Attività di didattica di supporto per Fondamenti di Informatica

Esercitazioni sulle prove di esame

AA 2019-20

Sim6 Multiqueue

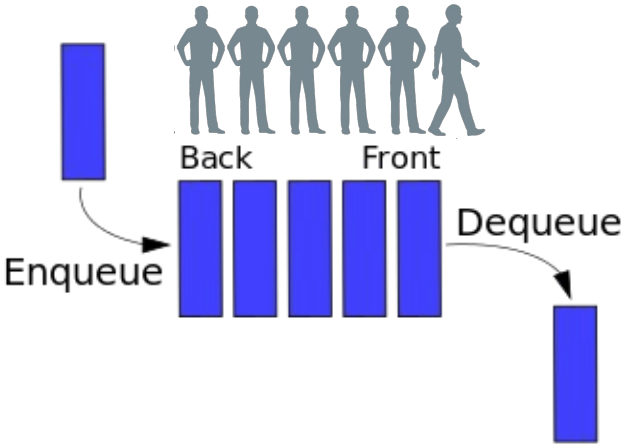
giulio.martini@igi.cnr.it

Slide originali a cura di Andrea Rigoni

TEMA DI ESAME:
Biglietteria con N sportelli

La coda

In informatica per coda (queue) si intende una struttura dati di tipo **FIFO**, First In First Out (il primo in ingresso è il primo ad uscire).



Operazioni sulla coda:

Accodamento di un elemento

Detta anche operazione di ***enqueue***, serve a mettere un elemento in coda.

Estrazione di un elemento

Detta anche operazione di ***dequeue***, serve a rimuovere un elemento dalla testa della coda.

Una possibile interfaccia per definire una coda potrebbe essere:

- **enqueue()** per inserire un oggetto nella coda
- **dequeue()** per esaminare ed eliminare dalla coda l'oggetto che vi si trova da più tempo
- **front()** per esaminare l'oggetto che verrebbe eliminato da dequeue(), senza estrarlo

Multiqueue

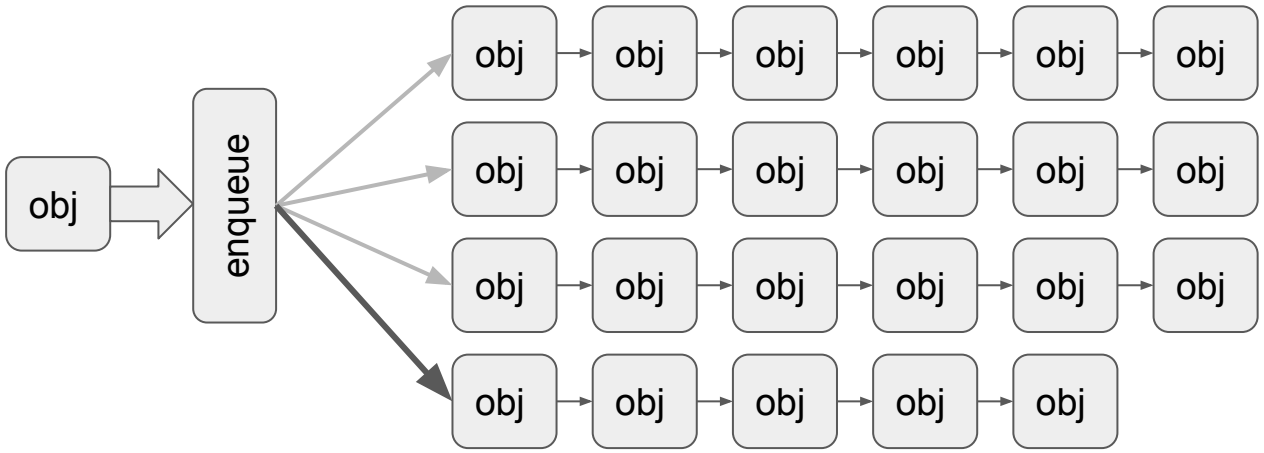
La **coda multipla** o multiqueue e' una struttura dati astratta che definisce una collezione di code di attesa (FIFO) osservando una proprietà specifica per l'inserimento. Ad esempio in questo caso la proprietà e' il bilanciamento stretto nel numero di elementi delle code.

