Strutture dati Lineari

Spicoli Piersilvio

Dipartimento di Informatica – Bari

1. **Lista**
2. **Pila**
3. **coda**

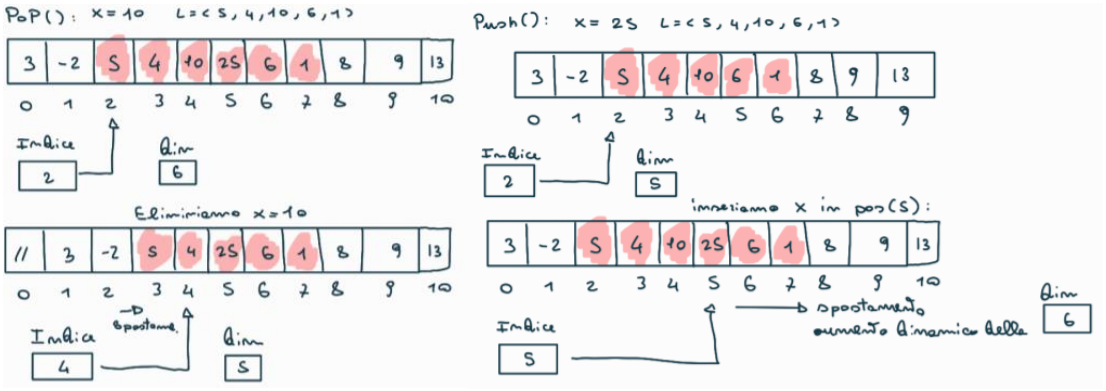
Lista

Definizione di lista

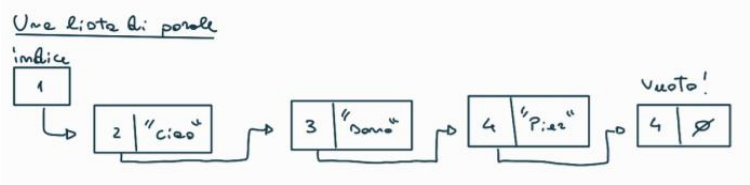
Una lista è una sequenza di finita, anche vuota, di elementi omogenei fra loro. La lista, a differenza di un insieme di elementi, può consentire la memorizzazione degli elementi duplicati conoscendone, inoltre, la posizione di ogni elemento. Essa verrà denotata come
L = < a1, a2, …, ai> ∀ i= 1….n con pos(i) la posizione i-esima dell'elemento e a(i) il suo elemento i-esimo.
La lista è definita come una struttura dati dinamica poiché, a differenza di un classico array, la sua dimensione può variare in base all'operazione di inserimento (push) o di cancellazione di un elemento i-esimo (pop). Inoltre, utilizza una modalità d'accesso ad ogni singolo elemento per mezzo della scansione dell'intera lista (non è possibile accedervi in maniera diretta!). Una lista, al momento della creazione, viene definita come lista vuota, ossia con nessun elemento all'interno di essi (|L| = 0). Denotiamola come L = < >. Inoltre, la dimensione di una lista tiene anche conto del numero di elementi i-esimi duplicati all'interno.
Diamo una proprietà che ci servirà con l'implementazione di tale stuttura:
Sia una lista L = < a1, a2, …, ai> ∀ i= 1….n. Si definisce sottolista L' ⊆ L la lista L' = < ak,…,aj > tale che
$1 \leq k \leq j \leq n$. Tale sottolista parte parte dalla posizione k-esima, definita come primo elemento di essi, fino ad arrivare alla posizione j-esima. Si osservi che L = < > è sottolista di qualsiasi lista.

