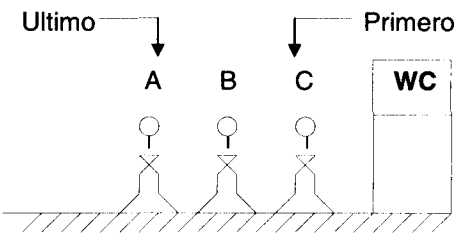
9

COLAS

9.1 DEFINICIÓN

Una cola es una lista ordenada en la cual las operaciones de inserción se efectúan en un extremo llamado **ultimo** y las operaciones de borrado se efectúan en el otro extremo llamado **primero**.



En términos prácticos es lo que supuestamente se debe hacer para tomar un bus, para comprar las boletas para entrar a un cine o para hacer uso de un servicio público.

Las operaciones sobre una cola serían entonces:

Crear() la cual crea una cola vacía.
Encolar(i, C) la cual añade un elemento i al final de la cola.
Desencolar(C) la cual remueve el primer elemento de la cola.
Primero(C) la cual da el primer elemento de la cola.
Esvacia(C) la cual es falsa o verdadera dependiendo de si la cola está vacía o no.

La especificación completa de esta estructura de datos es:

ESTRUCTURA COLA
Funciones:
Crear()→ cola
Esvacia(cola) → lógico
Encolar(elemento, cola) → cola
Desencolar(cola) → cola
Primero(cola) → elemento

Axiomas:
Para todo C ∈ colas, i ∈ elementos
Esvacia(crear) ::= Verdad
Esvacia(Encolar(i, C)) ::= Falso
Primero(Crear) ::= error
Primero(Encolar(i, C)) ::= if Esvacia(C) then i
else Primero(C)
Desencolar(Crear) ::= Crear
Desencolar(Encolar(i, C)) ::= if Esvacia(C) then Crear
else Encolar(i, Desencolar(C))

Si tenemos una cola: **a, b, c** donde **a** es el primer elemento de la cola y **c** es el último, de acuerdo a las funciones definidas la cola es así:

Encolar(**c**,Encolar(**b**,Encolar(**a**,Crear)))

Dejo al estudiante la tarea de probar si son correctos los axiomas definidos.

9.2 REPRESENTACION DE COLAS

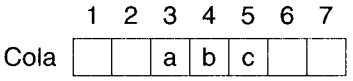
En general las colas se pueden representar en vector o como lista ligada. En la representación en vector se puede hacer de dos formas: en forma no circular y en forma circular. Veremos las tres formas de representación.

Representación de colas en un vector, en forma no circular

Para representar colas en esta forma se requiere un vector, que llamaremos **cola** y dos variables: una que llamaremos **primero**, que indique la posición del vector en la cual se halla el primer dato de la cola, y otra, que llamaremos **ultimo**, la cual indica la posición en la cual se halla el último dato de la cola.

En el vector se guardan los datos, **primero** apuntará hacia la posición anterior en la cual se halla realmente el primer dato de la cola, **ultimo** apuntará hacia la posición en la que realmente se halla el último dato de la cola.

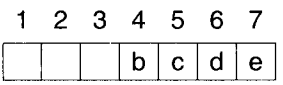
Si tenemos un vector con la siguiente configuración:



La variable **primero** valdrá 2 indicando que el primer elemento de la cola se halla en la posición 3 del vector; la variable **ultimo** valdrá 5 indicando que el último elemento de la cola se halla en la posición 5 del vector.

Las operaciones sobre la cola que son encolar y desencolar funcionan de la siguiente forma: si se desea encolar basta con incrementar la variable **ultimo** en uno y llevar a la posición **ultimo** del vector el dato a encolar; si se desea desencolar basta con incrementar en uno la variable **primero** y extraer de dicha posición el dato que allí se halla.

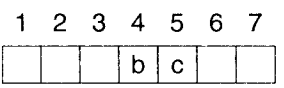
En general, si el vector **cola** definido tiene **n** elementos, cuando la variable **ultimo** sea **n**, se podría pensar que la cola está llena.



