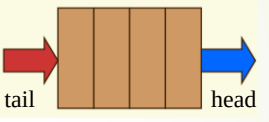
Programmazione e Strutture Dati (PR&SD)
I° ANNO – Informatica
Prof. V. Fuccella

ADT *QUEUE (CODA)*

ADT *Queue*

- Una ***queue***(***coda***) è una sequenza di elementi di un determinato tipo, in cui gli elementi si aggiungono da un lato (***tail***) e si tolgono dall'altro lato (***head***).



- La sequenza viene gestita con la modalità ***FIFO (First-in-first-out):*** il primo elemento inserito nella sequenza sarà il primo ad essere eliminato.
- La coda è una struttura dati *lineare* a *dimensione variabile*
  - Si può accedere direttamente solo alla testa (***head***) della lista.
  - Non è possibile accedere ad un elemento diverso da ***head***, se non dopo aver eliminato tutti gli elementi che lo precedono (cioè quelli inseriti prima).

ADT: Queue

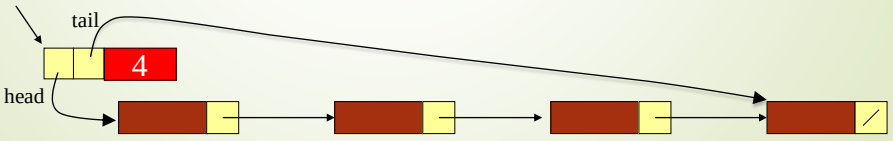
Sintattica	Semantica
Nome del tipo: Queue Tipi usati: Item, boolean	Dominio: insieme di sequenze $S=a_1,\ldots,a_n$ di tipo Item L'elemento **nil** rappresenta la coda vuota
newQueue() → Queue	newQueue() → q • Post: q = nil
isEmptyQueue(Queue) → boolean	isEmptyQueue(s) → b • Post: se q=nil allora b = true altrimenti b = false
enqueue(Queue, Item) → Queue	enqueue(q, e) → q' • Post: q = <a1, …, an> AND q' = <a1, …, an, e>
dequeue(Queue) → Queue	dequeue(q) → q' • Pre: q = <a1, a2, …, an> n>0 • Post: q' = <a2, …, an>

ADT *Queue*
Implementazione

- Tra le possibili implementazioni, le più usate sono realizzate tramite:
  - **Lista concatenata**
  - **Array**

ADT Queue
Implementazione con Liste

- È possibile utilizzare gli operatori di:
  - Rimozione dalla testa
  - Aggiunta in coda
- Per motivi di efficienza, conviene avere accesso sia al primo elemento sia all'ultimo. Occorre modificare il tipo lista come un puntatore ad una struct che contiene
  - Un intero numelem che indica il numero di elementi della coda
  - Un puntatore head ad uno struct nodo
  - Un puntatore tail ad uno struct nodo



ADT Queue
Modifica Implementazione ADT Lista

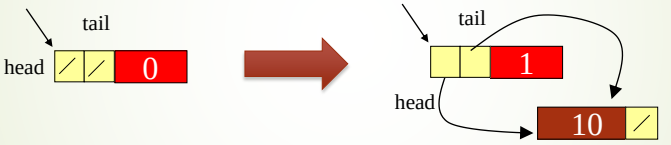
- Dobbiamo innanzitutto aggiungere il puntatore tail
- Poi bisogna modificare gli operatori principali:
  - *RemoveHead*
    - Deve eventualmente aggiornare entrambi i puntatori head e tail.
  - *addListTail*, grazie alla presenza del puntatore tail,
    - Non deve più scorrere gli elementi della lista fino all'ultimo
    - Deve eventualmente aggiornare entrambi i puntatori head e tail.

ADT Queue
Modifica removeHead (ADT List)

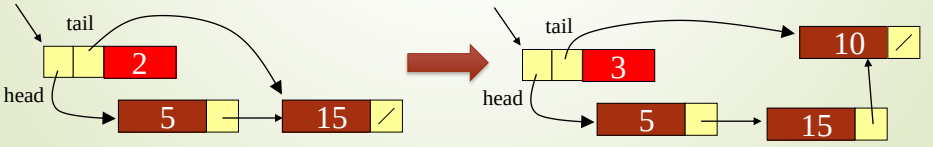
- Bisogna prima salvare il puntatore al nodo da eliminare (quello puntato da head)
- Head dovrà quindi puntare al successivo
- A questo punto si può deallocare la memoria del nodo da rimuovere
- Se la coda aveva un solo elemento, ora è vuota, per cui bisogna porre anche il puntatore tail a NULL

ADT Queue
Modifica addListTail (ADT List)

- Dobbiamo innanzitutto creare un nuovo nodo a cui dovrà puntare il puntatore tail
- Poi bisogna distinguere il caso in cui la coda di input è vuota e il caso in cui non è vuota
  - Coda vuota: il puntatore head dovrà puntare al nuovo nodo



  - Coda non vuota: il puntatore next dell'ultimo nodo dovrà puntare a nuovo

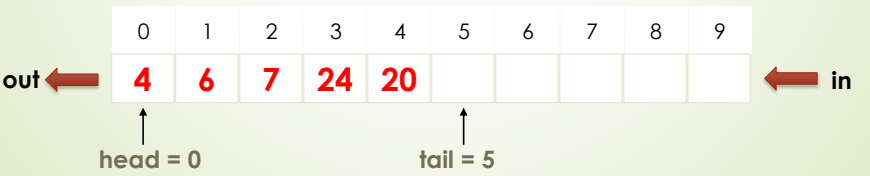


ADT QUEUE

Implementazione con Array

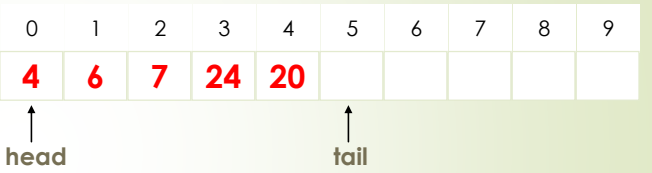
ADT Queue
Implementazione con array

- La coda è implementata come un puntatore ad una struct queue che contiene tre elementi:
  - Un array di MAXQUEUE elementi
  - Un intero che indica la posizione *head* della coda
  - Un intero che indica la posizione *tail* della coda
- Quando la coda si riempie, non è possibile eseguire l'operazione enqueue …