Implementazioni delle liste

Una lista è possibile rappresentarla per modo sequenziale oppure collegata.
Una lista che utilizza il metodo sequenziale può essere implementata per mezzo di un array monodimensionale accompagnato da due variabili cui una rappresenterà l'indice della posizione del primo elemento della lista su cui partirò scandendo linearmente ogni singola posizione per accedere ad una determinata posizione e una che rappresenterà il numero di elementi attualmente memorizzati.
Descriviamone il suo funzionamento:
Consideriamo quindi una lista composta da interni piena cui sappiamo per mezzo dell'invocazione dei metodi la prima posizione e il numero di elementi contenuti. Per effettuare l'operazione di inserimento, la classe che definisce l'oggetto lista invocherà il metodo push() che consente di inserire l'elemento in una determinata posizione. Per far ciò, l'indice che scandisce le posizioni partirà dal primo elemento fino alla posizione in cui si vuole inserire il dato, shiftando verso destra gli elementi adiacenti. Tale operazione, in termini di complessità, può risultare O(n) lineare poiché la dimensione della lista aumenta in modo proporzionale agli elementi spostati alla fase di inserimento. Tale discorso è analogo per la cancellazione di un singolo elemento in posizione i-esima, a differenza però, che l'operazione di spostamento viene fatto da sinistra. Come già detto, tale implementazione richiede che la lunghezza massima di una lista sia puramente logica in quanto dinamica rispetto l'array, di conseguenza è necessario definire a priori una dimensione che, dinamicamente, sia piccola o grande. Difatti, per mezzo della classe <vector>, si incorre alla tecnica di raddopiamento – dimezzamento. Tale operazione, in termini di complessità, può risultare costosa in quanto consente di creare un array di appoggio su cui "incollare" gli elementi copiati dalla lista precedente, quindi la complessità aumenta in termini di spazio. Diamo una vista logica di tale implementazione:

