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

Esercizio svolto (1)

Esercizio. Si vuole simulare una coda tramite due pile. In particolare, si possono usare –senza dettagliarle- le seguenti funzioni:

- `inserisciPila`(x, p)
- `estraiPila`(p)
- `test_di_pila_vuota`(p)

dove p è la pila sulla quale si eseguono le operazioni.

Descrivere le operazioni di `inserisciCoda(x)` ed `estraiCoda()` con le 2 pile.

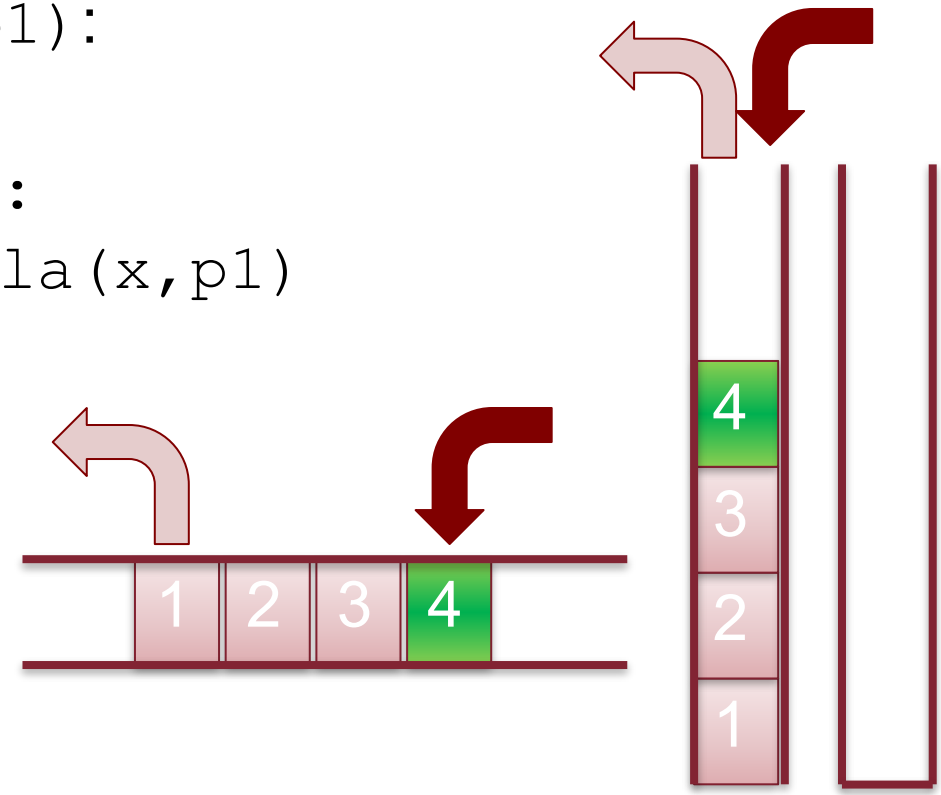
Esercizio svolto (2)

Soluzione. Utilizziamo una pila $p1$ per memorizzare i dati ed una pila $p2$ come appoggio.

Abbiamo molti modi di procedere; qui scegliamo quello in cui `inserisciCoda(x)` coincide con `inserisciPila(x,p1)`:

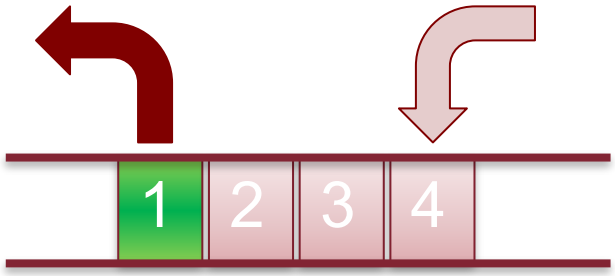
```
def inserisciCoda(x):
    p1=inserimentoPila(x,p1)
    return p1
```

