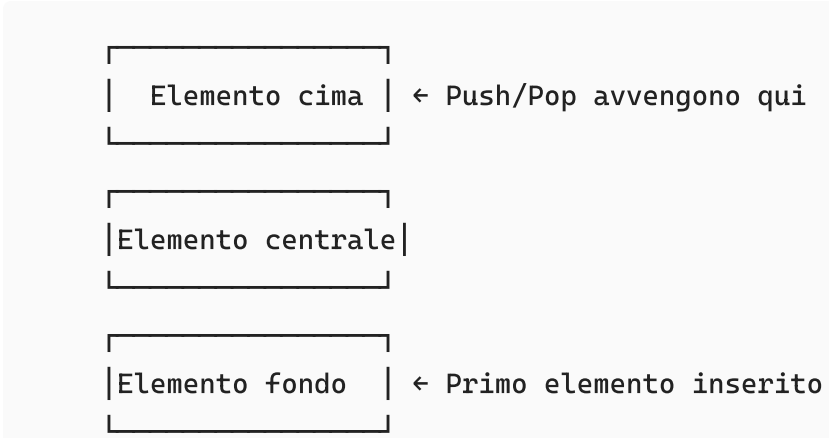
1. Pila (Stack - LIFO)



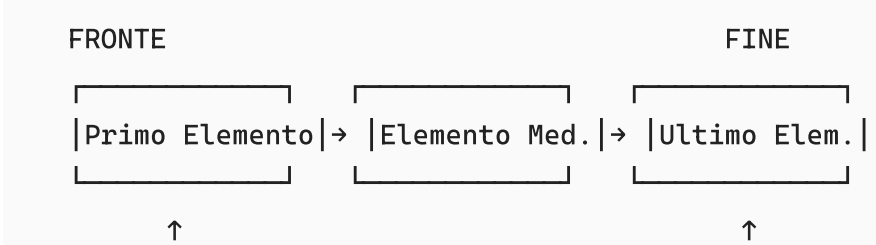
Metodi Principali e Complessità

Metodo	Descrizione	Complessità
push(elem)	Aggiunge elemento in cima	O(1)
pop()	Rimuove elemento dalla cima	O(1)
top()/peek()	Visualizza elemento in cima	O(1)
isEmpty()	Controlla se è vuota	O(1)
size()	Restituisce numero elementi	O(1)
bottom()	Visualizza elemento in fondo	O(1)

Casi d'uso

- Gestione delle chiamate di funzione (Call Stack)
- Valutazione delle espressioni
- Algoritmi di backtracking
- Funzionalità di annullamento nelle applicazioni

2. Coda (Queue - FIFO)



```
    Rimuovi qui                    Aggiungi qui
  (dequeue/remove)               (enqueue/add)
```

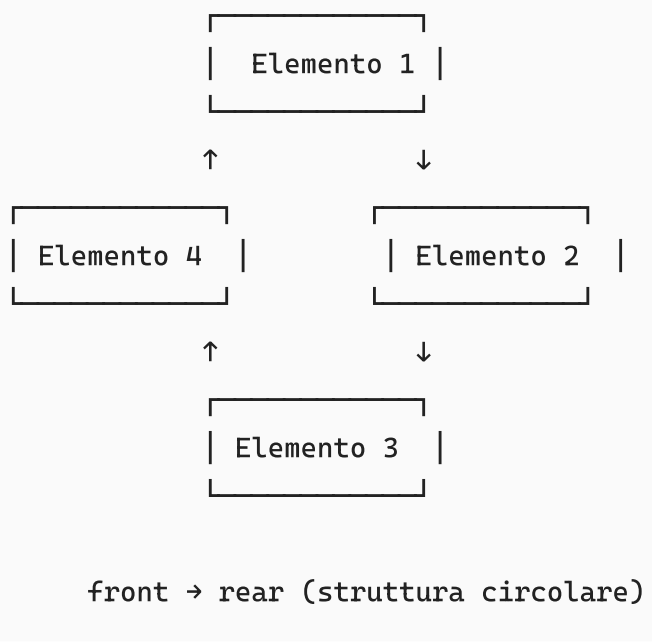
Metodi Principali e Complessità

Metodo	Descrizione	Complessità
enqueue(elem)	Aggiunge elemento alla fine	O(1)
dequeue()	Rimuove dal fronte	O(1)
getFront()	Visualizza elemento frontale	O(1)
isEmpty()	Controlla se è vuota	O(1)
size()	Restituisce numero elementi	O(1)

Casi d'uso

- Pianificazione dei task
- Ricerca in ampiezza (BFS)
- Gestione code di stampa
- Allocazione risorse

