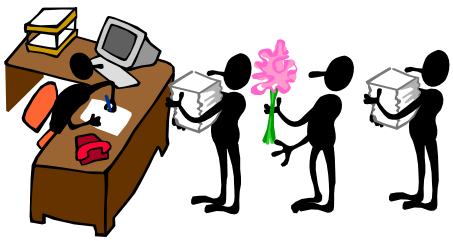
Queue

Corso: Strutture Dati

Docente: Annalisa De Bonis

La Coda

- La coda (queue) e` un TDA che immagazzina oggetti arbitrari in cui inserimenti e cancellazioni avvengono secondo lo schema FIFO (first-in-first-out)
  - Un nuovo oggetto viene inserito in coda (rear)
  - L'elemento cancellato e` sempre quello che si trova davanti a tutti gli altri (front)
    - Viene cancellato l'elemento che e` stato piu` a lungo nella coda



Operazioni sulla coda

enqueue(x)	aggiunge x alla fine (rear) della coda
dequeue()	restituisce l'elemento in testa (front) alla coda eliminandolo
front()	restituisce il valore in testa alla coda
isEmpty()	restituisce true se la coda non ha elementi
size()	restituisce il numero di elementi della coda

Condizioni di errore

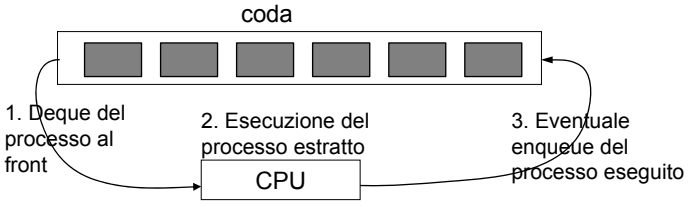
- Le operazioni dequeue e front provocano un errore se invocate su una coda vuota

Applicazioni della coda

- Lista di attesa per la fruizione di un servizio
- Ordine di accesso ad una risorsa condivisa (per esempio le stampanti)
- Struttura dati ausiliaria di algoritmi

Esempio di applicazione: sistema multitasking con scheduler Round Robin

- L'algoritmo di scheduling **Round Robin** esegue i processi nell'ordine d'arrivo
  - I processi vengono messi in una coda in base all'ordine di arrivo
  - A ciascun processo viene allocato lo stesso tempo t
  - Il controllo viene trasferito al processo che si trova al front della coda
    - Il processo viene cancellato dalla coda
    - Se allo scadere del tempo t l'esecuzione del processo non è terminata allora il processo viene reinserito nella coda



Interfaccia della Coda

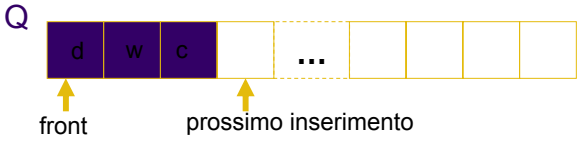
- Corrisponde al TDA Queue
- Richiede la definizione di EmptyQueueException

```
public interface Queue<E> {
  public int size();
  public boolean isEmpty();
  public E front() throws EmptyQueueException;
  public void enqueue (E element);
  public E dequeue() throws EmptyQueueException;
}
```

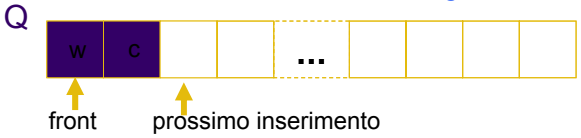

Code implementate con array

due soluzioni inefficienti

- Soluzione 1: Il front e` Q[0] e gli elementi vengono aggiunti da sinistra a destra
  - Svantaggio: ogni volta estraiamo un elemento bisogna shiftare a sinistra di una posizione i rimanenti elementi



