Analisi e progettazione del software
Compito di metà corso
24 novembre 2016

Una *Coda* è una struttura dati simile alla *Pila*, con la differenze che l'elemento eliminato dall'operazione *Pop* non è l'ultimo inserito bensì il primo inserito tra quelli presenti.

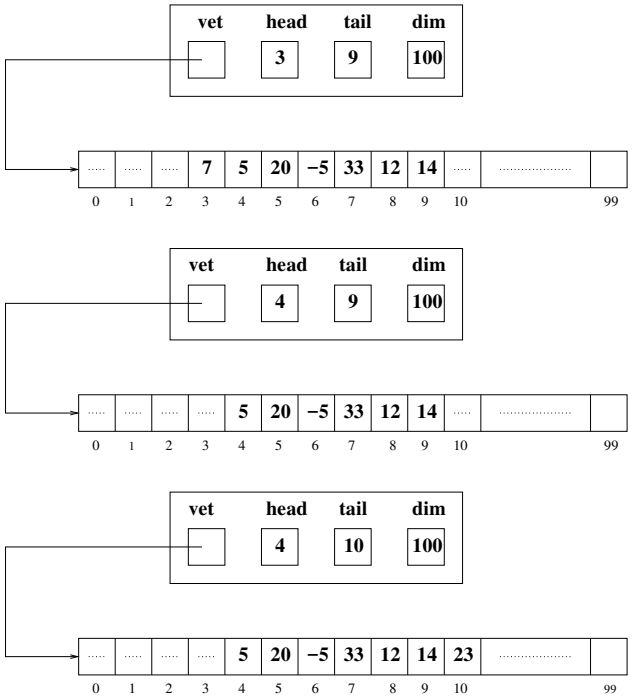
Ad esempio, data la coda di interi $\langle 7, 5, 20, -5, 33, 12, 14\rangle$, l'elemento affiorante è 7. L'esecuzione dell'operazione *Pop* porta la coda nello stato $\langle 5, 20, -5, 33, 12, 14\rangle$. La successiva operazione *Push(23)* porta nella stato $\langle 5, 20, -5, 33, 12, 14, 23\rangle$ (l'elemento affiorante è 5).

Si vuole progettare la classe `Coda` usando la stessa impostazione della classe `Pila` vista in classe: un vettore dinamico allocato dal costruttore che viene riallocato all'occorrenza dalla funzione `Push`.

Si richiede però di implementare le funzioni `Push` e `Pop` *senza fare spostamenti di elementi all'interno del vettore* ad ogni esecuzione.

Questo comporta che durante l'evoluzione dell'oggetto la coda si sposti all'interno del vettore. Sarà quindi necessario mantenere come dati membro della classe non l'indice dell'elemento affiorante `top`, ma due indici (chiamati `head` e `tail`) che memorizzano rispettivamente l'elemento affiorante (cioè di prossima uscita) e l'ultimo elemento inserito.

La figura seguente mostra l'oggetto che rappresenta la coda dell'esempio e la sua evoluzione a seguito delle operazioni *Pop* e *Push(23)*.



Si noti che il fatto che l'elemento in coda occupi l'ultima locazione del vettore non implica che il vettore sia pieno. In questo caso, all'invocazione dell'operazione *Push* invece di riallocare il vettore è richiesto di "ritirare" indietro la coda (ricopiando gli elementi nelle locazioni a partire da 0). Solo quando il vettore è effettivamente pieno, allora è corretto riallocarlo.

Esercizio 1 (punti 3) Si scriva la definizione della classe `Coda` con elementi interi.

Esercizio 2 (punti 12) Si scrivano le definizioni del costruttore senza argomenti e delle funzioni di classe `Push`, `Pop`, `Top` ed `EstVuota`, utilizzando l'eccezione `logic_error` per verificare le precondizioni.

Esercizio 3 (punti 8) Si scrivano le definizioni del costruttore di copia, dell'operatore di assegnazione e del distruttore.

Esercizio 4 (punti 5) Si scriva la definizione dell'operatore `==` che verifica se due code sono uguali, cioè contengono gli stessi elementi e nello stesso ordine.

Esercizio 5 (punti 4) Si scriva una funzione esterna (non *friend*) che prende come parametro una coda `c` ed un intero `n` e verifica se `c` ha almeno `n` elementi.