Costo: $\Theta(1)$



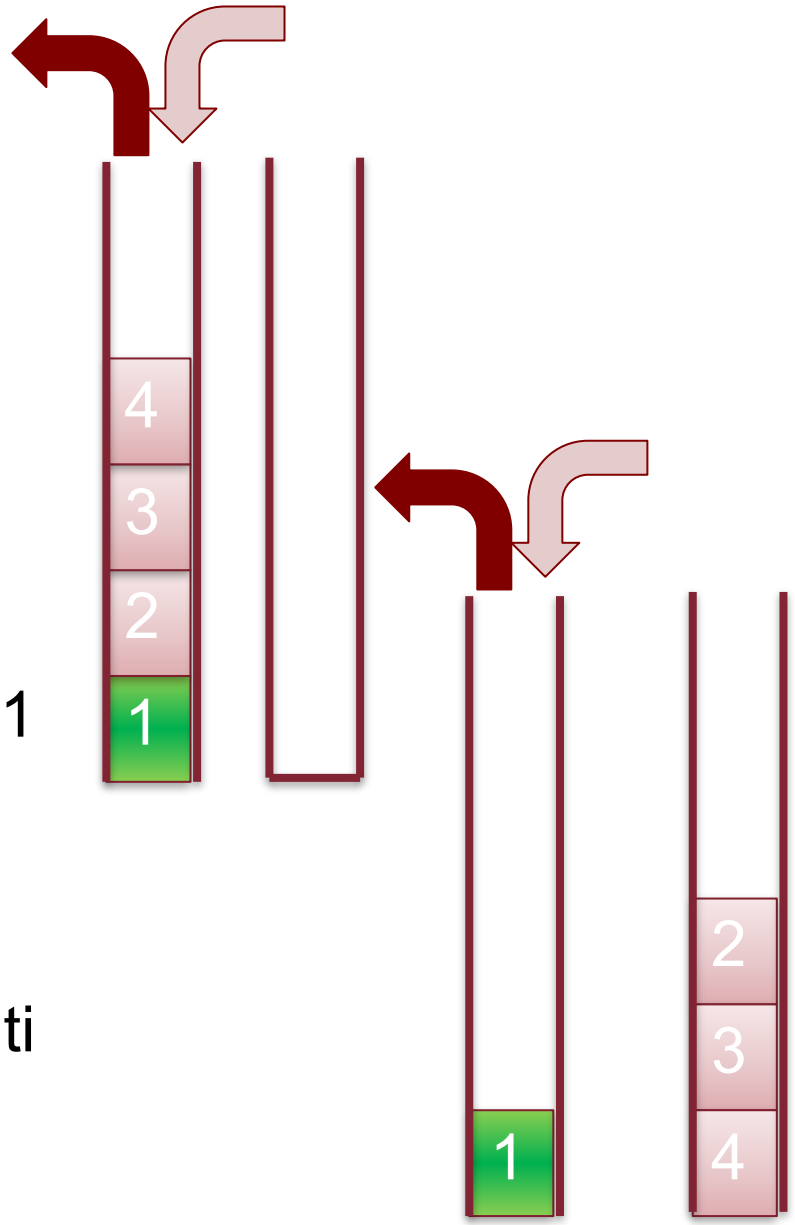
Esercizio svolto (3)

Per la `estraiCoda()`:



Dobbiamo rovesciare la pila 1 per estrarre l'ultimo elemento

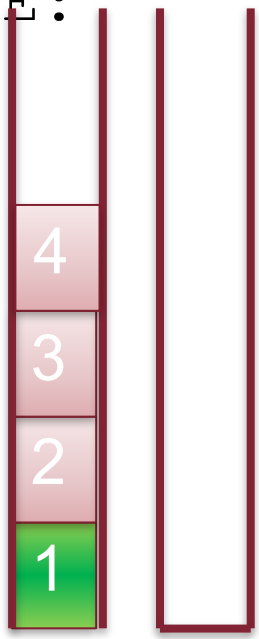
e poi riportare gli elementi indietro…



Esercizio svolto (4)