3. Coda Circolare



Metodi Principali e Complessità

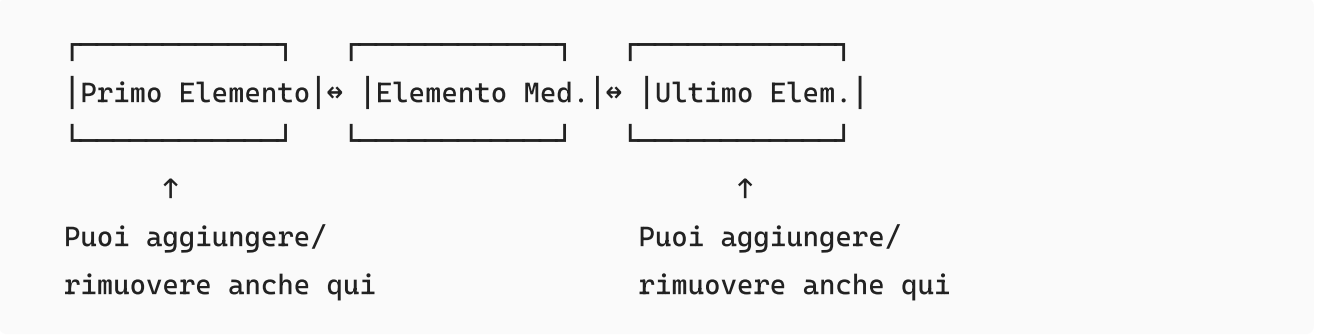
Metodo	Descrizione	Complessità
enqueue(elem)	Aggiunge elemento alla fine	O(1)

Metodo	Descrizione	Complessità
dequeue()	Rimuove dal fronte	O(1)
getFront()	Visualizza elemento frontale	O(1)
isEmpty()	Controlla se è vuota	O(1)
isFull()	Controlla se è piena	O(1)
size()	Restituisce numero elementi	O(1)

Caratteristiche principali

- Dimensione fissa, uso efficiente della memoria
- Utilizza aritmetica del modulo per avvolgersi
- Puntatori front e rear per tracciare la posizione
- Migliore di una coda semplice quando si lavora con spazio fisso

4. Coda a Doppia Estremità (Deque)



Metodi Principali e Complessità

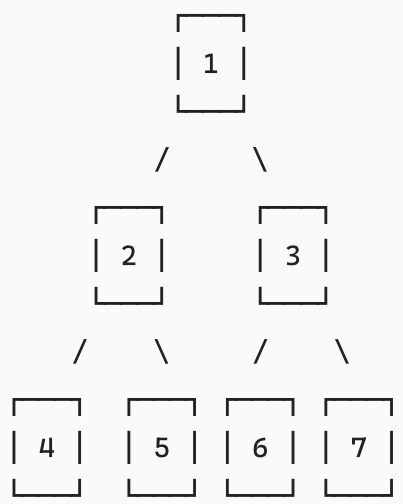
Metodo	Descrizione	Complessità
addFirst(elem)	Aggiunge all'inizio	O(1)
addLast(elem)	Aggiunge alla fine	O(1)
removeFirst()	Rimuove dall'inizio	O(1)
removeLast()	Rimuove dalla fine	O(1)
getFirst()	Visualizza primo elemento	O(1)
getLast()	Visualizza ultimo elemento	O(1)

Casi d'uso

- Controllo dei palindromi
- Implementazione sia di stack che di code
- Problemi con finestra scorrevole

- Algoritmi di work stealing

5. Coda a Priorità (Priority Queue)



Metodi Principali e Complessità

Metodo	Descrizione	Complessità
add(elem)/offer()	Aggiunge con priorità	O(log n)
remove()/poll()	Rimuove priorità più alta	O(log n)
peek()	Visualizza priorità più alta	O(1)
contains(obj)	Verifica presenza	O(n)
size()/isEmpty()	Operazioni base	O(1)

Caratteristiche principali

- Elementi ordinati per priorità, non per inserimento
- Tipicamente implementato con struttura dati heap
- La PriorityQueue di Java è un min heap per default
- Può utilizzare un Comparatore personalizzato per l'ordinamento

6. Lista Collegata (LinkedList)

Lista Semplicemente Collegata

Lista Doppiamente Collegata (Implementazione Java)

