Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Esercizi su liste concatenate

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

Sulla base delle slides a cura di T. Calamoneri e G. Bongiovanni per il corso di informatica generale AA 2019/2020

Esercizi sulle liste concatenate.
Assumiamo 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. Scrivere una funzione ITERATIVA che, preso in input un intero n, restituisce il puntatore ad una lista concatenata di n nodi contenenti nell'ordine i valori $n, n-1, n-2, \ldots, 1$.

```
def creaLista(n):
    if n==0: return None
    listaP=Nodo(n)
    p=listaP
    while True:
        n-=1
        if n==0: return listaP
        p.next=Nodo(n)
        p=p.next
```

Costo computazionale: $\Theta(n)$

Esercizio 1b. Scrivere una funzione RICORSIVA che, preso in input un intero n, restituisce il puntatore ad una lista concatenata di n nodi contenenti nell'ordine i valori $n, n-1, n-2, \ldots, 1$.

```
def creaLista_Ric(n):
    if n==0: return None
    p=Nodo(n)
    p.next=creaLista_Ric(n-1)
    return p
```

Costo computazionale: $\Theta(n)$

Esercizio 2. Scrivere una funzione ITERATIVA che, preso in input il puntatore alla testa di una lista concatenata e una chiave x restituisce lil numero di occorrenze di x della lista.

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

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

Costo computazionale: $\Theta(n)$

Esercizio 2b. Scrivere una funzione RICORSIVA che, preso in input il puntatore ad una lista concatenata ed una chiave x restituirsce il numero di occorrenze di x nella lista.

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

Costo computazionale: $\Theta(n)$

Esercizio 3. Scrivere una funzione RICORSIVA che preso in input il puntatore alla testa di una lista concatenata stampa le chiavi dei nodi presenti nella lista.

```
def stampa(p):
    while p:
        print(p.key, end='  ')
        p = p.next
    print()
```

Costo computazionale: Θ(n)

Esercizio 4. Scrivere una funzione RICORSIVA che preso in input il puntatore alla testa di una lista concatenata stampa le chiavi dei nodi presenti nella lista in ordine inverso.

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

Costo computazionale: $\Theta(n)$

Esercizio 5. Scrivere una funzione ITERATIVA che dato in input il puntatore alla testa di una lista concatenata restituisce il valore dell'ultimo nodo della lista, restituisce *None* se la lista è vuota.

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

Costo computazionale: $\Theta(n)$

Esercizio 5b. Scrivere una funzione RICORSIVA che preso in input il puntatore alla testa di una lista concatenata restituisce il valore dell'ultimo nodo della lista, restituisce *None* se la lista è vuota.

IDEA: se l'elemento ispezionato ha almeno un successivo, allora l'ultimo elemento si trova nel resto della lista. Altrimenti è l'elemento stesso.

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

Costo computazionale:

- T(n) = T(n-1) + Θ(1)
- T(0) = T(1) = Θ(1)

Costo computazionale: $\Theta(n)$

Esercizio 6. Scrivere una funzione ITERATIVA che preso in input il puntatore alla testa di una lista concatenata elimina da questa l'ultimo nodo.

IDEA: ricordiamo che, come regola generale, per poter eliminare un elemento si deve conoscere anche il puntatore dell'elemento che lo precede nella lista, per poter aggiornare il suo campo `next`.

```
def rimuoveUltimo(p):
    if p==None or p.next==None: return None
    q=p
    while q.next.next!=None:
        q= q.next
    q.next=None
    return p
```

Costo computazionale: $\Theta(n)$

Esercizio 6b. Scrivere una funzione RICORSIVA che preso in input il puntatore alla testa di una lista concatenata elimina da questa l'ultimo nodo.

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

Costo computazionale:

- $T(n) = T(n-1) + \Theta(1)$
- $T(0) = T(1) = \Theta(1)$

Costo computazionale: $\Theta(n)$

Esercizio 7. Scrivere una funzione RICORSIVA che preso in input il puntatore alla testa di una lista concatenata non vuota ed un intero x, rimuovere dalla lista le occorrenze di x e restituisce la testa della lista.

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

Costo computazionale: $\Theta(n)$

Esercizio 8a. Data in input una lista concatenata ed un intero x, inserire in testa alla lista un nuovo nodo con chiave x.

```
def inserireInTesta(lista,x):
    nuovo=Nodo(x)
    nuovo.next=lista
    return nuovo
```

Costo computazionale: $\Theta(1)$

Esercizio 8. Data in input una lista concatenata ed un intero x, inserire in coda alla lista un nuovo nodo con chiave x.

```
def inserireInCodaR(lista,x):
    if lista==None:
        nuovo=Nodo(x)
        return nuovo
    lista.next=inserireCodaR(lista.next,x)
    return lista
```

Costo computazionale: $\Theta(n)$

Esercizio 9. Scrivere una funzione che preso in input il puntatore alla testa di una lista concatenata restituisce 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 (senza creare nuovi record).

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.
- Alla fine assegniamo $None$ a *alla lista di partenza*, dato che la lista originale è stata svuotata, e lo restituiamo assieme agli altri due puntatori.

Ad esempio per la lista concatenata contenente nell'ordine i valori $1, 2, 3, 4, 5, 6\, e\, 7$ verranno restituiti i puntatori $p1$ e $p2$ dove $p1$ punta alla lista concatenata contenente i valori $1, 3, 5, e\, 7$ (non importa in che ordine) mentre $p2$ punta alla lista concatenata contenente i valori $2, 4, e\, 6$ (non importa in che ordine.

```
def separa(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. Scrivere una funzione che preso in input il puntatore alla testa di una lista concatenata stampa 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 funziona $conta(p, x)$ che data una lista p ed un valore x ci dice 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).

versioni iterativa e ricorsiva della funzione conta:

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

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

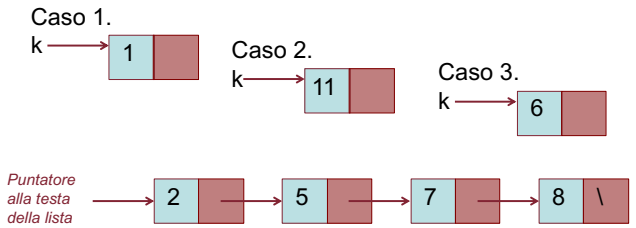
Costo computazionale: $\Theta(n)$

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

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

Esercizio 11. Scrivere una funzione ITERATIVA che preso in input il puntatore p alla testa di una lista concatenata ordinata e il puntatore k ad un nodo inserisce il nodo nella lista in modo da rispettare l'ordine e restituisce 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. Negli altri due casi la testa della lista non cambia: nel caso due l'elemento finisce tra i nodi della lista e nel caso tre finisce in coda alla lista.

```
def inserisci_inOrd(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
```

Esercizio 11b. Scrivere una funzione RICORSIVA che preso in input il puntatore p alla testa di una lista concatenata ordinata e il puntatore k ad un nodo inserisce il nodo nella lista in modo da rispettare l'ordine e restituisce il puntatore (eventualmente cambiato) alla testa della lista.

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

la funzione $estraiMax(p)$ estrae dalla lista concatenata non vuota e con elementi distinti puntata da p il nodo con chiave massima e lo restituisce insieme al puntatore della lista modificata.

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

Costo computazionale: $\Theta(n)$

Esercizio 12. Scrivere una funzione che preso in input il puntatore alla testa di una lista concatenata restituisce 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, massimo=estraiMax(p)
        massimo.next=p_ordinata
        p_ordinata=massimo
    return p_ordinata
```

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