Liste Concatenate

Algoritmi 1 – Lezione 10

A. Monti
Sapienza Università di Roma

Corso: Algoritmi 1

Indice

Strutture Statiche vs Dinamiche

Strutture Statiche (es. array)

- Dimensione fissa decisa all'inizio
- Memoria allocata in blocco contiguo
- Accesso diretto agli elementi tramite indice `A[i]`
- **Limite:** difficili da ridimensionare

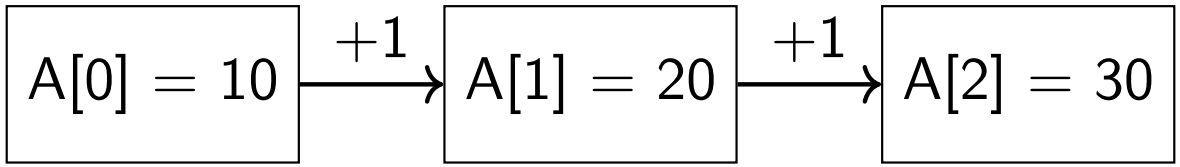
Strutture Dinamiche (es. liste concatenate)

- Crescono o si riducono durante l'esecuzione
- Ogni elemento può trovarsi ovunque in memoria
- Accesso sequenziale seguendo i riferimenti
- **Limite:** accesso più lento e overhead di puntatori

Navigazione negli Array

In un array, gli elementi sono memorizzati in modo contiguo.

- Ogni elemento ha la stessa dimensione
- Si accede direttamente con lindice: `A[i]`
- Lo spostamento tra elementi è un semplice calcolo sull'indirizzo iniziale

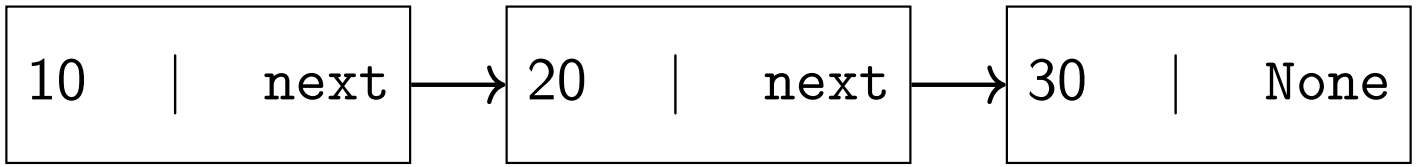


Accesso diretto: tempo O(1) per raggiungere l'elemento i-esimo

Navigazione nelle Liste Concatenate

In una lista concatenata, gli elementi (nodi) non sono contigui.

- Ogni nodo contiene:
  - un valore
  - un riferimento al nodo successivo (*next*)
- Per attraversare la lista si parte dalla testa e si segue *next*



Accesso sequenziale: tempo $O(i)$ per raggiungere il nodo i-esimo

Array vs Lista Concatenata

Caratteristica	**Array (statico)**	**Lista concatenata (dinamica)**
Memoria	Blocchi contigui	Nodi sparsi in memoria
Accesso	Diretto: `A[i]`	Sequenziale: tramite *next*
Spostamento	Offset di memoria (calcolo)	Seguendo i riferimenti
Inserimento/Rimozione	Costoso (richiede copia)	Veloce (se hai il riferimento)

- Gli array sono più performanti grazie all'accesso diretto.
- Le liste concatenate offrono flessibilità nella modifica della struttura.

Le Liste di Python: Statiche o Dinamiche?

Che tipo di struttura è una `list` in Python?

Risposta: un array dinamico

- Memoria contigua e accesso diretto con `A[i]`
- Cresce automaticamente quando serve

Attenzione: Le liste Python **non sono** liste concatenate. Sono allocate in un blocco unico e, quando esauriscono lo spazio, Python:

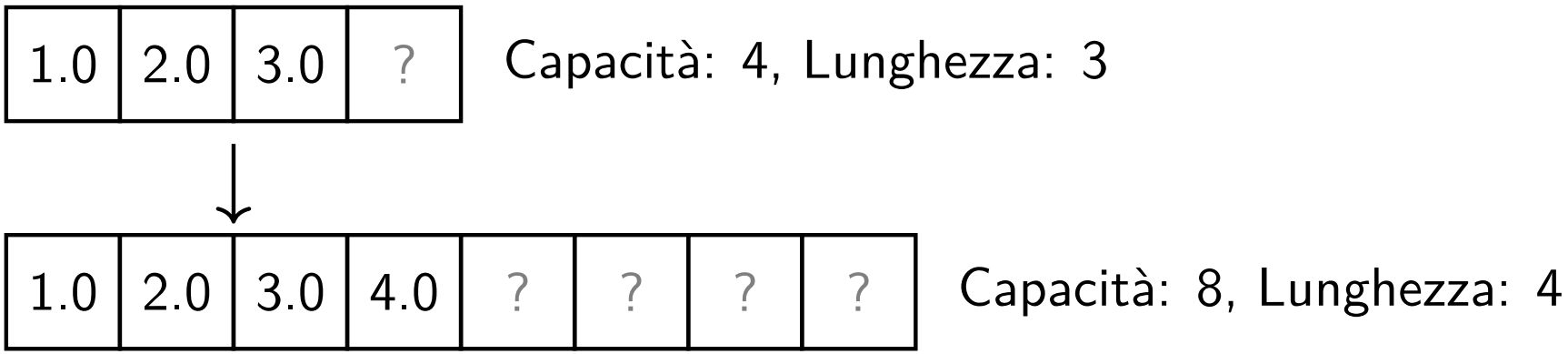
- Alloca un array più grande
- Copia tutti gli elementi nel nuovo array

- **Vantaggi:** flessibilità + accesso veloce
- **Svantaggi:** riallocazioni costose in certi casi.

Riallocazione automatica delle liste Python

Le liste Python usano un array sottostante con capacità extra.

Esempio: aggiunta di un elemento a una lista piena



- **Python rialloca solo quando necessario.**
- *La crescita è "a salti", per ridurre il numero di copie.*

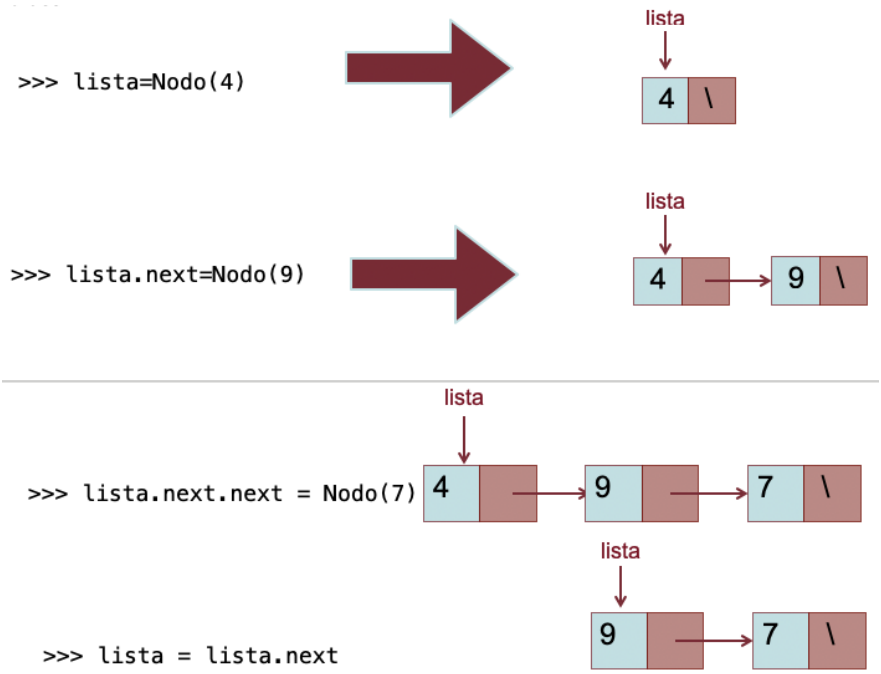
Le liste Python sono un ibrido: dinamiche ma basate su array.

Liste Concatenate in Python: Definizione e Collegamento

Un nodo si rappresenta con una classe:

```
class Nodo:
    def __init__(self, key=None, next=None):
        self.key = key          # Valore del nodo
        self.next = next        # Riferimento al prossimo nodo
```

Possiamo creare e collegare nodi manualmente:



La variabile `lista` punta al primo nodo della catena.