Cancelliamo il front e shiftiamo gli elementi a sinistra



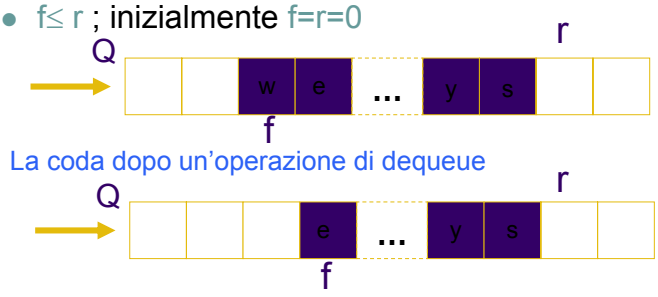
La coda dopo un'operazione di dequeue

Code implementate con array

due soluzioni inefficienti

Soluzione 2: Due variabili tengono traccia del front e del rear

- f : indice dell'elemento in testa (incrementata da deque)
- r : indice successivo a quello in coda (incrementata da enqueue)
- f≤ r ; inizialmente f=r=0



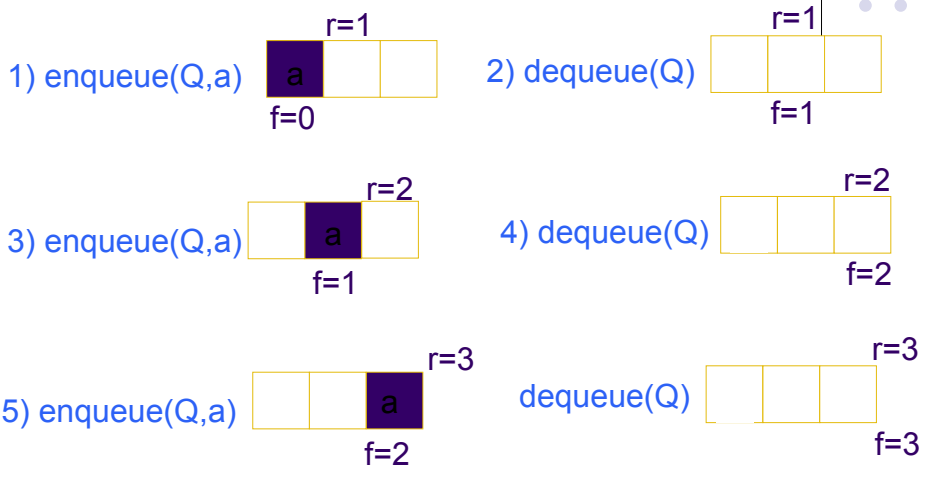
La coda dopo un'operazione di dequeue

Code implementate con array

due soluzioni inefficienti

- Svantaggio della soluzione 2:
  - Potrei non riuscire ad inserire elementi nella coda pur avendo spazio a disposizione nell'array.
  - Si consideri, per esempio, una coda inizialmente vuota implementata mediante un array di dimensione N. Se si inserisce e cancella N volte un elemento si ottiene f=r=N non rendendo possibili ulteriori inserimenti (un ulteriore inserimento provoca un errore di tipo array-out-of-bounds).

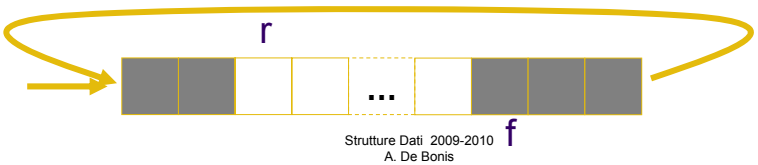
- Svantaggio della soluzione 2:



Code implementate con array

soluzione efficiente

- L'array viene usato "in maniera circolare"
- Due variabili tengono traccia del front e del rear
  - f : indice dell'elemento in testa
  - r : indice successivo a quello in coda
  - Inizialmente f=r=0