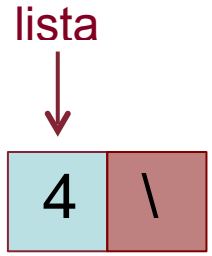
```
def estraiCoda():
    while test_di_pila_vuota(p1)==FALSE:
        p1,x=estraiPila(p1)
        p2=inserisciPila(x,p2)
    p2,aux=estraiPila(p2)
    while  test_di_pila_vuota(p2)==FALSE:
        p2,x=estraiPila(p2)
        p1=inserisciPila(x,p1)
    return p1,aux
```

Costo: $\Theta(n)$

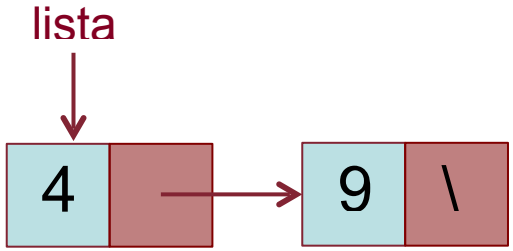


```
class Nodo:
    def __init__(self, key=None, next=None):
        self.key = key
        self.next = next
```

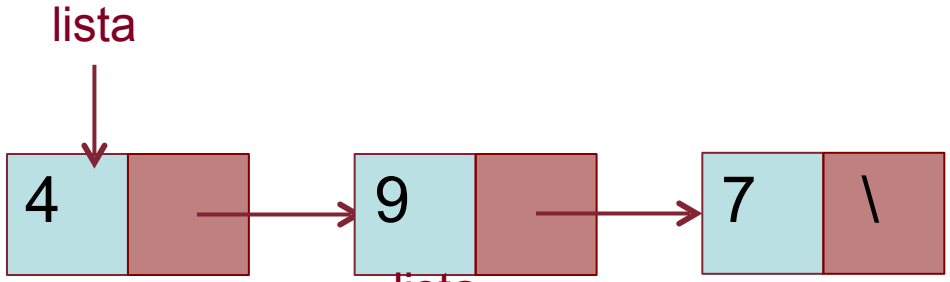
```
>>> lista=Nodo(4)
```



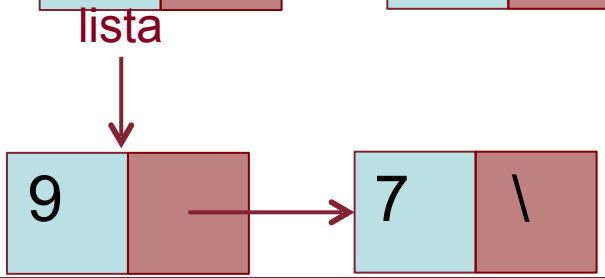

```
>>> lista.next=Nodo(9)
```




```
>>> lista.next.next=Nodo(7)
```




```
>>> lista=lista.next
```



Esercizio 1. Dato in input un intero n restituire 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 2. Data in input una lista concatenata, restituire la sua lunghezza.

Soluzione: si fa una scansione della lista, contando i nodi attraversati.

```
def  lunghezza(listaP):
    n=0
    while listaP:
        n+=1
        listaP=listaP.next
   return n
```

```
>>> lista=creaLista(5)
>>> lunghezza(lista)
5
```

Costo computazionale: $\Theta(n)$

Esercizio 2- soluzione ricorsiva

Esercizio. Data in input una lista concatenata restituire la sua lunghezza.

IDEA: se la lista ha almeno un elemento allora la sua lunghezza è $1+$ la lunghezza della lista privata del primo elemento.

```
def lunghezzaR(listaP):
    if listaP==None: return 0
    return 1+ lunghezzaR(listaP.next)
```

```
>>> lista=creaLista(6)
>>> lunghezzaR(lista)
6
```

Costo computazionale: $\Theta(n)$

Esercizio 3. Dato in input una lista concatenata stampare le chiavi.