Esempio 1: Creazione di una Lista Concatenata (1/2)

Dato un array di interi, creare una lista concatenata che li contenga e restituire la testa della lista.

Ecco la versione iterativa:

```
def Crea(A):
    if A == []:
        return None
    p = Nodo(A[0])
    # q punta all'ultimo elemento della lista
    q = p
    for i in range(1, len(A)):
        # aggiungi l'elemento dell'array  in coda alla lista
        q.next = Nodo(A[i])
        q = q.next
    return p
```

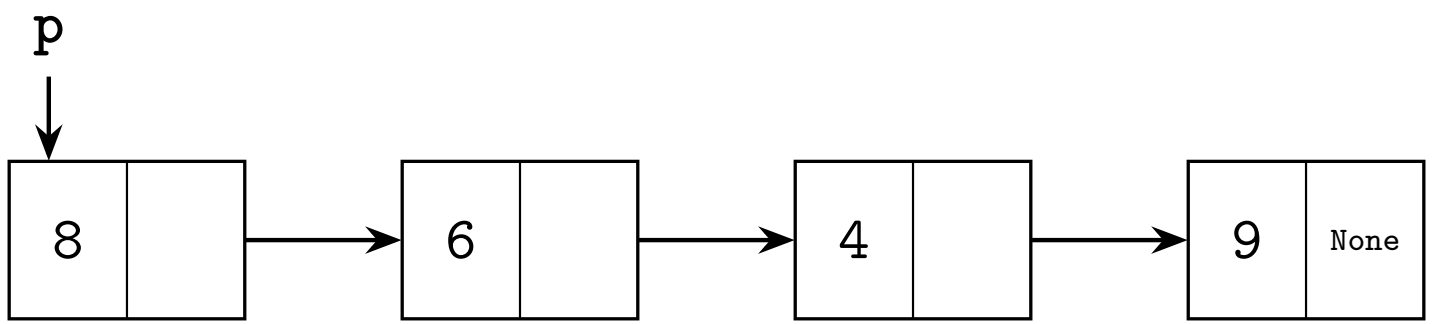
Esempio 1: Creazione di una Lista Concatenata (2/2)

Ecco la versione ricorsiva:

```
def CreaR(A, i=0):
    if i == len(A):
        # Caso base: fine dell'array
        return None
    # Costruisce ricorsivamente la coda della lista
    # e poi crea il nodo corrente
    q = CreaR(A, i + 1)
    return Nodo(A[i], q)
```

Entrambe le versioni hanno complessità temporale $\Theta(n)$ dove n è il numero di elementi dell'array.

Invocate su `A = [8,6,4,9]` producono la lista concatenata seguente:



Esempio 2: Stampa di una Lista Concatenata

Data la testa p di una lista, stamparne gli elementi.

Versione iterativa:

```
def Stampa(p):
    while p:
        print(p.key)
        p = p.next
```

Versione ricorsiva:

```
def StampaR(p):
    if p is None:
        return
    print(p.key)
    StampaR(p.next)
```

Entrambe le versioni hanno complessità temporale $\Theta(n)$.
La versione iterativa usa spazio $O(1)$, la ricorsiva spazio $O(n)$ per lo stack delle chiamate.

Esercizio 3: Stampa Inversa di una Lista Concatenata

Dato il puntatore p alla testa di una lista concatenata, stamparne gli elementi in ordine inverso.

Versione iterativa:

```
def StampaReverse(p):
    A = []
    while p is not None:
        A.append(p.key)
        p = p.next
    while A:
        print(A.pop())
```

Versione ricorsiva:

```
def StampaReverseR(p):
    if p is None:
        return
    StampaReverseR(p.next)
    print(p.key)
```

Entrambe le versioni hanno complessità temporale $\Theta(n)$.
La versione iterativa usa spazio di lavoro $O(n)$ per la lista temporanea;
La versione ricorsiva spazio di lavoro $O(n)$ per lo stack.

Esempio 4: Inserimento in Lista Ordinata (1/1)

Dato il puntatore p alla testa di una lista concatenata ordinata in modo crescente ed un valore x, scrivere una funzione che inserisca x nella lista, mantenendo lordine.
Il valore x deve essere inserito solo se non è già presente nella lista.
La funzione deve restituire la testa della lista, eventualmente modificata.

