Attività di didattica integrativa per Fondamenti di Informatica

Esercitazioni sulle prove di esame

AA 2023/2024

Sim3 Priority Queue

giulio.martini@igi.cnr.it

Slide originali a cura di Andrea Rigoni

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).

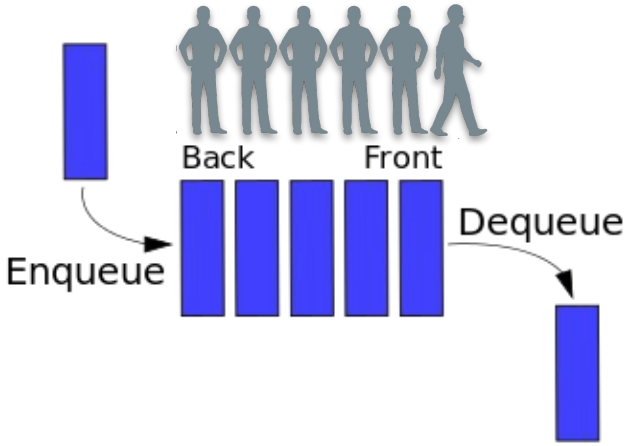
Accodamento di un elemento

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.

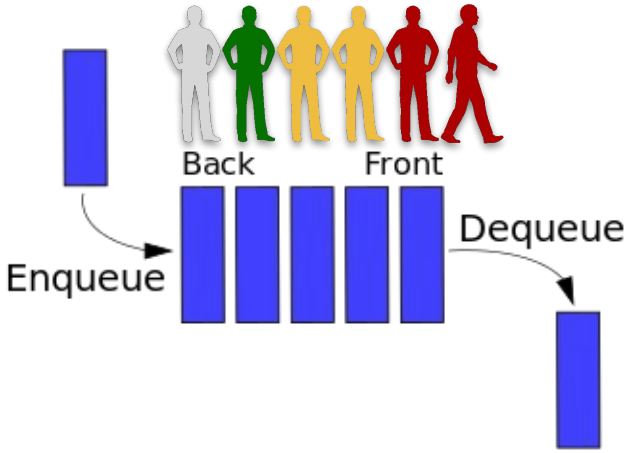
Come detto l'ordine di uscita e' basato unicamente per rispettare l'ordine di arrivo.



La coda di priorità

Una **coda di priorità** è una struttura dati astratta, simile ad una coda o ad una pila, ma diversa da queste in quanto ogni elemento inserito all'interno della coda possiede una sua "priorità". **In una coda di priorità, ogni elemento avente priorità più alta, viene inserito prima rispetto ad un elemento avente priorità più bassa**. In particolare, l'elemento con priorità più alta si trova in testa alla coda, quello con priorità più bassa si troverà, appunto, in coda.

Un esempio di coda di priorità' e' la gestione dei pazienti del pronto soccorso: Viene comunque prestato soccorso in ordine di arrivo ma viene anche assegnato un codice di precedenza basato sulla urgenza.



TEMA DI ESAME:
Agenda Elettronica

Interfaccia

```
interface PriorityQueue //non modificare!!
{
    //    svuota la coda di priorita`
    void makeEmpty();

    //    restituisce true se la coda e' vuota, false altrimenti
    boolean isEmpty();

    /*  Metodo di inserimento
        Inserisce la coppia "chiave valore" nella coda di priorita`. Notare che
        la coda di priorita` puo` contenere piu` coppie con la stessa chiave.
        Questo perche` il campo chiave non serve ad identificare univocamente
        un elemento (come nel caso di un dizionario), ma serve invece a definire
        la priorita` di un elemento. E` ovvio che piu` elementi possono avere
        la stessa priorita`.  */
    void insert (int key, Object value);

    /*  Metodo di rimozione
        Rimuove dalla coda la coppia con chiave minima, e restituisce un
        riferimento al suo campo value. Se sono presenti piu` coppie con chiave
        minima, effettua la rimozione di una qualsiasi delle coppie con chiave
        minima (ad es. la prima coppia con chiave minima che e` stata trovata)
        Lancia EmptyQueueException se la coda di priorita` e` vuota    */
    Object removeMin() throws EmptyQueueException;

    /*  Metodo di ispezione
        Restituisce un riferimento al campo value della coppia con chiave minima
        (ma *non* rimuove la coppia dalla coda).  Se sono presenti piu` coppie
        con chiave minima, restituisce il campo value di una qualsiasi delle
        coppie con chiave minima (ad esempio la prima coppia con chiave minima
        che e` stata trovata). Lancia EmptyQueueException se la coda e` vuota    */
    Object getMin() throws EmptyQueueException;
}
```