In questo senso la struttura della multi coda ha molto a che fare con la **hash-table.** Generalmente si considera la funzione di hashing dipendente solo dal valore dell'oggetto inserito e non da altri fattori, mentre più genericamente la coda multipla può avere un legame anche con la coda stessa.

ESEMPIO:

Suddivisione di code di attesa allo sportello per classi di eta' -> **HashTable**

Suddivisione di code per massimizzare la velocità di svuotamento -> **CodaMultipla**



Esempio di Dizionario - Il traduttore

Si vuole gestire una rubrica telefonica realizzando un **dizionario** che contiene **coppie di tipo “nome traduzione”**. Il campo “nome” è una stringa che ha la funzione di chiave per il dizionario, mentre il campo “traduzione” deve corrispondere alle possibili traduzioni in italiano delle

Materiale fornito:

- Un’interfaccia per la coda (**Queue**) e una per la multicoda (**MultiQueue**)
- Un’eccezione EmptyQueueException
- Classe ArrayQueue e ArrayMultiQueue
- Classe MultiQueueTester con il metodo main
- File di prova

Queue Interface

```
interface Queue
{   // Restituisce true se la coda e` vuota. Restituisce false se la coda
    // contiene almeno un elemento
    boolean isEmpty();

    // Svuota la coda
    void makeEmpty();

    // Restituisce il numero di elementi presenti nella coda
    int size();

    // Inserisce l'oggetto obj nella coda
    void enqueue(Object obj);

    // Elimina dalla coda il primo oggetto, e lo restituisce.
    // Lancia EmptyQueueException se la coda e` vuota
    Object dequeue() throws EmptyQueueException;

    // Restituisce il primo oggetto della coda, senza estrarlo
    // Lancia EmptyQueueException se la coda e` vuota
    Object getFront() throws EmptyQueueException;
}
```

MultiQueue Interface

```
interface MultiQueue
{
    // Restituisce true se la multicoda e` vuota, cioe` se tutte le N
    // code della multicoda sono vuote. Restituisce false se la multicoda
    // contiene almeno un elemento, cioe` se almeno una delle N code della
    // multicoda contiene almeno un elemento
    boolean isEmpty();

    // Svuota la multicoda, cioe` svuota tutte le N code della multicoda
    void makeEmpty();

    // Inserisce l'oggetto obj nella multicoda. Tra tutte le N code della
    // multicoda, l'oggetto viene accodato a quella che contiene il minor
    // numero di elementi. Nel caso in cui piu` code contengano un numero
    // di elementi pari al minimo, la scelta è indifferente
    void add(Object obj);

    // Disaccoda dalla i-esima coda il suo primo elemento e lo restituisce.
    // L'indice intero i specifica quale e` la coda da cui disaccodare il
    // primo elemento. Di conseguenza i deve essere tale che 0 <= i < N.
    // Lancia EmptyQueueException se la la i-esima coda e` vuota
    Object remove(int i) throws EmptyQueueException;
}
```

Richieste:

STEP 1

Implementare la classe ArrayQueue che implementa Queue

- La classe deve inoltre sovrascrivere il metodo toString, che restituisca una stringa contenente gli elementi della coda, ciascuno su una riga diversa.

STEP 2

Implementare la classe ArrayMultiQueue che implementa MultiQueue

- La classe deve realizzare un **costruttore** che riceve un parametro esplicito intero N > 0 e crea una multicoda vuota, costituita da una sequenza di N code vuote.
- La classe deve inoltre sovrascrivere il metodo **toString**, che restituisca una stringa contenente gli elementi delle N code della multicoda, secondo il seguente formato:
  - gli elementi delle N code vengono scritti in sequenza, secondo il formato del metodo toString della classe ArrayQueue (si veda il corpo di tale metodo),
  - la scrittura degli elementi della i-esima coda ($0 \leq i < N$) comincia con la riga “CODA i:”.