Versione iterativa:

```
def Inserisci(p, x):
    # Caso speciale: inserimento in testa
    if p is None or p.key > x:
        return Nodo(x, p)
    q = p
    while q.next is not None and q.next.key < x:
        q = q.next
    # Inserisce solo se x non è già presente
    if q.next is None or q.next.key > x:
        q.next = Nodo(x, q.next)
    return p
```

Esempio 4: Inserimento in Lista Ordinata (2/2)

Versione ricorsiva:

```
def InserisciR(p, x):
    if p is None or p.key > x:
        # Caso base: lista vuota o x va inserito prima del nodo corrente
        # Crea un nuovo nodo e lo collega al resto della lista
        return Nodo(x, p)
    if p.key == x:
        # Caso in cui x è già presente: non si inserisce
        # Nessuna modifica alla lista
        return p
    # Caso ricorsivo: prosegue con il resto della lista
    # Collega il risultato ricorsivo al nodo corrente
    p.next = InserisciR(p.next, x)
    # Restituisce la testa della lista (possibilmente non modificata)
    return p
```

Entrambe le versioni hanno complessità $O(n)$, poiché nel caso peggiore, quando l'elemento da inserire è maggiore del massimo della lista e va inserito in coda, è necessario scorrere tutta la lista per determinare la posizione corretta.

Esempio 5: Cancellazione da Lista Concatenata (1/2)

Dato il puntatore p alla testa di una lista concatenata e un valore x, scrivere una funzione che elimini dalla lista la prima eventuale occorrenza di x. La funzione deve restituire la testa della lista, eventualmente modificata.

Versione iterativa:

```
def Cancella(p, x):
    if p is None:
        # Caso speciale: lista vuota
        return None
    if p.key == x:
        # Caso in cui il valore da eliminare è in testa
        return p.next
    q = p
    while q.next is not None:
        if q.next.key == x:
            # Nodo trovato: salta il nodo da eliminare
            q.next = q.next.next
            return p
        q = q.next
    # x non trovato
    return p
```

Esempio 5: Cancellazione da Lista Concatenata (2/2)

Versione ricorsiva:

```
def CancellaR(p, x):
    if p is None:
        # Caso base: lista vuota
        return None
    if p.key == x:
        # Nodo da eliminare trovato in testa
        return p.next
    # Ricorsione sul nodo successivo
    p.next = CancellaR(p.next, x)
    return p
```

Complessità temporale: $O(n)$ Nel caso peggiore, l'elemento non è presente e bisogna scorrere lintera lista.

Complessità spaziale:

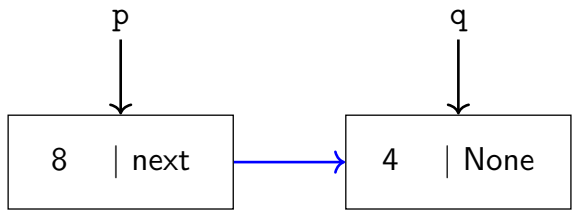
- **Versione iterativa:** $O(1)$ (spazio costante).
- **Versione ricorsiva:** $O(n)$ (profondità della ricorsione).

Gestione automatica della memoria (Garbage Collection)

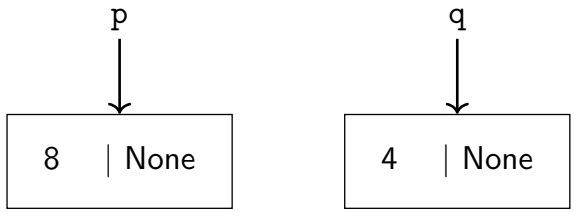
In Python, un oggetto (es. un nodo) viene eliminato solo quando nessun riferimento lo punta più.

Esempio: La lista contiene due nodi. Il puntatore q punta ancora al nodo 4, anche dopo che è stato scollegato dalla lista con $p.next = None$.

Prima di $p.next = None$**:**



Dopo $p.next = None$**:**



Il nodo con chiave 4 è scollegato da p ma resta in memoria perché ancora puntato da q.
In Python, la memoria occupata da un oggetto viene liberata **automaticamente** quando **nessun riferimento** punta più a quell'oggetto.