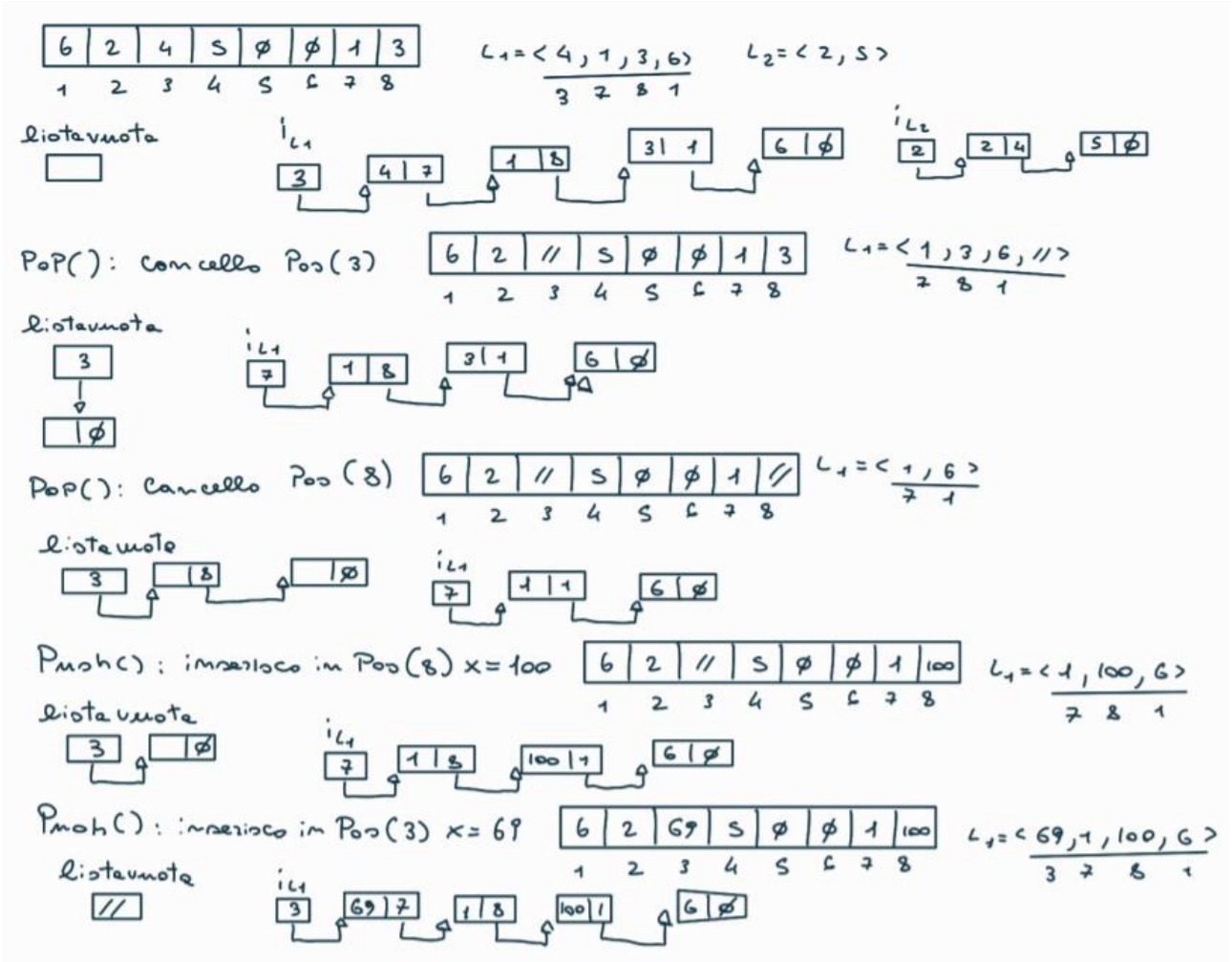


Il metodo con liste collegate consente di memorizzare i suoi elementi associando ad ognuno di essi un proprio riferimento (indice della loro posizione) che permetta di individuare dove è memorizzato l'elemento successivo. Tale rappresentazione è la seguente:



Tale rappresentazione fornisce a sua volta due tipi di implementazioni: Con cursori, puntatori e simmetriche.

L'implementazione coi cursori prevede un array monodimensionale che, a differenza dell'implementazione sequenziale, riesce ad aggirare l'ostacolo dell'aggiornamento della dimensione logica della lista. Un cursore è un tipo di variabile che consente di memorizzare la prima posizione dell'elemento di una o piu lista. In parallelo abbiamo un altro array "logico" detto listalibera che contiene ogni riferimento successivo libero all'interno dell'array fisico.
Consideriamo due liste di interi memorizzati all'interno di un array. La lista disporrà di due variabili cursore che punteranno entrambi al primo riferimento dei rispettivi elementi e ogni istanza della lista è composta dal valore dell'elemento e il riferimento successivo al prossimo elemento. Seguendo i cursori si trovano gli elementi successivi, ossia accedere alla pos(i) di un qualunque elemento come pos(i -1) esimo se $2 \leq i \leq n + 1$, oppure pos(i) = 0 se l'elemento è il primo. L'ultimo elemento della lista è definito con pos(n + 1). Per poter aggiornare una lista cosi, ricorriamo all'utilizzo della lista libera, che memorizza tutti i riferimenti dell'array liberi per inserimenti. Per poter cancellare un elemento dalla lista, sarà sufficiente metterlo in testa rispetto alla prima posizione della listalibera aggiornandone il riferimento e far puntare il precedente elemento cancellato al suo suo successivo dell'elemento eliminato. Per poter inserire un elemento in lista, verifico in primis se in listalibera viene fornito un indice di riferimento libero, poi scrivo l'elemento in questo riferimento aggiornando poi i riferimenti facendo puntare quello che prima puntava al successivo, al nuovo elemento e il nuovo elemento, al successivo del precedente (in poche parole in mezzo!). In termini di complessità, le operazioni di push() e pop() hanno O(1) poiché l'unica variazione dipende esclusivamente dal riferimento su cui punta la lista. Tale implementazione non risolve ancora la staticità dell'array, in quanto si necessita di memorizzare anche i riferimenti degli elementi.



