De Faveri Tomas

TAD PILA

```
from algo1 import *
from linkedlist import *

"""
push(S,element)

Descripción: Agrega un elemento al comienzo de S, siendo S una
estructura de tipo LinkedList
Entrada: La pila S sobre la cual se quiere agregar el elemento
(LinkedList) y el valor del elemento (element) a agregar.
Salida: No hay salida definida
"""

def push(S,element):
    add(S,element)



"""

pop(S)
Descripción: extrae el primer elemento de la pila S, siendo S
una estructura de tipo LinkedList
Poscondición: Se debe desvincular el Node a eliminar.
Entrada: la pila S (Linkedlist) sobre el cual se quiere realizar
la eliminación
Salida: Devuelve el elemento eliminado. Devuelve None si la pila
está vacía.
"""

def pop(S):
    if S.head != None:
        value = S.head.value
        S.head = S.head.nextNode
        return value
    else:
        return None
```

De Faveri Tomas

TAD COLA

```
from algo1 import *
from linkedlist import *

"""
enqueue(Q,element)

Descripción: Agrega un elemento al comienzo de Q, siendo Q una
estructura de tipo LinkedList.
Entrada: La cola Q (LinkedList) sobre la cual se quiere agregar
el elemento y el valor del elemento (element) a agregar.
Salida: No hay salida definida.
"""

def enqueue(Q, element):
    add(Q, element)

"""
dequeue(Q)

Descripción: extrae el último elemento de la cola Q, siendo Q
una estructura de tipo LinkedList.
Poscondición: Se debe desvincular el Node a eliminar.
Entrada: la cola Q (Linkedlist) sobre el cual se quiere realizar
la eliminación.
Salida: Devuelve el elemento de la cola. Devuelve None si la
cola está vacía.
"""

def dequeue(Q):

    value = None

    if Q.head != None and length(Q) > 1:
        currentNode = Q.head
        while currentNode.nextNode.nextNode != None:
            currentNode = currentNode.nextNode

        value = currentNode.nextNode.value
        currentNode.nextNode = None

    elif Q.head != None and length(Q) == 1:
```

```
        value = Q.head.value
        Q.head = None


    return value
```

TAD PRIORITY QUEUE

```
from algo1 import *
from linkedlist import *
from mystack import *
from myqueue import *


class PriorityQueue:
    head=None

class PriorityNode:
    value=None
    nextNode=None
    priority=None


def mostrarPriorityQueueConTexto(L, t):
    current = L.head
    print(t + ": [")
    while current != None:
        if current.nextNode != None:
            print("--- V: " + str(current.value) + " - PRIORITY: " +
str(current.priority) + ", ")
        else:
            print("--- V: " + str(current.value) + " - PRIORITY: " +
str(current.priority))
        current = current.nextNode
    print("]")



"""
Crear un módulo de nombre mypriorityqueue.py que implemente una cola
con prioridad.
```

De Faveri Tomas

```
Una cola con prioridad es un TAD similar a una cola en la que los
elementos tienen
adicionalmente, una prioridad asignada. En una cola de prioridades un
elemento con mayor
prioridad será encolado antes que un elemento de menor prioridad. Si
dos elementos tienen
la misma prioridad, se desencolarán siguiendo el orden de cola. (Fifo o
Lifo?)
"""

"""
enqueue_priority(Q,element,priority)
Descripción: Agrega un elemento a Q con la prioridad priority
(entero), siendo Q una estructura de tipo PriorityQueue
Entrada: La cola Q sobre la cual se quiere agregar el elemento
(PriorityQueue), el valor del elemento (element) a agregar y un
número que indica la prioridad.
Salida: Devuelve la posición donde se inserto el elemento.

"""
def enqueue_priority(Q,element,priority):

    newNode = PriorityNode()
    newNode.value = element
    newNode.priority = priority

    position = 0


    if Q.head == None:
        Q.head = newNode

    elif priority >= Q.head.priority and Q.head != None:
        newNode.nextNode = Q.head
        Q.head = newNode

    else:
        currentNode = Q.head

        while (currentNode.nextNode != None) and (priority <
currentNode.nextNode.priority):
            position += 1
            currentNode = currentNode.nextNode
```

```
        position += 1
        newNode.nextNode = currentNode.nextNode
        currentNode.nextNode = newNode

    return position




"""
dequeue_priority(Q)
Descripción: extrae el primer elemento de la cola Q con la mayor
prioridad (un valor mayor del campo priority, indica una mayor
prioridad), siendo Q una estructura de tipo PriorityQueue
Poscondición: Se debe desvincular el Node a eliminar.
Entrada: la cola sobre el cual se quiere realizar la eliminación
(PriorityQueue)
Salida: Devuelve el elemento con mayor prioridad. Devuelve None
si la cola está vacía.

"""

def deletePriorityNode(Q, nodeToDelete):
    currentNode = Q.head

    if Q.head != nodeToDelete:

        while currentNode.nextNode != nodeToDelete:
            currentNode = currentNode.nextNode

        currentNode.nextNode = currentNode.nextNode.nextNode
    else:
        Q.head = Q.head.nextNode

def dequeue_priority(Q):

    if Q.head != None:
        currentNode = Q.head

        maxPriority = Q.head.priority

        while currentNode.nextNode != None:
```

```
            if (currentNode.nextNode.priority < maxPriority):
                break
            else:
                currentNode = currentNode.nextNode

        value = currentNode.value
        deletePriorityNode(Q, currentNode)

    else:
        value = None


    return value
```

```
----------------------------------------------------------------------
Ran 21 tests in 0.003s

OK
```