



CICLO 1

[FORMACIÓN POR CICLOS]

Fundamentos de **PROGRAMACIÓN**



Ingeni@
Soluciones TIC



**UNIVERSIDAD
DE ANTIOQUIA**

Facultad de Ingeniería



Lectura

COLAS

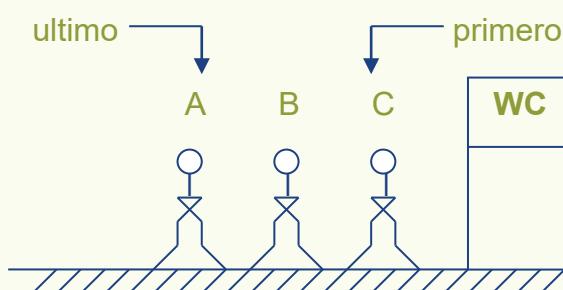


Introducción

Hasta ahora hemos estado tratando colecciones de datos en las cuales las operaciones de inserción y borrado se efectúan en forma aleatoria, es decir, dichas operaciones se pueden hacer al principio, al final o en un sitio intermedio. Hay situaciones en las que dichas operaciones siguen siempre un patrón definido, por ejemplo, que las operaciones de inserción se efectúen solo al final de la colección o solo al principio, o que las operaciones de borrado se efectúen solo al principio o solo al final. Dependiendo de la forma de procesamiento que se adopte, se define la estructura. En esta lectura trataremos colecciones en las cuales la estrategia a seguir es que el primero que entra es el primero que sale. En otras palabras, las operaciones de inserción se hacen siempre al final y las de borrado, al principio. Una colección de datos que se trabaje de esta forma recibe el nombre de cola.

Definición.

Una cola es una lista ordenada en la cual las operaciones de inserción se efectúan en un extremo llamado último y las operaciones de borrado se efectúan en el otro extremo llamado primero. Es una estructura FIFO (First Input First Output).



En términos prácticos es lo que supuestamente se debe hacer para tomar un bus, para comprar las boletas de un cine o parautilizar un servicio público. Las operaciones sobre una cola son:

crear: crea una cola vacía.

esVacia(): retorna verdadero si la cola está vacía, falso de lo contrario.

esLlena(): retorna verdadero si la cola está llena, falso de lo contrario.

encolar(d): inserta un dato d al final de la cola.

desencolar(): remueve el primer elemento de la cola.

siguiente(): retorna el dato que se halla de primero en la cola.

Representación de colas en un vector.

Definiremos la clase cola derivada de la clase vector. Manejaremos la cola circularmente en el vector. Para manejar una cola circularmente en un vector se requiere definir un vector de **n** elementos con los subíndices en el rango desde 0 hasta **n - 1**, es decir, si el vector tiene 10 elementos, los subíndices variarán desde 0 hasta 9. Adicionalmente se manejan dos variables: **primero** y **ultimo**: **primero** apunta hacia la posición anterior en la que realmente se halla el primer dato de la cola, y **ultimo** apunta hacia la posición en la que realmente está el último dato de la cola. La operación de encolar consiste en sumarle 1 a **ultimo** y asignar a esa posición del vector el dato a encolar, mientras que desencolar consiste en sumerle 1 a **primero** y retornar el dato que se halla en esa posición. El incremento de las variables **primero** y **ultimo** se hace utilizando la operación **módulo (%)**, de la siguiente manera:

$$\text{primero} = (\text{primero} + 1) \% n$$

$$\text{ultimo} = (\text{ultimo} + 1) \% n$$

Recuerde que la operación **módulo (%)** retorna el residuo de una división entera. Si se tiene una cola en un vector, con la siguiente configuración:

0	1	2	3	4	5	6
V			b	c		

n vale 7, los subíndices varían desde 0 hasta 6, **primero** vale 2 y **último** vale 4.