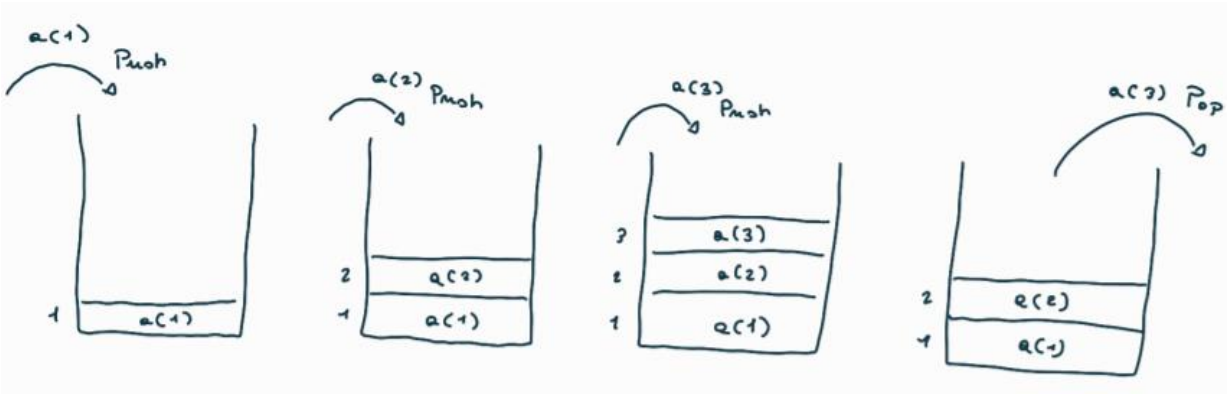
L'implementazione con i puntatori risolve l'allocazione statica che limitava appunto l'utilizzo della lista per mezzo di allocazioni dinamiche. Con le variabili puntatori accediamo alla locazione in cui l'elemento i-esimo è memorizzato, chiedere le nuove locazioni di memoria e rilasciare quelle utilizzate precedentemente dalla listavuota. In modo analogo ai cursori, pos(i) è il valore del puntatore alla cella che contiene l'i-esimo elemento.
Per la realizzazione, la lista inizializzata conterrà un indirizzo vuoto di partenza a cui punta il primo elemento, che è composto dal valore e l'indirizzo di memoria successivo. Per inserire un elemento, viene invocato il metodo new per creare dinamicamente una nuova locazione di memoria a cui farò puntare l'elemento precedente al nuovo elemento e questo al successivo del precedente. Tale implementazione comporta tutti i vantaggi dell'implementazione precedente. L'inserimento in questo caso può avvenire sia in coda che all'inizio: si crea l'elemento, lo si fa puntare a ciò che prima puntava il puntatore di inizio lista e aggiorno il puntatore di inizio lista affinchè punti a questo. A differenza, per inserire l'elemento in coda si inserisce nel nuovo elemento il terminatore di fine lista e si fa puntare quello che puntava il terminatore al nuovo elemento. Per la cancellazione si invoca il metodo delete salvando la posizione dell'elemento puntato e scrivendo questo indirizzo nel precedente di quello che doveva essere eliminato. L'eliminazione in testa avviene salvando l'indirizzo successivo dell'elemento cancellato in testa e aggiornando quindi l'elemento da puntare in testa. Analogo in coda aggiornando l'indirizzo del penultimo elemento cancellato affinchè punti alla fine della lista. In termini di complessità, il metodo push() e pop() hanno O(1) in quanto, nonostante aumenti la complessità in termini di spazio, la complessità in termini di tempo rimane invariata perché varia solo il riferimento a cui punta il punatore (simile all'implementazione con cursori).
In entrambe le implementazioni, per accedere direttamente all'i-esimo elemento, avremo complessità O(n) poiché ci costerà in ogni caso scandire ogni riferimenti degli elementi.

L'implementazione di liste simmetriche(circolari) rappresenta una lista infinita cui l'ultimo riferimento punterà al primo elemento, ossia il suo successivo, e il precedente del primo il suo ultimo.

Pila

Definizione di pila

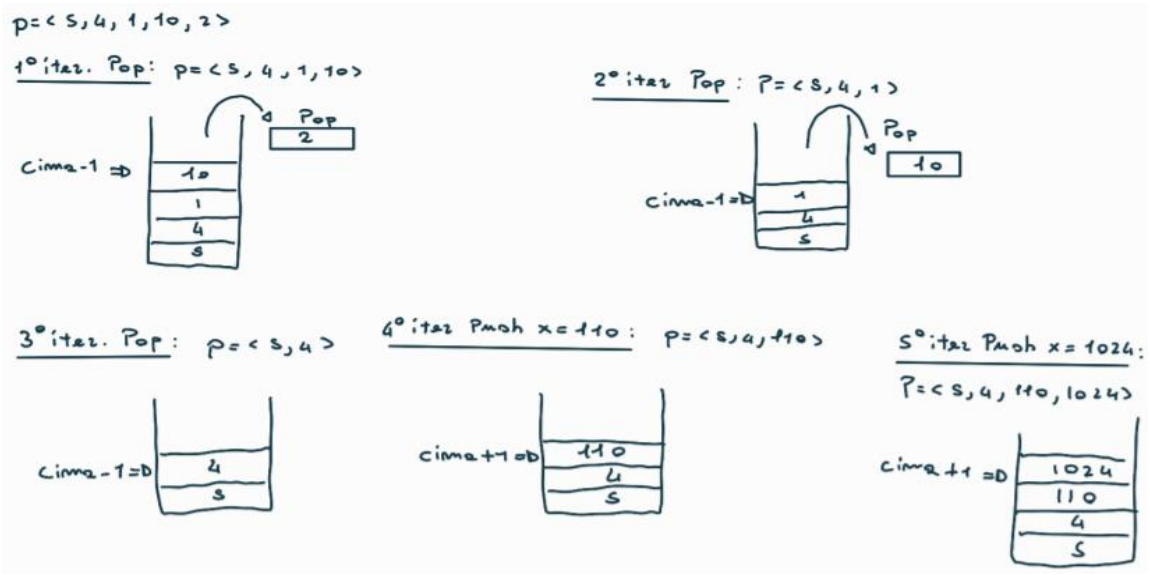
Le pile(stack) sono alla base della gestione del sistema operativo in quanto viene applicata all'implementazione delle chiamate a funzioni per la gestione generale dei processi. Una Pila è una sequenza dinamica di un certo tipo in cui è possibile aggiungere o eliminare elementi solo da un estremo della sequenza. Infatti, tale struttura dati può essere vista come un'estensione(o caso speciale) della lista in cui l'ultimo elemento inserito è il primo ad essere rimosso (politica LIFO) è non è possibile accedere ad alcun elemento che non sia quello in testa. La definizione formale e le sue proprietà sono analoghe a quelle della lista.