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

```
>>> lista=creaLista(7)
>>> stampa(lista)
7  6  5  4  3  2  1
```

Costo computazionale: Θ(n)

Esercizio 4. Data in input una lista concatenata stampare la lista a partire dall'ultimo elemento.

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

```
>>> >>> stampa(lista)
7  6  5  4  3  2
>>> stampaInversaR(lista)
2  3  4  5  6  7
```

Costo computazionale: $\Theta(n)$

Esercizio 5. Data in input una lista concatenata restituire il valore dell'ultimo nodo.

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

```
>>> stampa(lista)
7  6  5  4  3  2  1
>>> ultimo(lista)
1
```

Costo computazionale: $\Theta(n)$

Esercizio svolto – soluzione ricorsiva

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

Costo computazionale:

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

Costo computazionale: $\Theta(n)$

Esercizio 6. Data in input una lista concatenata eliminare da questa l'ultimo elemento.

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`.

Esercizio 6. Data in input una lista concatenata eliminare da questa l'ultimo elemento.

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

```
>>> stampaLista(lista)
5  4  3  2  1
>>> lista=rimuoveUltimo(lista)
>>> stampaLista(lista)5  4  3  2
```

Costo computazionale: $\Theta(n)$

Esercizio svolto – versione ricorsiva

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

Costo computazionale:

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

Costo computazionale: $\Theta(n)$

Esercizio 7. Data in input una lista concatenata non vuota ed un intero x, rimuovere dalla lista le occorrenze di x.

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

```
>>> stampa(lista)
3  8  9  2  1  5  3  11  13  3
>>> lista=rimuoviR(lista,3)
>>> stampa(lista)
8  9  2  1  5  11  13
```

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. Data in input una lista concatenata, restituire due liste concatenate, una con gli elementi di posto pari nella lista di partenza, ed una con gli elementi 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.

Esempio per di comportamento la funzione *separa*(*lista*) **richiesta:**

```
>>> lista=creaLista(7)
>>> lista, listaD,listaP=separa(lista)


>>> stampaLista(lista)

>>> stampaLista(listaD)
1   3   5   7
>>> stampaLista(listaP)
2   4   6
```

```
def separa(lista):
    listaD, listaP=None, None
    dispari=True
    while lista!=None:
        p=lista.next
        if dispari:
            lista.next=listaD
            listaD=lista
        else:
            lista.next=listaP
            listaP=lista
        dispari=not dispari
        lista=p
    return lista, listaD, listaP
```

Costo computazionale: $\Theta(n)$

Esercizio 10. Data in input una lista di interi stampare tutti i valori che compaiono almeno due volte nella lista.

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(lista, x):
    c=0
    while lista:
        if lista.contenuto==x:
            c+=1
        lista=lista.next
    return c

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

Costo computazionale: $\Theta(n)$

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

>>> stampaLista(a)
8  4  20  10  20  5  3  8
>>> stampaRipetuti(a)
20 8
```

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

Esercizio 11. Data in input una lista concatenata di interi restituire la lista ordinata (senza creare nuovi record).

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.

la funzione *estraiMax(lista)* estrae dalla *lista* concatenata il record con chiave massima e lo restituisce insieme alla lista modificata.

```
def estraiMax(lista):
    if lista==None: return None, None
    if lista.next==None: return None, lista
    precMassimo, massimo=None,lista
    p=lista
    while p.next!=None:
        if p.next.key>massimo.key:
            massimo=p.next
            precMassimo=p
        p=p.next
    if precMassimo==None:
        lista=lista.next
    else:
        precMassimo.next=massimo.next
    return lista, massimo
```

Costo computazionale: $\Theta(n)$

```
def selectionSort(lista):
    listaOrdinata=None
    while lista!=None:
        lista, massimo=estraiMax(lista)
        massimo.next=listaOrdinata
        listaOrdinata=massimo
    return listaOrdinata
```

```
>>> stampaLista(lista)
6  8  9  2  1  5  3  11  13  12
>>> lista=selectionSort(lista)
>>> stampaLista(lista)
1  2  3  5  6  8  9  11  12  13
```

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