ESERCIZI

Suggerimento: Prima di cercare le soluzioni in rete, prova a ragionarci da solo: gli esercizi che seguono sono pensati per aiutarti a mettere alla prova la tua comprensione, la tua capacità di sviluppare una prova di correttezza e costruire un algoritmo corretto.
Solo dopo averci riflettuto, confronta la tua soluzione con altre possibili versioni.

Esercizi

1. Progetta una funzione che, data la testa di una lista concatenata, calcoli la somma di tutti i valori presenti nella lista.
   Scrivi sia una versione iterativa che una versione ricorsiva della funzione e valuta la complessità temporale e spaziale degli algoritmi proposti.
2. Progetta una funzione che, data la testa di una lista concatenata, restituisca i puntatori alla testa di due liste: una contenente gli elementi in posizione pari e l'altra quelli in posizione dispari nella lista di partenza. Non è consentito creare nuovi nodi.
   La complessità temporale della funzione deve essere $O(n)$ e quella spaziale $O(1)$.
3. Progetta una funzione che, dati i puntatori alle teste di due liste concatenate i cui nodi sono ordinati in ordine crescente, simuli l'**operazione di fusione** delle due liste, fondendo i loro elementi in un'unica lista anch'essa ordinata in ordine crescente. La funzione deve restituire il puntatore alla testa della nuova lista risultante. Non è consentito creare nuovi nodi. Valutare la complessità della funzione proposta.
4. Progetta una funzione che, data la testa di una lista concatenata, divida la lista in due metà. Se la lista ha un numero dispari di nodi, la seconda metà dovrebbe avere un nodo in più. La funzione deve restituire i puntatori alla testa delle due liste risultanti.
   La complessità temporale della funzione deve essere $O(n)$ e quella spaziale $O(1)$.

Esercizi

1. Progetta una funzione che, data la testa di una lista concatenata e un intero x, restituisca una coppia (c, n), dove:
   - c è il numero di occorrenze di x nella lista;
   - n è il numero totale di nodi presenti nella lista.

   Scrivi sia una versione iterativa che una versione ricorsiva della funzione e valuta le loro complessità temporali e spaziali.
2. Progetta una funzione che, data la testa di una lista concatenata e un intero k, ruoti la lista di k posizioni verso destra. La funzione deve restituire il puntatore alla testa della lista modificata ed avere complessità temporale $O(n)$.
3. Progetta una funzione che, data la testa di una lista concatenata, restituisca il puntatore alla testa di una nuova lista che contiene gli stessi nodi ma in ordine inverso. La funzione proposta deve avere complessità spaziale $O(1)$.
4. Una lista è palindroma se la sequenza di elementi è la stessa sia in avanti che all'indietro.
   - Progetta una funzione che, data la testa di una lista doppiamente concatenata, verifichi se la lista è palindroma.
     La funzione proposta deve avere complessità spaziale $O(1)$ e complessità temporale $O(n)$.
   - Progetta una funzione che, data la testa di una lista concatenata, verifichi se la lista è palindroma.
     La funzione proposta deve avere complessità spaziale $O(1)$ e complessità temporale $O(n)$.

Esercizi

1. Progetta una funzione che, data la testa di una lista concatenata di interi, stampi tutti i valori che compaiono almeno due volte nella lista. Valuta la complessità temporale e spaziale della funzione proposta.
2. Progetta una funzione che, data la testa di una lista concatenata, determini se questa è **ciclica**. Una lista a puntatori ciclica se un nodo della lista punta a un nodo precedente nella sequenza o alla testa della lista stessa, creando un ciclo.
   La funzione proposta deve avere complessità temporale $O(n)$ e complessità spaziale $O(1)$
3. Progetta una funzione che, data la testa di una lista concatenata, restituisca la stessa lista ordinata (senza creare nuovi nodi). Valuta la complessità dell'algoritmo proposto.
4. Dato un intero x, la testa p e la coda q di una lista concatenata progetta le seguenti funzioni:
   - *CancellaD*(p, q , x) che elimina dalla lista la prima eventuale occorrenza di x e restituisce la testa e la coda della lista eventualmente modificate.
   - *InserisciD*(p, q, x) che inserisce nella lista l'elemento x, se non già presente e e restituisce la testa e la coda della lista eventualmente modificate. Si assume che la lista sia ordinata in modo crescente.

   le funzioni proposte devono avere complessità temporale $O(n)$.