Implementazioni Pila

È possibile implementare la pila per mezzo di rappresentazione sequenziale(array) oppure collegata(con puntatori o cursori).

Rappresentazione sequenziale: Come abbiamo già visto, la pila è un particolare tipo di lista che segue una politica di tipo LIFO, ampliamente diversa per come seguono le operazioni sulle liste. Per poter inserire/cancellare un elemento i-esimo all'interno della lista, era di norma "spostare" gli elementi vicini a destra/sinistra dell'array. Per la pila invece, tali operazioni vengono effettuate in cima la struttura, ma con lo stesso problema legato allo spostamento degli elementi in inserimento/cancellazione. Per ovviare a tale problema, il concetto di pila permette di capovolgere in modo inverso (dall'ultimo fino al primo, con l'ultimo in cima) la sequenza di elementi. Tale operazione quindi permette un vantaggio notevole in quanto la complessità in termini di spazio e di tempo diventi O(1), rispetto allo spostamento all'interno delle liste che provoca O(n). Quindi tale implementazione si basa sul memorizzare la sequenza degli elementi in modo inverso, si tiene traccia attraverso una variabile cursore(detta cima) l'indice in cui si trova il primo elemento a1 e quando viene invocato il metodo di Push (inserimento) basterà effettuare cima + 1, in modo da aggiungere l'elemento e aggiornare la variabile cima all'elemento ultimo aggiunto (LIFO). Quando viene invocato il metodo Pop(cancellazione) è sufficiente puntare la variabile cima all'ultimo elemento an e porre cima - 1. Si tenga in considerazione che tale implementazione può essere fatta per mezzo della classe vector e, come per le liste, presenta un problema di staticità di dimensione in quanto fatta con array. Per ovviare tale problema, si definisca una dimensione accettabile e con la tecnica di raddoppiamento/dimezzamento si implementi una dimensione variabile in base alle richieste, definendo cosi formalmente come struttura dinamica.



Rappresentazione collegata: Questa realizzazione si basa sempre sull'aggiunta/rimozione dell'elemento in cima la pila, senza pero definire la dimensione dell'array, ne di dover raddoppiare/dimezzare la dimensione quando aggiorniamo la pila. Essa è analoga all'implementazione collegata delle liste, dove con una variabile puntatore puntiamo all'indirizzo successivo del prossimo elemento. Conviene quindi rappresentare ogni elemento per mezzo di una struttura che ne definisce il valore dell'elemento e il puntatore all'indirizzo dell'elemento prossimo. Per poter realizzare questa implementazione, ci serviamo di una variabile puntatore di partenza che punta alla cima della pila. Per effettuare le operazioni di inserimento in cima la pila, bisognerà allocare nuova memoria con l'operatore new, far puntare il nuovo puntatore a cui puntava la cima e aggiornare la cima, affinché punti all'indirizzo del nuovo elemento. Per effettuare l'operazione di cancellazione si invoca il metodo delete per deallocare la sua posizione dalla cima e far puntare la cima al prossimo elemento. La complessità computazionale rimane comunque costante, ma con un aumento di spreco di spazio in quanto si alloca e dealloca dinamicamente le referenze degli elementi in pila O(1). L'idea illustrativa è simile alla rappresentazione collegata della lista, solo che fatta in modo inverso (dall'ultimo al primo).

