota che se x compare più volte in *lista* ci sarà un unico nodo p tale che nella sottolista che ha per testa p la chiave x compare due volte (in questo modo ciascun elemento che compare più di una volta verrà stampato una sola volta).

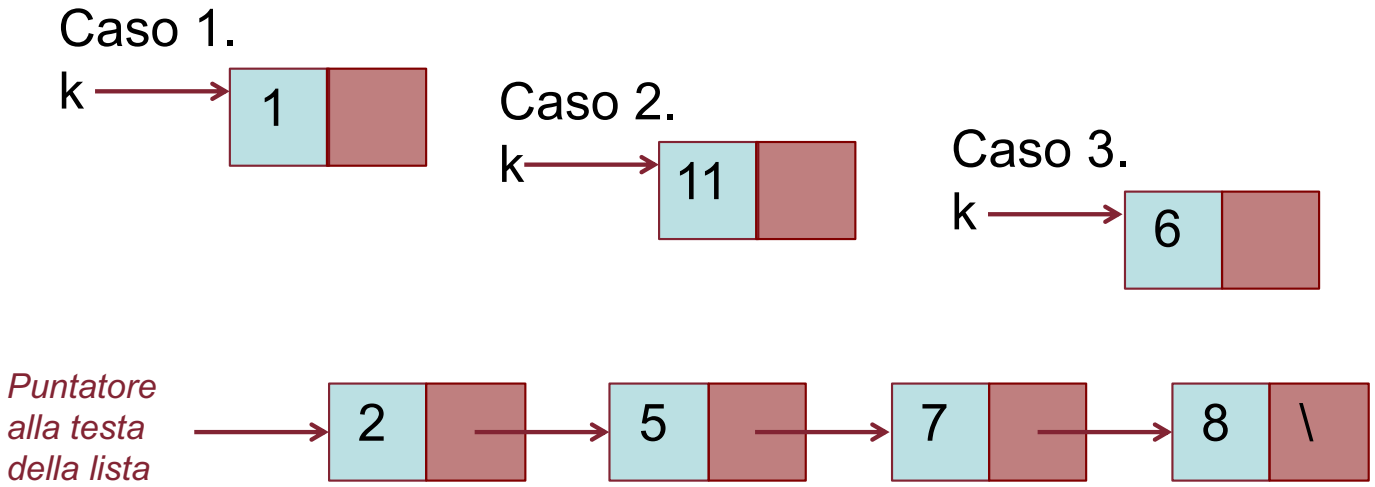
```
def es10(p):
    while p:
        if conta(p, p.key) == 2:
            print(p.key)
        p = p.next

def conta(p, x):
    if p == None: return 0
    if p.key == x: return 1 + conta(p.next, x)
    return conta(p.next, x)
```

Costo computazionale: $\Theta(n^2)$

Esercizio 11.

Progettare una funzione ITERATIVA che, preso il puntatore p alla testa di una lista concatenata ordinata e il puntatore k ad un nodo, inserisca il nodo nella lista in modo da rispettare l'ordine delle chiavi e restituisca il puntatore (eventualmente cambiato) alla testa della lista.

Ad esempio:



nel caso uno l'elemento finisce in testa alla lista e la testa della lista cambia. Nel caso due l'elemento finisce tra i nodi della lista e nel caso tre finisce in coda alla lista in entrambi questi casi la testa della lista non cambia.

```
def es11(p, k):
    if p == None:
        return k
    if k.key ≤ p.key:
        #l'elemento va al primo posto
        k.next = p
        return k
    q = p
    while q.next != None AND q.next.key <= k.key:
        q =q.next
    if q.next==None:
        # l'elemento va posizionato all'ultimo posto
        q.next=k
    else:
        # l'elemento va posizionato tra q e q.next
        k.next = q.next
        q.next = k
    return p
```

Costo computazionale: $O(n)$

Esercizio 11b.

Progettare una funzione RICORSIVA che, preso il puntatore p alla testa di una lista concatenata ordinata e il puntatore k ad un nodo, inserisca il nodo nella lista in modo da rispettare l'ordine delle chiavi e restituisca il puntatore (eventualmente cambiato) alla testa della lista.

La funzione deve avere complessità $O(n)$

Costo computazionale: $O(n)$

```
def es11R(p, k):
    if p == None or k.key <= p.key:
        # k deve diventare  la testa della lista
        k.next = p
        return k
    p.next = es11R(p.next,k)
    return p
```

Costo computazionale: $O(n)$

Esercizio 12.

Progettare una funzione che, preso il puntatore alla testa di una lista concatenata, restituisca il puntatore alla lista con i nodi ordinati (senza creare nuovi record).

Ad esempio:

se i valori dei nodi che compaiono nella lista concatenata sono nell'ordine:

6 8 9 2 1 5 3 11 13 12

verrà restituito il puntatore alla lista concatenata che ha gli stessi nodi ma riordinati in modo che i valori compaiono nell'ordine:

1 2 3 5 6 8 9 11 12 13.

IDEA:

- Adattiamo il Selection Sort.

- Ad ogni iterazione selezioniamo il massimo, lo stacchiamo dalla lista e lo inseriamo in testa ad una nuova lista ordinata.

Nota: anche se durante le iterazioni coesistono due liste distinte non viene allocata nuova memoria, semplicemente i record vengono spostati dalla prima alla seconda lista.

```
def selectionSort(p):
    p_ordinata = None
    while p! = None:
        p, nodo_massimo = estraiMax(p)
        nodo_massimo.next = p_ordinata
        p_ordinata = nodo_massimo
    return p_ordinata
```

Resta da scrivere la funzione $estraiMax(p)$ che, in tempo $\Theta(n)$, estrae dalla lista concatenata non vuota puntata da p un nodo con chiave massima e lo restituisce insieme al puntatore alla testa della lista modificata.

Costo computazionale: $\Theta(n^2)$

```
def estraiMax(p):
    nodo_massimo = q = p
    while q:
        if q.key > nodo_massimo.key:
            nodo_massimo = q
        q = q.next
    if nodo_massimo == p:
        return p.next, nodo_massimo
    q = p
    while q.next != massimo:
        q = q.next
    q.next = massimo.next
    return p, massimo
```

Costo computazionale: $\Theta(n)$