STEP 3

Implementare la classe MultiQueueTester

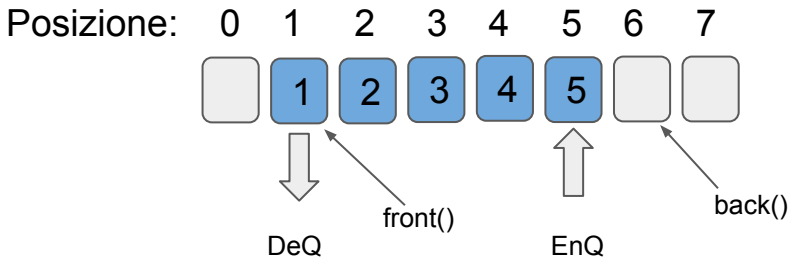
- riceve un numero intero N come argomento sulla riga di comando
- crea un oggetto di tipo ArrayMultiQueue, contenente N code, inizialmente vuote;
- accetta ripetutamente comandi dall'utente, introdotti da tastiera, finchè l'utente non introduce il comando di terminazione del programma; i comandi disponibili sono:
  - **A** -> add
  - **R** -> remove
  - **P** -> print
  - **Q** -> quit

STEP 1: ArrayQueue

Soluzione: struttura di base della classe ArrayQueue

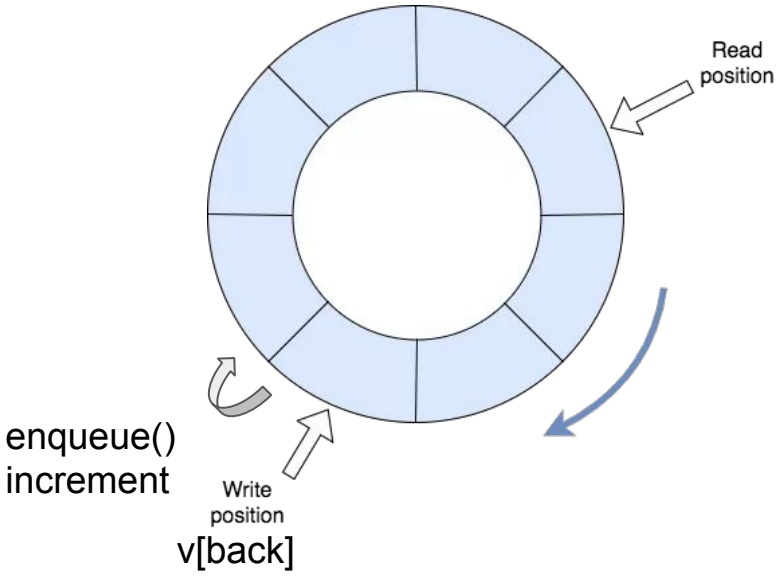
```
class ArrayQueue implements Queue
{
    public ArrayQueue()
    {   v = new Object[INITSIZE];
        makeEmpty(); }

    public void makeEmpty()
    {  front = back = 0; }

    public boolean isEmpty()
    {  return (back == front); }

    public int size()
    {
       if(back>=front)
         return (back - front);
       else
         return (back + v.length - front);
    }

    private Object[] v;
    private int front, back;
    private static int INITSIZE = 10;
}
```