Applicazioni della Pila

Tale struttura dati risulta molto flessibile in termini di applicazioni in quanto permette la granularità di essere dinamica per mezzo di una politica LIFO. Un esempio di applicazione risiede nei problemi di tipo ricorsivo: l'esecuzione di una procedura ricorsiva infatti, prevede il salvataggio dei dati su cui lavora al momento della chiamata ricorsiva(richiama se stesso) e questi dati salvati, rimangono in pila fintantoché non termini la computazione, con l'idea del fatto che una chiamata più recente con i suoi dati, termini prima la sua esecuzione, e quindi in LIFO. Tale ricorsione la si incontra negli stack dei sistemi operativi, problemi ricorsivi che diventino iterativi per mezzo dello stack, come la succesione di fibonacci o le torri di hanoi. Si consideri in oltre che per trasformare un algoritmo ricorsivo in iterativo(o viceversa) basti creare una pila, sostituire ogni chiamata ricorsiva con istruzioni di chiamata a procedura(i maledetti salti) che eseguino il loro blocco di istruzione finché non terminino, per poi uscire dalla procedura.

Coda

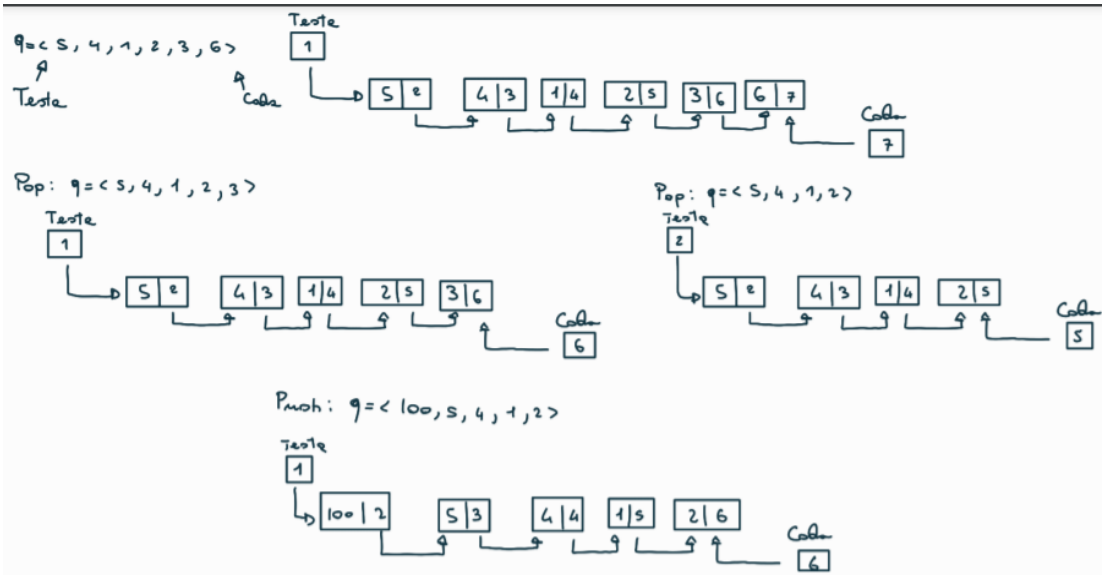
Definizione di coda

La coda(queue) è una struttura dati di tipo dinamica basata sulla politica FIFO, dove l'inserimento di un elemento i-esimo avviene ad un estremo e la cancellazione di un elemento avviene dall'estremo opposto da una sequenza di elementi. È un particolare dato di tipo astratto che si basa su sequenze nelle quali gli elementi sono definiti secondo un ordine di arrivo o di priorità. Rispetto alla Lista, la coda è limitata in termini di inserimento e cancellazione, in quanto essi avvengo solo ed esclusivamente agli estremi, e ciò porta ad avere una maggior efficienza in termini implementativi sugli estremi, ossia la testa e la coda.