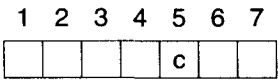
Pero, como se observa en la figura el vector tiene espacio al principio, por consiguiente podemos pensar e mover los datos hacia el extremo izquierdo, actualizar **primero** y **ultimo** para luego encolar.

En otras palabras, la condición de cola llena será cuando **primero** sea igual a cero y **ultimo** sea igual a **n**.

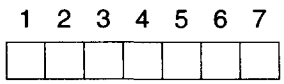
Consideremos ahora operaciones sucesivas de desencole para el ejemplo dado: después de ejecutar la primera operación de desencole los valores de las variables **ultimo** y **primero** y del vector **cola** será la siguiente:



donde la variable **primero** valdrá 3 y la variable **ultimo** 5.
Al desencolar nuevamente, la configuración del vector será:



donde la variable **primero** vale 4 y la variable **ultimo** 5.
Si desencolamos de nuevo, la configuración del vector será:



y la variable **primero** vale 5 y la variable **ultimo** 5.

En otras palabras **primero** es igual a **ultimo** y la cola está vacía, o sea, la condición de cola vacía será **primero = ultimo.**

Teniendo definidas las condiciones de cola llena y cola vacía podemos proceder a escribir los algoritmos para encolar y desencolar representando la cola en un vector de **N** elementos.

```
sub_programa encolar(cola, n, primero, ultimo, d)
      if  ultimo = n  then
            if primero = 0  then
                  cola_llena
            else
            for i = primero + 1  to  n  do
                  cola(i - primero) = cola(i)
            end(for)
            ultimo = ultimo - primero
            primero = 0
      end(if)
      ultimo = ultimo + 1
      cola(ultimo) = d
fin(encolar)
sub_programa desencolar(cola, primero, ultimo, d)
      if  primero = ultimo  then
            cola_vacia
      end(if)
      primero = primero + 1
      d = cola(primero)
fin(desencolar)
```

En el subprograma para encolar los parámetros **cola, n, primero, ultimo** y **d** son: el vector, el número de elementos de éste, la posición del primer elemento de la cola, la posición del último elemento de la cola y el dato a encolar, respectivamente.

En el subprograma desencolar **cola** es el vector, **primero** y **ultimo** son las variables para identificar las posiciones donde se hallan el primero y el último elemento de la cola en el vector, tal como se definió anteriormente y **d** es la variable donde regresa el dato desencolado.

si consideramos una situación como la siguiente:

1	2	3	4	5	6	7
	e	f	b	c	d	g

n vale 7, **ultimo** vale 7 y **primero** vale 1.

Si se desea encolar el dato **h** y aplicamos el subprograma ENCOLAR definido, la acción que efectúa dicho subprograma es mover los elementos desde la posición 2 hasta la 7, una posición hacia la izquierda de tal forma que quede espacio en el vector para incluir la **h** en la cola. El vector quedará así:

1	2	3	4	5	6	7
e	f	b	c	d	g	

con **primero** valiendo 0 y **ultimo** valiendo 6. De esta forma continúa la ejecución del subprograma **encolar** y se incluye el dato **h** en la posición 7 del vector. El vector queda así:

1	2	3	4	5	6	7
e	f	b	c	d	g	h

Con **primero** valiendo 0 y **ultimo** valiendo 7.
Si en este momento se desencola el vector queda así:

1	2	3	4	5	6	7
	f	b	c	d	g	h

con **primero** valiendo 1 y **ultimo** valiendo 7.

Si la situación anterior se repite sucesivas veces, el cual sería el peor de los casos, cada vez que se vaya a encolar se tendrían que mover **n-1** elementos en el vector, lo cual haría ineficiente el manejo de la cola, ya que el proceso de encolar tendría orden de magnitud **O(n)**.

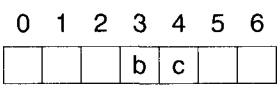
Para obviar este problema y poder efectuar los subprogramas de ENCOLAR y DESENCOLAR en un tiempo O(1) manejaremos el vector circularmente.