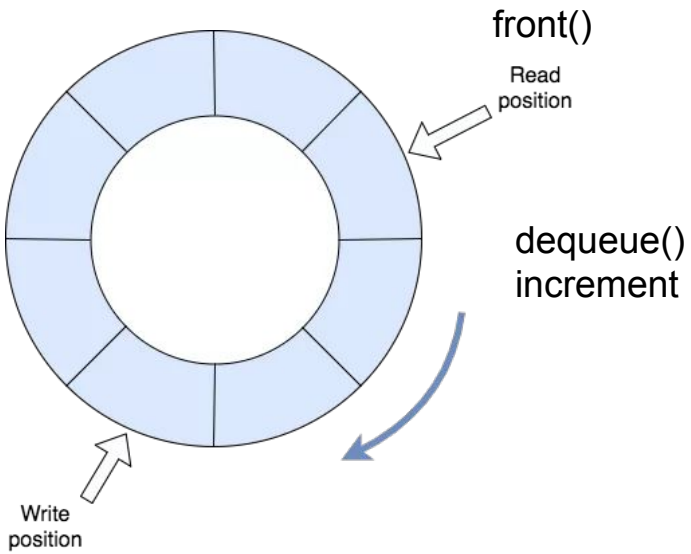
La soluzione più efficiente per l'implementazione di una coda con Array e' il buffer circolare



insert()

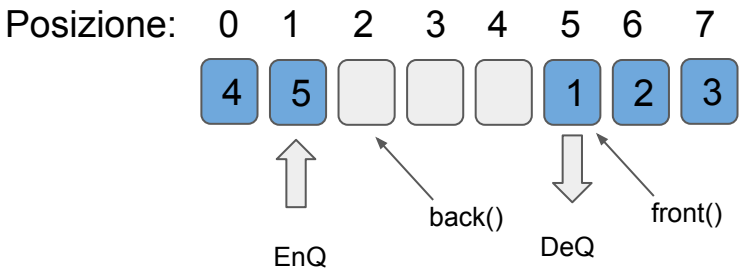
```
public void enqueue(Object obj)
{

    if (increment(back) == front)
       resize();

    v[back] = obj;
    back = increment(back);
}

public Object dequeue()
{
    Object obj = getFront();
    front = increment(front);
    return obj;
}


private int increment(int index)
{
    return (index + 1) % v.length;
}
```

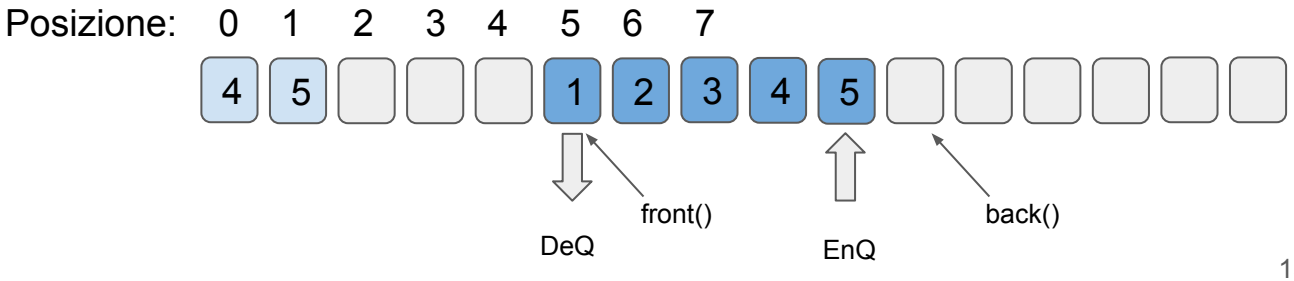




resize()

```
//metodo ausiliario: lo rendo private, non deve essere usato da altri
private void resize()
{
    Object[] newv = new Object[2*v.length];
    System.arraycopy(v, 0, newv, 0, v.length);
    v = newv;

    if (back < front)
    {
        System.arraycopy(v, 0, v, v.length/2, back);
        back += v.length/2;
    }

}
```





getFront, toString

```
public Object getFront()
{   if (isEmpty()) throw new EmptyQueueException();
     return v[front]; }
```

```
//metodo toString
public String toString()
{
    String s = "";
    if (front <= back)
    {
        for (int i = front; i<back; i++)
        {
            s = s + v[i] + "\n";
        }
        s = s + "\b";

    }
    else
    {
        for (int i = front; i<v.length; i++)
        {
            s = s + v[i] + "\n";
        }

        for (int i = 0; i<back; i++)
        {
            s = s + v[i] + "\n";
        }

        s = s + "\b";
    }

    return s;
}
```