Implementazioni Coda

L'implementazione che abbiamo visto fino adesso, come quelle della lista con array e vector, non sono utili per realizzare una coda in maniera efficiente poiché sono basate su un tipo di accesso diretto all'elemento i-esimo, e non esclusivamente agli estremi per la politica che adotta la coda (basti pensare che per fare una cosa del genere, scandiremmo tutti gli elementi per arrivare agli estremi e avere una complessità lineare in termini di tempo). Tuttavia esistono due tipi di rappresentazioni, una collegata (con puntatori) e una circolare:

Implementazione collegata: Per gestire questa rappresentazione utilizziamo i puntatori per accedere agli indirizzi in cui è memorizzato l'elemento. Ogni elemento, dunque, viene rappresentato per mezzo di una struttura composta da una variabile che contiene il valore dell'elemento e un puntatore all'indirizzo successivo. Inoltre abbiamo bisogno di un indirizzo di partenza(ossia un singolo puntatore) che permette di puntare all'indirizzo del primo elemento in testa e l'elemento in testa punterà col suo puntatore all'idirizzo dell'elemento successivo e cosi via. In questo modo formiamo una rappresentazione collegata come per le liste, ma con la differenza però di tenere traccia non solo la posizione del primo elemento, ma anche dell'ultimo elemento in coda per poter fare pop/push (FIFO).
All'invocazione di Push di un elemento x in testa, si definisce un nuovo puntatore in cui inserire il nuovo valore, far puntare il suo puntatore all'indirizzo dell'elemento in testa e aggiornare il puntatore dell'elemento x inserito. In termini di tempo, si compiono 4 operazioni che non influiscono in base alla dimensione dell'input, quindi O(1). **All'invocazione di push di un elemento x in coda** avviene in modo analogo (anche in termini di complessità) con la differenza che l'elemento in coda non punterà a NULL, ma all'indirizzo dell'elemento x, e x punterà infine a NULL. **All'invocazione di Pop di un elemento x in testa** basterà deallocare l'elemento puntato in testa e far puntare la testa al suo elemento successivo, con complessità O(1).
All'invocazione di Pop su un elemento x in coda bisognerebbe far puntare a NULL l'elemento precedente, e non a quello eliminato, però sorge un problema, ossia che non sappiamo quale sia l'indirizzo dell'elemento precedente. Per fare ciò abbiamo due soluzioni per poter ottenere l'indirizzo dell'elemento precedente:
Effettuare una ricerca lineare finché non si raggiunge l'ultimo elemento(ossia il precedente descritto prima) e farlo puntare a NULL, avendo però un costo di complessità in termini di tempo pari a O(n)
Implementare una lista circolare, con politica FIFO, ma con un aumento di spreco di memoria in quanto dichiaro
In tal caso, dopo aver descritto tutte le possibili operazioni di push/pop, conviene effettuare l'operazione di pop in coda e push in testa, rispettando quindi la politica FIFO e avere complessità O(1).



Implementazione circolare: Basandoci sempre su un array primitivo, notiamo che il problema della staticità non viene risolto in quanto, a differenza di una pila(LIFO), l'estensione della dimensione può avvenire o a un estremo o all'altro, senza oltrepassare l'indice 0 del primo elemento. Avremo quindi un array primitivo di n-elementi dove ogni elmento i-esimo potrà puntare agli indirizzi degli elementi successivi e precedenti, con il primo elemento che punterà al suo precedente, ossia l'ultimo, e l'ultimo potrà puntare al suo successivo, ossia il primo (in modo circolare). Usiamo quindi due variabili, una "primo" che punta al primo elemento in testa, "lunghezza" che punta all'ultimo elemento in coda. Un array del genere è composto da una maxlung di elementi con indice che varia da 0 a maxlung – 1, in cui consideriamo l'elemento di indice 0 come successore di quello di indice di maxlung – 1. All'invocazione del metodo push di un elemento x, si verifica se la lunghezza della coda ha raggiunto maxlung-1: in caso in cui non si avrà raggiunto maxlung-1, incrementiamo la variabile lunghezza facendola puntare all'ultimo elemento in coda. All'invocazione del metodo pop dalla testa della coda, si decrementa la variabile lunghezza e si incrementa di una posizione il puntatore alla testa della coda. L'operazione di push viene effettuata senza problemi quando lunghezza non raggiunge maxlungh-1, ma se nel caso in cui venga raggiunta maxlungh-1, l'array rimane comunque statico di dimensione, in quanto c'è la possibilità che non si poppa nessun elemento dalla testa. Quindi è utile rendere dinamica la dimensione con la tecnica di raddoppiamento e dimezzamento su maxlungh. La sua complessità, in generale, rimane invariata a O(1).