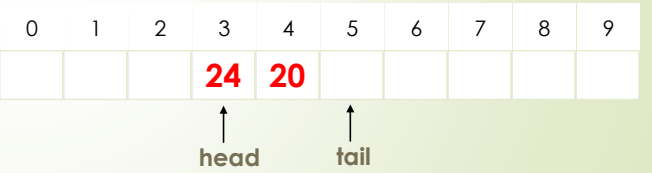


ADT Queue
Implementazione con array - Problemi

- Se l'array viene gestito normalmente, cioè mantenendo head <= tail, ci sono dei problemi…
- Esempio:



- Se rimuoviamo uno alla volta i primi tre elementi in coda otteniamo:



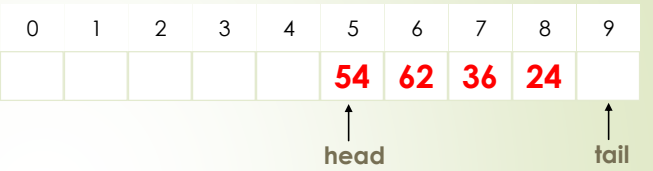
- Risultano disponibili solo le posizioni a destra di tail, ma sono libere anche quelle a sinistra di head

ADT Queue
Implementazione con array - Soluzioni

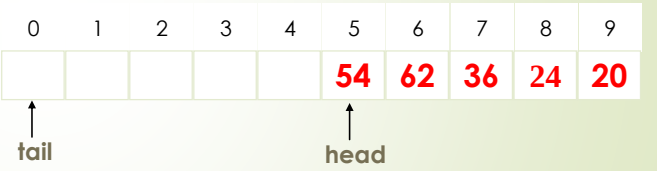
- **Prima soluzione**: ad ogni rimozione si compatta l'array nelle posizioni iniziali, con uno shift degli elementi
  - **TROPPO COSTOSO!**
- **Seconda soluzione**: si gestisce l'array in modo circolare
  - In ogni istante, gli elementi della coda si trovano nel segmento head, head+1,... tail-1
  - … ma non necessariamente head <= tail
  - Infatti, dopo aver inserito in posizione N-1 (ultima posizione dell'array), se c'è ancora spazio in coda, si inseriscono ulteriori elementi a partire dalla posizione 0 (prima posizione dell'array)
  - In questo modo si riesce a garantire che ad ogni istante la coda abbia capacità massima di N-1 elementi

ADT Queue
Implementazione con array circolare

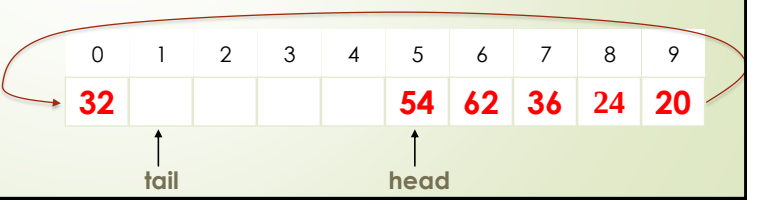
- Supponiamo di voler inserire 20 e 32 in questa coda



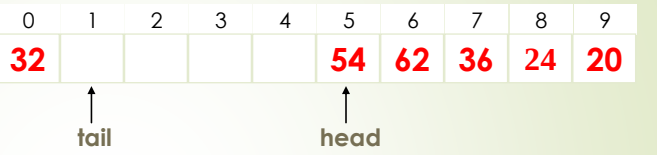
- Inseriamo 20 …



- Adesso inseriamo 32 …



ADT Queue
Implementazione con array circolare



- Adesso tail < head, perché la posizione 0 segue la posizione N-1
- In questo ordine circolare il successore di p è (p + 1) % N
  - Ogni volta che si inserisce un elemento tail avanza: tail = (tail + 1) % N
  - Ogni volta che si rimuove un elemento head avanza: head = (head + 1) % N
- La coda è piena (e non si può eseguire enqueue) se il successore di tail in questo ordine circolare è head
  - (tail + 1) % n == head La condizione comporta una locazione vuota necessaria a distinguere la condizione di buffer vuoto da quella di pieno
- Quando la coda è vuota, i valori di head e tail coincidono

Considerazioni

- Abbiamo visto due implementazioni diverse dell'ADT queue.
  - lista
    - **pro**: è una implementazione espandibile (unico limite è la capacità della memoria)
    - **contro**: la struttura è più complessa
  - Array circolare
    - **pro**: gli elementi sono memorizzati in modo contiguo e la struttura è più semplice
    - **contro**: dimensione fissata, bisogna conoscere a priori il numero massimo di elementi che la coda deve contenere, parte dello spazio è inutilizzato
  - Note su array circolare
    - Potrei usare la realloc per ridimensionare la coda (come fatto per lo stack) e poter quindi sempre inserire elementi ?
    - Sì, ma devo considerare che l'ordine degli elementi della coda nell'array non necessariamente va dalla posizione 0 alla posizione n-1 …
    - Farlo come esercizio …