Representación de colas circularmente en un 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, y el incremento de las variables **primero** y **ultimo** se hace utilizando la función u operación **módulo** de la siguiente manera:

```
primero = (primero + 1) % n
ultimo = (ultimo + 1) % n
```

Dicha operación retorna el residuo de una división entera. Si se tiene una cola en un vector, con la siguiente configuración:

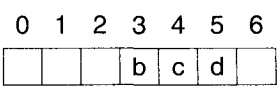


n vale 7, los subíndices varían desde 0 hasta 6, **primero** vale 2 y **ultimo** vale 4.

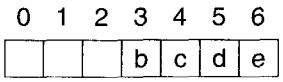
Si se desea encolar el dato **d** incrementamos la variable **ultimo** utilizando la operación **módulo** así:

ultimo = (ultimo + 1) % n

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



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

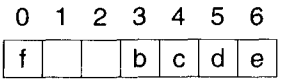


y las variables **primero** y **ultimo** valdrán 2 y 6 respectivamente.

Al encolar de nuevo, digamos el dato **f,** aplicamos el operador % obteniendo el siguiente resultado:

ultimo = (6+1) % 7, el cual es 0, ya que el residuo de dividir 7 por 7 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:



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

Los subprogramas para encolar y desencolar quedan así:

```
sub_programa encolar(cola, n, primero, ultimo, d)
      ultimo = (ultimo+1) mod n
      if  ultimo = primero  then
            cola_llena
      end(if)
      cola(ultimo) = d
fin(encolar)


sub_programa desencolar(cola, n, primero, ultimo, d)
      if primero = ultimo then
            cola_vacia
      end(if)
      primero = (primero+1) mod n
      d = cola(primero)
fin(desencolar)
```

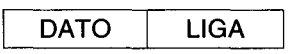
Es importante notar que la condición de cola llena y cola vacía es la misma, con la diferencia de que en el subprograma **encolar** se chequea la condición después de incrementar **ultimo**. Si la condición resulta verdadera invoca el

subprograma COLA_LLENA cuya función será dejar **ultimo** con el valor que tenía antes de incrementarla y detener el proceso, ya que no se puede encolar.

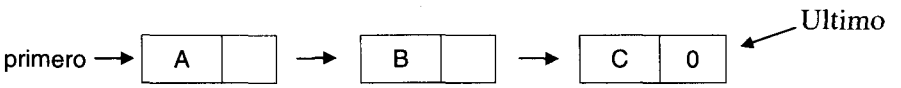
Por consiguiente habrá una posición del vector que no se utilizará, pero que facilita el manejo de las condiciones de cola vacía y cola llena.

Representación de colas como listas ligadas

Siempre que se desee representar algún objeto como lista ligada, lo primero que debe hacerse es definir la configuración del registro.

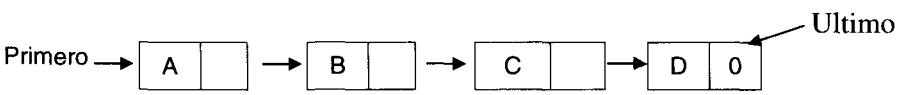


La condición de cola vacía es **primero = 0**.



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

Encolar: Consiste en insertar un registro al final de una lista ligada.



Si la cola es vacía el dato a encolar será simultáneamente el primero y el último.

```
sub_programa encolar(primero, ultimo, d)
        new(x)
        dato(x) = d
        liga(x) = 0
        if  primero = 0  then
                primero = x
        else
                liga(último) = x
        end(if)
        último = x
fin(encolar)
```

Desencolar: Consiste en borrar el primer registro de una lista ligada (exacto a Desapilar). Habrá que controlar si la cola está o no vacía.

```
sub_programa desencolar(primero, ultimo, d)
      if primero = 0 then
            cola-vacia
      end(if)
      d = dato(primero)
      x = primero
       primero = liga(primero)
      devolver_registro(x)
fin(desencolar)
```

9.3. MANEJO DE VARIAS PILAS Y COLAS

Manejo de dos pilas en un vector

Dos pilas las podremos manejar en un solo vector de la siguiente forma: Si **n** es el número de elementos del vector, dividimos inicialmente el vector en dos partes iguales. Llamemos **m** la variable que apunta hacia el elemento de la mitad.