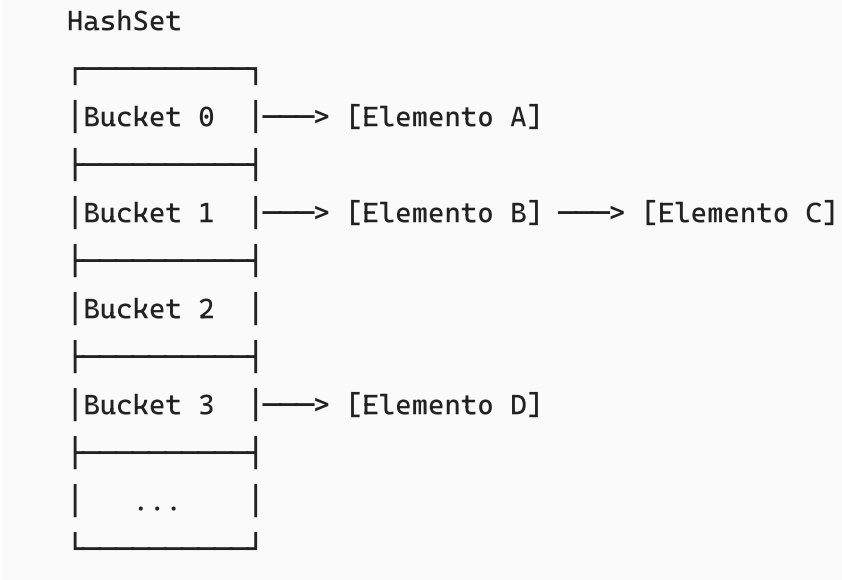
Metodi Principali e Complessità

Metodo	Descrizione	Complessità
add(elem)	Aggiunge alla fine	O(1)
add(index,elem)	Aggiunge in posizione	O(n)
get(index)	Accesso per indice	O(n)
remove(index)	Rimuove per indice	O(n)
addFirst()	Aggiunge all'inizio	O(1)
addLast()	Aggiunge alla fine	O(1)
iterator()	Ottiene un iteratore	O(1)

Casi d'uso

- Implementazione di pile e code
- Quando sono necessari frequenti inserimenti/cancellazioni
- Aritmetica polinomiale
- Playlist musicali

7. HashSet



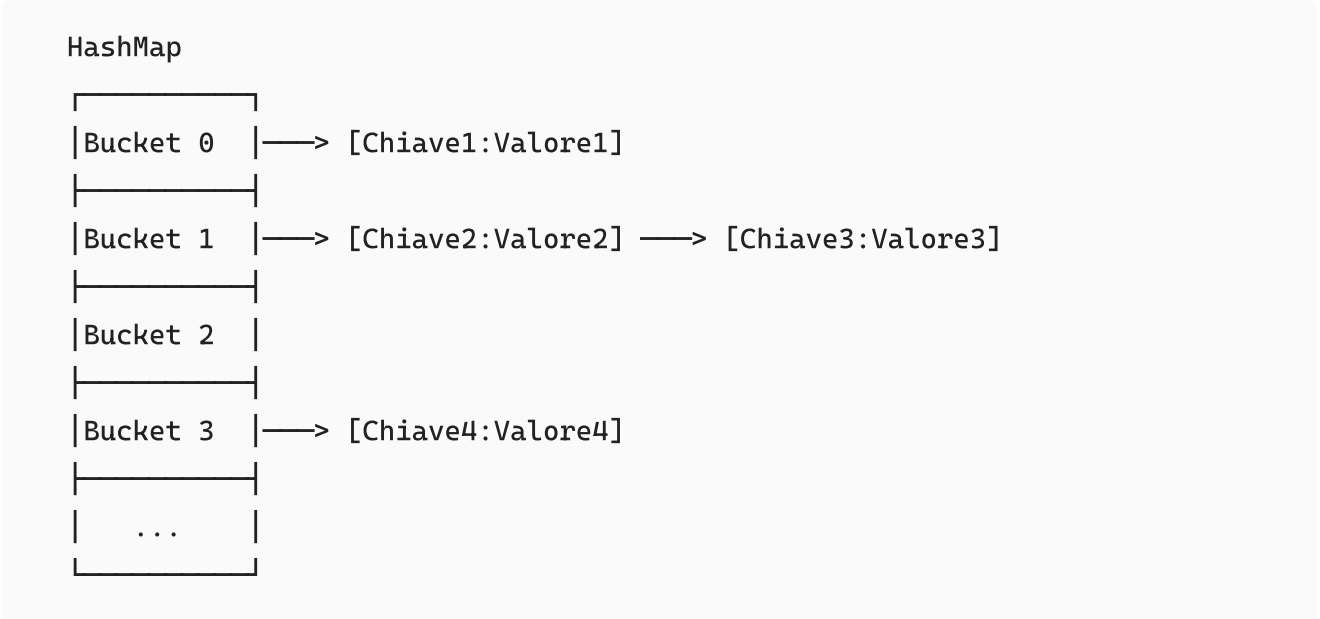
Metodi Principali e Complessità

Metodo	Descrizione	Complessità
add(elem)	Aggiunge elemento	O(1)
remove(elem)	Rimuove elemento	O(1)
contains(elem)	Verifica esistenza	O(1)
size()/isEmpty()	Operazioni base	O(1)
addAll()	Unione con altro set	O(n)
retainAll()	Intersezione con set	O(n)
removeAll()	Differenza con set	O(n)