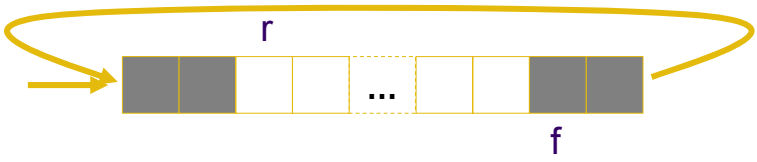


Code implementate con array

- r indicizza sempre un'entrata vuota

→ un array con N entrate accoglie al più N-1 elementi

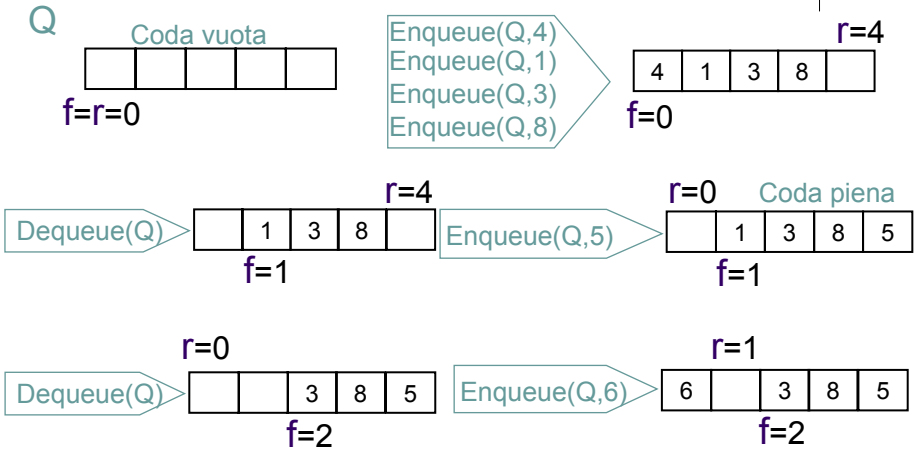
se f=r allora la coda è vuota



Code implementate con array

soluzione efficiente

- Esempio:



Code implementate con array

soluzione efficiente

Tempo O(1)

```
Enqueue(Q,x)
  if size(Q) = N – 1  //N e` la lunghezza dell'array
   then throw FullQueueException
   else
    Q[r ] ← x
    r ← (r + 1) mod N
```

Code implementate con array

soluzione efficiente

Tempo O(1)

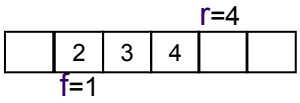
```
Dequeue(Q)
  if isEmpty(Q)  then
    throw EmptyQueueException
   else
    x ← Q[f]
    f← (f+ 1) mod N
  return x
```

Code implementate con array

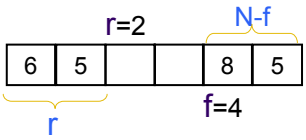
soluzione efficiente

Tempo O(1)

```
Size(Q)
    return (N-f+r)mod N
```



f≤r : size = r-f =(N+(r-f))mod N



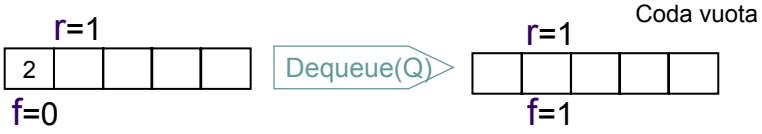
f>r : size = (N-f)+r = (N-f+r)mod N

Code implementate con array -10

soluzione efficiente

Tempo O(1)

```
IsEmpty(Q)
    return (f=r)
```



Esercizio 1

- Implementare l'interfaccia Queue (scrivere la classe ArrayQueue) usando un array di lunghezza fissata
  - public static final int CAPACITY = 1024;
- Scrivere un programma che testi la vostra implementazione di Queue invocandone tutti i metodi

Esercizio 2

- Modificare la classe ArrayQueue in modo che se il TDA Queue è pieno venga allocato dello spazio sufficiente a contenere nuovi elementi e non venga lanciata l'eccezione **FullQueueException**

Esercizio 3

- Descrivere come implementare una coda usando due stack
  - Analizzare la complessita` di enqueue() e dequeue()
- Descrivere come implementare uno stack usando due code
  - Analizzare la complessita` di pop() e push()
- Descrivere come implementare uno stack usando una coda
  - Analizzare la complessita` di pop() e push()
- Descrivere un algoritmo per invertire una coda in tempo linerare. Per accedere alla coda si possono utilizzare solo i metodi del TDA coda