DataType: Impegno

```
/*
    classe privata Impegno: rappresenta gli elementi della classe Agenda ed
    e` costituita da coppie "chiave valore" in cui il campo chiave e` di
    tipo int e rappresenta la priorita` dell'impegno, e il campo valore e`
    una stringa contenente un promemoria dell'impegno. Si considerano 4
    livelli di priorita`, numerati da 0 a 3. Conseguentemente il campo
    chiave puo` assumere valori solo in questo intervallo, dove il valore 0
    significa "massima priorita`" e il valore 3 significa "minima priorita`"
*/
private class Impegno //non modificare!!
{   public Impegno(int priority, String memo)
    {   if (priority>3 || priority<0) throw new IllegalArgumentException();
        this.priority = priority;
        this.memo = memo;
    }
    // metodi (pubblici) di accesso
    public int getPriority()
    {   return priority; }
    public Object getMemo()
    {   return memo; }
    //metodo toString sovrascritto
    public String toString()
    {   return priority + " " + memo; }
    //campi di esemplare (privati) della classe Impegno
    private int priority; //priorita` dell'impegno (da 0 a 3)
    private String memo; //promemoria dell'impegno
}
```

Richieste:

STEP 1

Scrivere la classe Agenda che implementa l'interfaccia PriorityQueue.
L'agenda conterrà coppie di tipo "chiave valore" appartenenti alla classe **Impegno**.
Si richiede inoltre di realizzare un metodo toString per la classe Agenda, che restituisca una stringa contenente gli elementi secondo il seguente formato: (1) ogni coppia viene scritta su una riga diversa, e (2) all'interno di ogni riga la coppia viene scritta seguendo il formato specificato dal metodo toString della classe Impegno (si veda il corpo di tale metodo).

STEP 2

Implementare la classe AgendaTester che contiene il metodo main() che:

- Crea un oggetto di tipo Agenda, vuoto.
- Accetta ripetutamente comandi dall'utente, introdotti da tastiera, finchè l'utente non introduce il comando di terminazione del programma. ("I", "R", "L", "Q")
- Dopo ognuna delle operazioni sopra elencate stampa il contenuto aggiornato dell'agenda.

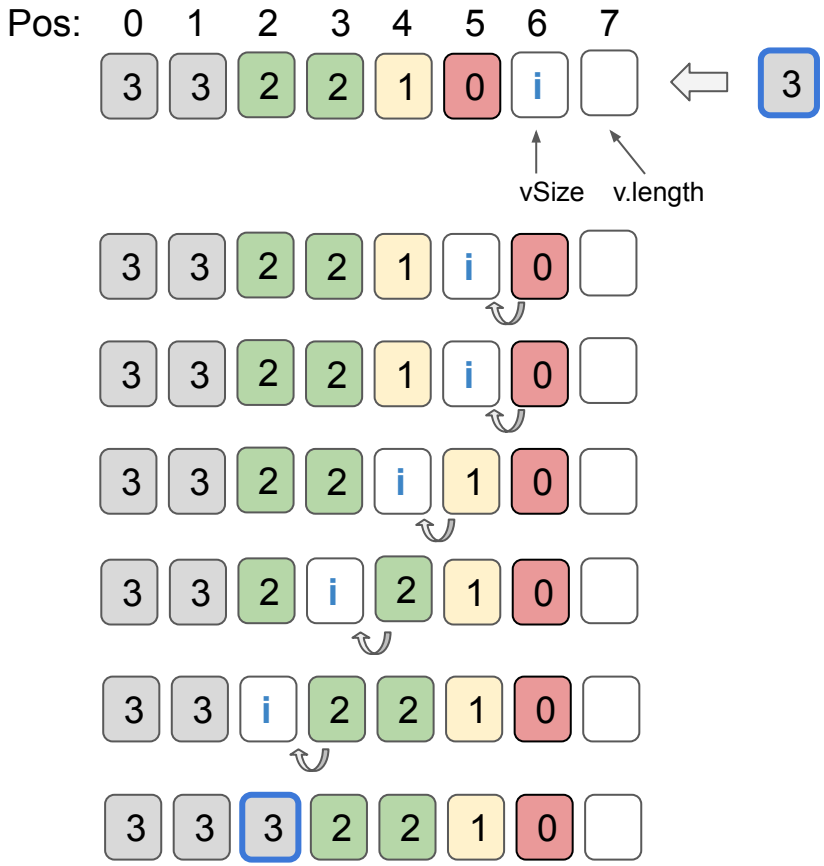
STEP 1:

Agenda

```
class Agenda implements PriorityQueue
{
    // campi di esemplare di Agenda
    Impegno[] v;
    int vSize;

    // COSTRUTTORE
    public Agenda() {
        v = new Impegno[1];
        makeEmpty();
    }

    // isEmpty
    public boolean isEmpty() { return vSize == 0; }

    // clear elements
    public void makeEmpty()  { vSize = 0; }
}
```

insert

```
public void insert (int key, Object value)
{   if (vSize == v.length)
        v = resize(2*v.length);
    //insertionSort: O(n), perche` inseriamo in un array ordinato
    //attenzione: l'array e` ordinato con le chiavi piu` alte all'inizio e
    //quelle piu` basse alla fine: cosi` removeMin e` O(1)
    int i = vSize;
    while (i > 0 && v[i-1].getPriority() < key) {
        v[i] = v[i-1];
        i--;
    }
    v[i] = new Impegno(key, (String)value);
    vSize++;
}
```



resize

```
private Impegno[] resize(int newLength)
{ if (newLength <v.length)
     throw new IllegalArgumentException();
  Impegno[] newv = new Impegno[newLength];
  System.arraycopy(v, 0, newv, 0, v.length);
  return newv;
}
```

Complessità

Se gli elementi con priorità più bassa vengono posizionati all'inizio dell'array, il metodo **removeMin**, che rimuove l'elemento di priorità massima, agirà "in cima" all'array ed avrà di conseguenza prestazioni **O(1)**

Se avessimo mantenuto ordinato l'array in senso opposto il metodo removeMin avrebbe avuto prestazioni O(n) perché si sarebbe dovuto ricompattare l'array dopo la rimozione del primo elemento

il metodo **insert** è comunque **O(n)** perché bisogna usare insertionSort per mantenere ordinato l'array, qualunque sia il senso dell'ordinamento

NOTA:

L’implementazione proposta rispetta la priorità ma non la precedenza di arrivo a pari priorità. Come si potrebbe modificare l’algoritmo per aggiungere la precedenza?

removeMin, getMin, toString

```
public Object removeMin() throws EmptyQueueException
{   if (isEmpty())  throw new EmptyQueueException();
    vSize--;
    return v[vSize].getMemo();
}

public Object getMin() throws EmptyQueueException
{   if (isEmpty())  throw new EmptyQueueException();
    return v[vSize-1].getMemo();
}

public String toString()
{   String s = "";
    for (int i = vSize-1; i >= 0; i-- )
       s = s + v[i] + "\n";
    return s;
}
```

STEP 2:

AgendaTester

- Si parte da una agenda vuota
- Set di istruzioni:
  - “I” (inserisci impegno in formato "chiave valore")
  - “R” (rimuovi il primo impegno di priorità massima)
  - “L” (legge il primo impegno di priorità massima)
  - “Q” (termina il programma)
- Dopo ogni operazione stampa il contenuto aggiornato dell’agenda.
- Input: impegni.txt

  java AgendaTester < impegni.txt

AgendaTester

```
public class AgendaTester
{   public static void main(String[] args) throws IOException
    {   Agenda a = new Agenda();
        Scanner in = new Scanner(System.in);
        boolean done = false;
        while (!done)
        {   System.out.println("Comando? I=inserisci,R=rimuovi,L=leggi,Q=quit");
            String cmd = in.nextLine();
            if (cmd.equalsIgnoreCase("Q") )
            {   System.out.println("Ciao");
                done = true;
            }
            else
            {   if (cmd.equalsIgnoreCase("I") )
                {   System.out.println("Inserisci impegno");
                    Scanner line = new Scanner(in.nextLine());
                    int key = Integer.parseInt(line.next());
                    String value = "";
                    while (line.hasNext())
                        value = value + line.next() + " ";
                    a.insert(key, value);
                }
                else if (cmd.equalsIgnoreCase("R") )
                {   String value = (String)a.removeMin();
                    System.out.println("Rimosso impegno piu` urgente: " +value);
                }
                else if (cmd.equalsIgnoreCase("L") )
                {   String value = (String)a.getMin();
                    System.out.println("L'impegno piu` urgente e`: " + value);
                }
                System.out.println("Contenuto dell'agenda:");
                System.out.println(a);
            }
        }
    }
}
```