La primera mitad del vector será para manejar la pila 1, y la segunda mitad del vector para manejar la pila 2.

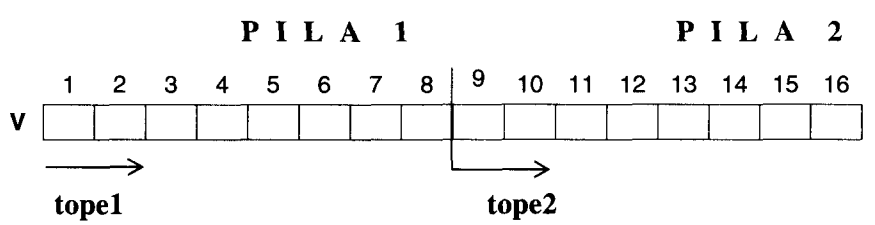
Cada pila requiere una variable tope para conocer en qué posición está el último dato de la pila. Llamemos estas variables **tope1** y **tope2** para manejar las pilas 1 y 2 respectivamente.

Consideremos el siguiente ejemplo:

Sea **V** el vector en el cual manejaremos las dos pilas.

El valor de **N** es 16.

Inicialmente el valor de **m** es 8. La mitad de **n**.



La variable **tope1** variará desde 1 hasta **m**.
La variable **tope2** variará desde **m+1** hasta **n**.
La pila 1 estará vacía cuando **tope1** sea igual a cero.
La pila 1 estará llena cuando **tope2** sea igual a **m**.
La pila 2 estará vacía cuando **tope2** sea igual a **m**.
la pila 2 estará llena cuando **tope2** sea igual a **n**.

Teniendo definido el diseño, lo que sigue es elaborar los algoritmos para manipular las pilas con él.

Consideremos primero un algoritmo para apilar un dato en alguna de las dos pilas. Habrá que especificar en cuál pila es que se desea apilar. Para ello utilizaremos un suiche (sw), el cual enviamos como parámetro del subprograma. Si el suiche es igual a 1 hay que apilar en la pila 1, si el suiche es igual a 2, hay que apilar en la pila 2.

El algoritmo es el siguiente:

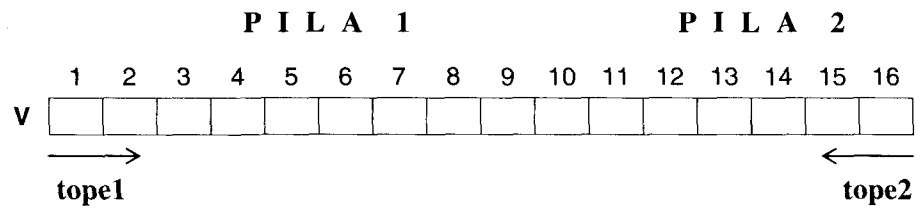
```
sub_programa apilar(v, m, n, tope1, tope2, sw, d)
      if sw = 1 then
            if tope1 = m then
                  pila_llena(sw)
            end(if)
            tope1 = tope1 + 1
            v(tope) = d
      else
            if tope2 = n then
                  pila_llena(sw)
            end(if)
            tope2 = tope2 + 1
            v(tope2) = d
      end(if)
fin(apilar)
```

El subprograma PILA_LLENA recibe como parámetro el valor del suiche.

La tarea de PILA_LLENA es: si es la pila 1 la que está llena buscará si la pila 2 está llena, en caso de no estarlo moverá los datos de la pila 2 una posición hacia la derecha, actualizará **m** y regresa al subprograma de apilar para apilar en la pila 1; si es la pila 2 la que está llena buscará si hay espacio en la pila 1, en cuyo caso moverá los datos de la pila 2 una posición hacia la izquierda, actualizará **m** y regresa a apilar en la pila 2.

Como se podrá observar la operación de apilar implica mover datos en el vector, debido a que una pila podrá crecer más rápido que la otra.

