UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED ELETTRICA
E MATEMATICA APPLICATA

Corso:
DESIGN AND ANALYSIS OF ALGORITHMS

APPUNTI

Carmine Terracciano

Mat. IE22700109

ANNO ACCADEMICO 2025/2026

Sommario

Capitolo 1

Array Stack

```
class Empty(Exception):
  pass

class ArrayStack:
  """Implementazione di ADT Stack che utilizza un oggetto list di Python
     per la memorizzazione."""

  def __init__(self):
    """Crea uno stack vuoto."""
    self._data = []                # instanza di list non pubblica

  def __len__(self):
    """Restituisce il numero di elementi nello stack."""
    return len(self._data)

  def is_empty(self):
    """Restituisce True se lo stack è vuoto."""
    return len(self._data) == 0

  def push(self, e):
    """Aggiunge l'elemento e al top dello stack."""
    self._data.append(e)    # il nuovo elemento è aggiunto in coda alla list

  def top(self):
    """Restituisce (m anon rimuove) l'elemento al top dello stack.
       Raise Empty exception se lo stack è vuoto."""
    if self.is_empty():
      raise Empty('lo stack è vuoto')
    return self._data[-1]          # legge l'ultimo elemento della list
```

```
29
30  def pop(self):
31    """Rimuove e restituisce l'elemento al top dello stack.
32       Raise Empty exception se lo stack è vuoto."""
33    if self.is_empty():
34      raise Empty('lo stack è vuoto')
35      # print("lo stack è vuoto")
36    return self._data.pop()               # rimuove l'ultimo elemento della
        list
```

Listing 1.1: Implementazione Python dell'ADT Stack utilizzando una lista per la memorizzazione degli elementi.

Capitolo 2

Array Queue

```
class Queue:
  """Classe astratta che implementa l'ADT Queue."""

  def __len__(self):
    """Restituisce il numero di elementi nella coda."""
    raise NotImplementedError('deve essere implementato dalla sottoclasse.')

  def is_empty(self):
    """Restituisce True se la coda è vuota."""
    raise NotImplementedError('deve essere implementato dalla sottoclasse.')

  def first(self):
    """Restituisce (ma non rimuove) l'elemento al front della coda.
        Raise Empty exception se la coda è vuota."""
    raise NotImplementedError('deve essere implementato dalla sottoclasse.')

  def dequeue(self):
    """Rimuove e restituisce l'elemento al front della coda.
        Raise Empty exception se la coda è vuota."""
    raise NotImplementedError('deve essere implementato dalla sottoclasse.')

  def enqueue(self, e):
    """Aggiunge un elemento al back della coda."""
    raise NotImplementedError('deve essere implementato dalla sottoclasse.')
```

Listing 2.1: Classe astratta che implementa l'ADT Queue.

```python
from .queue import Queue

class Empty(Exception):
  pass

class ArrayQueue(Queue):
  """Implementazione di ADT Queue basata sul tipo list di Python usato dome
      array circolare."""
  DEFAULT_CAPACITY = 10          # dimensione di default di nuove code

  def __init__(self):
    """Crea una coda vuota."""
    self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
    self._size = 0
    self._front = 0

  def __len__(self):
    """Restituisce il numero di elementi nella coda."""
    return self._size

  def is_empty(self):
    """Restituisce True se la coda è vuota."""
    return self._size == 0

  def first(self):
    """Restituisce (ma non rimuove) l'elemento al front della coda.
       Raise Empty exception se la coda è vuota.
    """
    if self.is_empty():
      raise Empty('Queue is empty')
    return self._data[self._front]

  def dequeue(self):
    """Rimuove e restituisce l'elemento al front della coda.
       Raise Empty exception se la coda è vuota.
    """
    if self.is_empty():
      raise Empty('Queue is empty')
    answer = self._data[self._front]
    self._data[self._front] = None         # favorisce garbage collection
    self._front = (self._front + 1) % len(self._data)
    self._size -= 1
    return answer
```

```
43
44  def enqueue(self, e):
45    """Aggiunge un elemento al back della coda."""
46    if self._size == len(self._data):
47      self._resize(2 * len(self._data))     # raddoppia la dimensione
          dell'array se pieno
48    avail = (self._front + self._size) % len(self._data)
49    self._data[avail] = e
50    self._size += 1
51
52  def _resize(self, cap):           # we assume cap >= len(self)
53    """Ridimensiona l'array portandolo a lunghezza cap."""
54    old = self._data                # conserva la vecchia copia dell'array
55    self._data = [None] * cap       # alloca una nuova list di dimensione
        cap
56    j = self._front
57    for k in range(self._size):
58      self._data[k] = old[j]        # shifta gli indici per riallinearli
59      j = (j + 1) % len(old)        # usa la vecchia dimensione come modulo
60    self._front = 0                 # front riallineato a 0
```

Listing 2.2: Implementazione Python dell'ADT Queue utilizzando una lista per la memorizzazione degli elementi.