Si se desea encolar el dato d incrementamos la variable **último** utilizando la operación **módulo** así:

$$\text{ultimo} = (\text{último} + 1) \% \text{n}$$

o sea, **último** = **(4 + 1) % 7**, la cual asignará a **último** el residuo de dividir 5 por 7, que es 5. El vector queda así:

0	1	2	3	4	5	6
			b	c	d	

Al encolar el dato **e**, el vector queda así:

0	1	2	3	4	5	6
			b	c	d	e

y las variables **primero** y **último** valdrán 2 y 6, respectivamente. Al encolar de nuevo, digamos el dato **f**, aplicamos el operador **%** y obtenemos el siguiente resultado:

$$\text{ultimo} = (6 + 1) \% 7, \text{ el cual es } 0, \text{ ya que el residuo de dividir } 7 \text{ por } 7 \text{ es cero.}$$

Por consiguiente, la posición del vector a la cual se llevará el dato **f** es la posición 0. El vector quedará con la siguiente conformación:

0	1	2	3	4	5	6
f			b	c	d	e

con **último** valiendo 0 y **primero** valiendo 2. De esta forma hemos podido encolar en el vector sin necesidad de mover elementos en él.



Con base en lo anterior, definimos la clase cola en Python así:

```
class cola(vector):
    def __init__(self, n):
        vector.__init__(self, n)
        self.primero = 0
        self.ultimo = 0

    def esLlena(self):
        return self.primero == self.ultimo

    def esVacia(self):
        return self.primero == self.ultimo

    def encolar(self, d):
        self.ultimo = (self.ultimo + 1) % self.n
        if self.esLlena():
            print("cola llena, no se puede encolar\n")
            self.ultimo = (self.ultimo - 1 + self.n) % self.n
            return
        self.V[ultimo] = d

    def desencolar(self):
        if self.esVacia():
            print("cola vacía, no se puede desencolar\n")
            return None
        self.primero = (self.primero + 1) % self.n
        return self.V[primero]

    def siguiente(self):
        if self.esVacia():
            print("cola vacía, no hay siguiente\n")
            return None
        aux = (self.primero + 1) % self.n
        return self.V[aux]
```

Es importante notar que la condición de cola llena y cola vacía es la misma, con la diferencia de que en el subprograma **encolar** se chequea la condición después de incrementar **ultimo**. Si la condición resulta verdadera, invoca el subprograma **colaLlena**, cuya función será dejar **ultimo** con el valor que tenía antes de incrementarla y detener el proceso, ya que no se puede encolar.



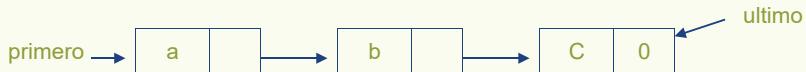
Por consiguiente, habrá una posición del vector que no se utilizará, pero que facilita el manejo de las condiciones de cola vacía y cola llena. Fíjese que en desencolar la primera instrucción es controlar la situación de cola vacía.

Un ejemplo del uso de esta clase es:

```
from claseVector import cola  
qu = cola(10)  
qu.encolar("a")  
qu.encolar("b")  
qu.encolar(316)  
qu.muestraCola()  
d = qu.desencolar()  
print("\nDato desencolado:", d)  
qu.muestraCola()
```

Representación de colas como listas ligadas.

Ahora, definamos la cola como derivada de la clase LSL:



Las operaciones sobre la cola son *Encolar* y *Desencolar*.

- **Encolar.** Consiste en insertar un registro al final de una lista ligada.
- **Desencolar.** Consiste en borrar el primer registro de una lista ligada. Habrá que controlar si la cola está o no vacía.

Nuestra clase cola en Python, derivada de la clase LSL es:

```
class cola(LSL):
    def __init__(self):
        LSL.__init__(self)

    def encolar(self, d):
        self.agregarDato(d)

    def desencolar(self):
        if self.esVacia():
            print("\nCola vacía no hay datos para desencolar")
            return None
        d = self.primer.retornarDato()
        p = self.primerNodo()
        self.borrar(p)
        return d

    def siguiente(self):
        if self.esVacia():
            print("\nCola vacía no hay siguiente")
            return None
        d = self.primer.retornarDato()
        return d
```

Un ejemplo de uso de esta clase es:

```
from nodoSimple import cola
a = cola()
b = a.esVacia()
print(b)
a.encolar("a")
a.encolar("e")
a.encolar("i")
a.encolar("o")
a.recorrerLista()
b = a.esVacia()
print("\n", b)
d = a.desencolar()
print("\ndato desencolado es: ", d)
a.recorrerLista()
d = a.siguiente()
print("\nel siguiente es:", d)
a.recorrerLista()
```

Observe bien: usted puede elaborar algoritmos con solo saber qué hacen los métodos de una clase.