Una mejor alternativa de diseño es la siguiente:



La pila 1 se llenará de izquierda a derecha y la pila 2 de derecha a izquierda.

En este segundo diseño no hay que preocuparse por cuál pila crezca más rápidamente. Las condiciones a controlar son:

La pila 1 estará vacía cuando **tope1** sea igual a cero.
La pila 2 estará vacía cuando **tope2** sea igual a **n+1**.
Las pilas estarán llenas cuando **tope1 + 1** sea igual a **tope2**.
el algoritmo para este segundo diseño es:

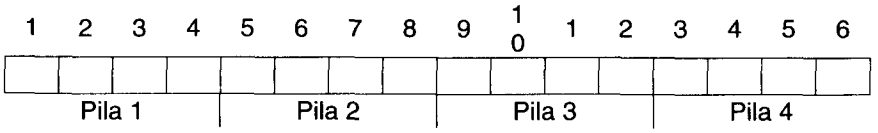
```
sub_programa apilar(v, sw, n, m, tope1, tope2, d)
      if  tope1 + 1 = tope2  then
            pila_llena
      end(if)
      if  sw = 1  then
            tope1 = tope1 + 1
            v(tope) = d
      else
            tope2 = tope2 - 1
            v(tope2) = d
      end(if)
fin(apilar)
```

Manejo de N pilas en un vector de M elementos

Ocurre muchas veces la necesidad de manejar más de dos pilas. Para la representación definida, si se desean manejar **n** pilas, una forma podría ser definir **n** vectores y **n** variables **tope**, una por cada pila que se desee manejar, sin embargo, esta forma no es la más aconsejable, ya que es impráctico e ineficiente escribir código para controlar **n** vectores y **n** variables, sin contar conque los programas

desarrollados serían particulares para el número de pilas que se deseen manejar, y que debido a que las pilas no se llenan a la misma velocidad ni uniformemente, el hecho de que una pila se llene detendría el proceso, sabiendo que en las otras pilas hay espacio para almacenar más datos.

Veremos un método para manejar **n** pilas en un vector de **m** elementos. Inicialmente se dividirá el vector en partes iguales, en tanto sea posible; serán tantas partes como pilas se deseen manejar. Consideremos el siguiente vector, que llamaremos **V**:



En nuestro ejemplo **m** vale 16 y **n** vale 4, es decir, vamos a representar 4 pilas en un vector de 16 elementos.

Necesitamos asignarle a cada pila una parte del vector, en este caso a cada pila le corresponderán 4 posiciones: la pila 1 desde la posición 1 hasta la 4, la pila 2 desde la posición 5 hasta la 8, la pila 3 desde la posición 9 hasta la 12 y la pila 4 desde la posición 13 hasta la 16.

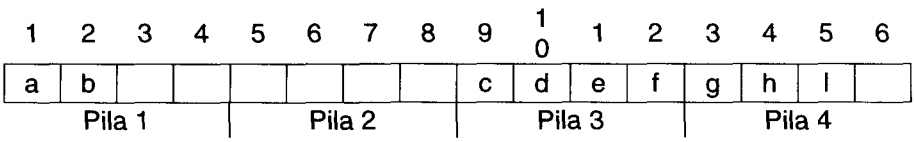
Necesitamos conocer en qué posición del vector comienza cada pila, para ello utilizaremos un vector que llamaremos **bases**, el cual en la posición **i** contendrá la posición anterior, en el vector **V**, a la que en realidad comienza la pila **i**. Más técnicamente, **bases(i)** permite conocer la posición en la cual comienza la pila **i** en el vector **V**.

Necesitamos conocer la posición del vector en la cual se halla el último elemento de cada pila, para ello utilizaremos un vector que llamaremos **topes.**

topes(i) dará la posición, en el vector **V**, en la cual se halla el último elemento de la pila **i**. Si vamos a manejar **n** pilas, entonces el vector de **topes** tendrá **n** elementos.