Caratteristiche principali

- Nessun duplicato consentito
- Nessun ordine garantito
- Ricerca/inserimento/eliminazione molto veloci
- Utilizza il metodo hashCode() dell'elemento

8. HashMap



Metodi Principali e Complessità

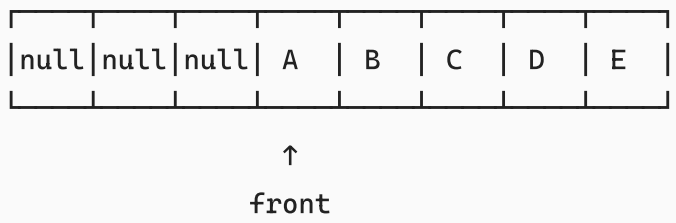
Metodo	Descrizione	Complessità
put(key,value)	Memorizza mappatura	O(1)
get(key)	Recupera valore	O(1)

Metodo	Descrizione	Complessità
remove(key)	Rimuove mappatura	O(1)
containsKey(key)	Verifica esistenza chiave	O(1)
containsValue(v)	Verifica esistenza valore	O(n)
entrySet()	Ottiene set di coppie	O(1)

Casi d'uso

- Caching
- Conteggio frequenze
- Ricerca veloce dei dati
- Implementazione di dizionari

9. CodaArrayListEstremoFissoLiberato



Metodi Principali e Complessità

Metodo	Descrizione	Complessità
enqueue(elem)	Aggiunge elemento alla fine	O(1)
dequeue()	Rimuove dal fronte	O(1)
getFront()	Visualizza elemento frontale	O(1)
isEmpty()	Controlla se è vuota	O(1)
size()	Restituisce numero elementi	O(1)
compact()	Compatta array (privato)	O(n)

Caratteristiche principali

- Utilizza ArrayList con puntatore frontale
- Operazioni di dequeue in tempo costante O(1)
- Compattazione periodica per ottimizzare lo spazio
- Efficiente gestione della garbage collection

Tabella Riassuntiva delle Complessità

Struttura Dati	Accesso	Ricerca	Inserimento	Cancellazione	Sp
ArrayList	O(1)	O(n)	O(1)/O(n)*	O(n)	O(
Pila (Stack)	O(1)	O(n)	O(1)	O(1)	O(
Coda (Queue)	O(1)	O(n)	O(1)	O(1)	O(
Coda Circolare	O(1)	O(n)	O(1)	O(1)	O(
Deque	O(1)	O(n)	O(1)	O(1)	O(
Coda a Priorità	O(1)**	O(n)	O(log n)	O(log n)	O(
Lista Collegata	O(n)	O(n)	O(1)***	O(1)***	O(
HashSet	N/A	O(1)	O(1)	O(1)	O(
TreeSet	N/A	O(log n)	O(log n)	O(log n)	O(
HashMap	N/A	O(1)	O(1)	O(1)	O(
TreeMap	N/A	O(log n)	O(log n)	O(log n)	O(
CodaArrayListEstremoFissoLib.	O(1)	O(n)	O(1)	O(1)	O(

- O(1) ammortizzato per aggiunta alla fine, O(n) per aggiunta in posizione arbitraria
  Solo per l'elemento con priorità più alta
  ***** Quando la posizione è già nota

Come Scegliere la Struttura Dati Corretta

Quando si seleziona una struttura dati, considerare:

1. **Operazioni necessarie**: Quali operazioni verranno eseguite più frequentemente?
2. **Complessità temporale**: Quali caratteristiche di prestazioni sono richieste?
3. **Vincoli di memoria**: Quanta memoria è disponibile?
4. **Requisiti di ordinamento**: Gli elementi devono essere in un ordine specifico?
5. **Accesso concorrente**: La struttura sarà acceduta da più thread?

Guida rapida:

- **Accesso casuale veloce per indice?** → ArrayList
- **Operazioni LIFO?** → Stack o ArrayDeque
- **Operazioni FIFO?** → Queue, LinkedList, o ArrayDeque
- **Inserimento/rimozione a tempo costante alle estremità?** → ArrayDeque
- **Elementi ordinati in base alla priorità?** → PriorityQueue
- **Inserimento/rimozione veloce nel mezzo?** → LinkedList
- **Eliminare i duplicati?** → HashSet o TreeSet

- **Mappature chiave-valore?** → HashMap o TreeMap
- **Mappature chiave-valore in ordine?** → TreeMap
- **Preservare l'ordine di inserimento?** → LinkedHashMap o LinkedHashSet