STEP 2: ArrayMultiQueue

ArrrayMultiQueue

```
class ArrayMultiQueue implements MultiQueue
{
    // variabile esemplare
    private Queue[] code;

    // COSTRUTTORE
    // crea una multicoda vuota, costituita da una sequenza di N code vuote
    public ArrayMultiQueue(int N)
    {   if (N<=0) throw new IllegalArgumentException();
        code = new Queue[N];
        for (int i = 0; i < code.length; i++)
            code[i] = new ArrayQueue();
    }

    public boolean isEmpty()
    {
        for (int i = 0; i < code.length; i++)
            if (!code[i].isEmpty()) return false;
        return true;
    }

    public void makeEmpty()
    {   for (int i = 0; i < code.length; i++)
            code[i].makeEmpty();
    }
}
```

add, remove, toString

```
public void add(Object obj)
    {
        if (!(obj instanceof String) )
            throw new IllegalArgumentException();

        int min = 0;
        for (int i = 1; i < code.length; i++)
            if (code[i].size() < code[min].size())
                min = i;

        code[min].enqueue(obj);
    }

    public Object remove(int i)
    {
        if (i<=0 || i >= code.length) throw new IllegalArgumentException();
        return code[i].dequeue();
    }

    //metodo toString ..... da completare .........
    public String toString()
    {
        String s = "";
        for (int i = 0; i < code.length; i++)
            s = s + "\nCODA " + i + ":\n" + code[i];
        return s;
    }
```

STEP 3: MultiQueueTester

Nuova istanza di ArrayMultiQueue

- Viene letto N da args

```
if ( args.length != 1 || !( args[0].matches("[0-9]+") ) ||
     Integer.parseInt(args[0]) <= 0  )
{   System.out.println(
              "Uso: \"$MultiQueueTester N\", con N intero positivo");
    System.exit(1);
}

int N = Integer.parseInt(args[0]);
```

- Viene creata l'istanza per la multicoda

```
MultiQueue b = new ArrayMultiQueue(N);
```

Lettura comandi da System.in

```
Scanner in = new Scanner(System.in);
boolean done = false;

while (!done)
{   System.out.println( "Comando? A=aggiungi,R=rimuovi,P=stampa,Q=quit" );
    String cmd = in.nextLine();
    if (cmd.equalsIgnoreCase( "Q") )
    {   System.out.println( "La biglietteria chiude, arrivederci" );
        done = true;
    }
    else
    {   if (cmd.equalsIgnoreCase( "A") )
        {   System.out.println( "Nome persona da aggiungere?" );
            String persona = in.nextLine();
            b.add(persona);
            System.out.println( "Aggiunto " + persona + " in coda");
        }
        else if (cmd.equalsIgnoreCase( "R") )
        {   System.out.println( "Numero coda da cui rimuovere?" );
            int i = 0;
            try{
                i = Integer.parseInt(in.nextLine());
                String persona = (String) b.remove(i);
                System.out.println( "Rimosso "+persona+" da coda "+i);
                }
            catch(EmptyQueueException e){
              System.out.println( "La coda "+i+ " e` vuota!");
                }
            catch(NumberFormatException e){
              System.out.println( "Devi inserire un intero 0<=i<" + N);
                }
            catch(IllegalArgumentException e){
              System.out.println( "Devi inserire un intero 0<=i<" + N);
                }
        }
        else if (cmd.equalsIgnoreCase( "P") )
        {   System.out.println( "Situazione della biglietteria:" );
            System.out.println(b);
        }
    }
}
```