Inicialmente los vectores de **bases** y **topes** tendrán los mismos datos indicando que cada pila se halla vacía. Es decir, la condición de que la pila **i** se halla vacía es: **topes(i) = bases(i)**.

Si tenemos el vector **V** con la siguiente configuración:



Los vectores de **bases y topes** tendrán la siguiente información:

	1	2	3	4	5
BASES	0	4	8	12	16

	1	2	3	4
TOPES	2	4	12	15

La pila 3 se encuentra llena. Observe que **topes(3)** es igual a **bases(4)**. La condición de que la pila **i** se encuentra llena es: **topes(i) = bases(i+1).** Es por esta razón por la que el vector de **bases** tiene un elemento más que el vector de **topes**: se facilita el control de pila llena de la pila **n**.

Teniendo definida la representación y las condiciones de pila vacía y pila llena desarrollemos los algoritmos para dar las condiciones iniciales y para apilar y desapilar de cualquier pila.

Las condiciones iniciales se definen conocidos **m** y **n**, mediante las siguientes instrucciones:

```
p = m / n
for i = 1 to n do
      topes(i) = (i-1)*p
      bases(i) = (i-1)*p
end(for)
bases(n+1) = m
```

El subprograma para apilar en la pila **i** es:

```
sub_programa apilar(v, topes, bases, i, d)
      if topes(i) = bases(i+1) then
            pila_llena(i)
      end(if)
      topes(i) = topes(i) + 1
      v(topes(i)) = d
fin(apilar)
```

Donde **v** es el vector, **i** la pila en la cual se desea apilar y **d** el dato a apilar.
El subprograma para desapilar de la pila **i** es:

```
sub_programa desapilar(v,topes,bases,i,d)
      if topes(i) = bases(i) then
            pila_vacia(i)
      end(if)
      d = v(topes(i))
      topes(i) = topes(i) - 1
fin(desapilar)
```

Donde **v** es el vector, **i** la pila de la cual se va a desapilar y **d** el dato desapilado.

En el subprograma para apilar se chequea la condición de pila llena, la cual si resulta verdadera, ocasiona la ejecución del subprograma PILA_LLENA, enviando como parámetro la variable **i**, indicando que fue esta la pila que se llenó.

El subprograma PILA_LLENA buscará en las otras pilas para ver si hay espacio disponible, en cuyo caso hará los movimientos apropiados en el vector **V**, y las actualizaciones apropiadas en los vectores de **bases** y **topes** para poder apilar el dato requerido en la pila **i**.

La táctica que sigue dicho subprograma es: primero busca en las pilas a la derecha de la pila **i** (desde la pila **i+1** hasta la pila **n**), de no hallar espacio en ninguna de estas pilas buscará en las pilas a la izquierda de la pila **i** (desde la pila **i-1** hasta la pila 1), si aún no encuentra espacio, significa que todas las pilas están llenas y la operación de apilar no se puede efectuar, por tanto detendrá el proceso emitiendo un mensaje apropiado.

Veamos el algoritmo PILA_LLENA

```
sub_programa pila_llena(i)
      j = i + 1                                  // busca a la derecha de la pila i
      while j <= n and topes(j) = bases(j+1) do
            j = j + 1
      end(while)
      if j <= n then                             // encontró espacio en la pila j
            k = topes(j)
            while k > topes(i) do                // mueve los elementos
                  v(k+1) = v(k)                  //  en el vector v
                  k = k - 1
            end(while)
```

```
            for  k = i+1 to j  do                          // actualiza los
                  topes(k) = topes(k) + 1                  // vectores de topes
                  bases(k) = bases(k) + 1                  // y bases
            end(for)
            return                                  // retorna al subprograma de apilar
      end(if)
      j = i -1                                      // busca a la izquierda de la pila i
      while  j > 0  and  topes(j) = bases(j+1)  do
            j = j - 1
      end(while)
      if  j > 0  then                                 // encontró espacio en la pila j
            for  k = bases(j+1) to topes(i) - 1  do
                  v(k) = v(k+1)                       // mueve los elementos del vector v
            end(for)
            for  k = j+1 to i  do                       // actualiza los vectores
                  topes(k) = topes(k) - 1               // de topes y bases
                  bases(k) = bases(k) - 1
            end(for)
            return                                    // retorna al subprograma de apilar
      end(if)
      write('todas las pilas están llenas')
      stop
fin(pila_llena)
```

Note que el subprograma PILA_LLENA no efectúa la operación de apilar, él sólamente abre espacio y retorna al subprograma APILAR para que éste haga dicha operación.

Manejo de n pilas como listas ligadas

Con **n** pilas deberán tenerse **n** listas (una por cada pila). Para no tener que manejar **n** variables **tope**, utilizaremos un vector de apuntadores que llamaremos **topes. topes(i)** apunta hacia el primer registro de la lista que representa la pila **i**. Ver figura 9.1.

Los subprogramas para apilar y desapilar son similares a los presentado anteriormente para una pila. Basta con reemplazar **tope** por **topes(i)**.

```
sub_programa apilar(topes, i, d)
      new(x)
      dato(x) = d
      liga(x) = topes(i)
```

```
        topes(i) = x
fin(apilar)

sub_programa desapilar(topes, i, d)
        if topes(i) = 0 then
                pila_vacía(i)
        end(if)
        d = dato(topes(i))
        x = topes(i)
        topes(i) = liga(topes(i))
        devolver_registro (x)
fin(desapilar)
```

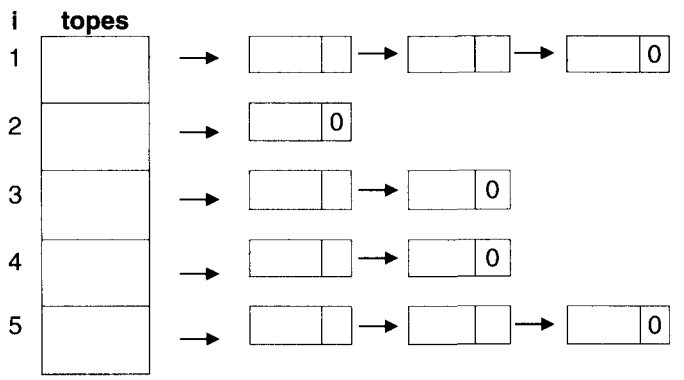
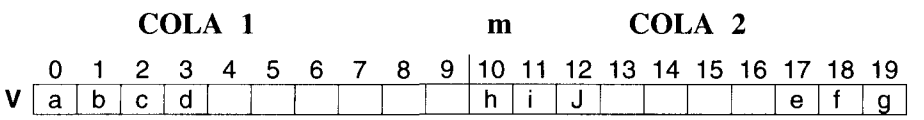


Figura 9.1

Estos subprogramas son mucho más eficientes que manejando la pila en un vector. La principal diferencia estriba en que en el subprograma para apilar se evita el proceso de pila_llena.

Manejo de dos colas en un vector de n elementos



Veamos ahora cómo trabajar dos colas, cada una circularmente, en un vector de **n** elementos. En nuestro ejemplo **n = 20**.

Inicialmente, a cada cola le corresponderá una mitad del vector. La cola 1 se representará desde la posición cero hasta la posición **m – 1**, y la cola 2 se representará desde la posición **m** hasta la posición **n – 1**.

Llamemos **p1** y **u1** las variables para identificar las posiciones en las cuales se hallan el primero y el último de la cola 1, y **p2** y **u2** las variables para identificar las posiciones en las cuales se hallan el primero y el último de la cola 2.

Las operaciones para encolar y desencolar en la cola 1 son idénticas al manejo circular de una cola en un vector:

```
p1 = (p1 + 1) % m
u1 = (u1 + 1) % m
```

Para encolar y desencolar en la cola 2, y que ésta se comporte circularmente, debemos tener en cuenta que la porción de vector que le corresponde va desde **m** hasta **n – 1**.

El número de elementos en ella es **m – n**, y para que su comportamiento sea circular debemos, temporalmente, convertir los valores de **p2** y **u2** a la forma desde 0 hasta **m – n**.

Para lograr esto, basta con restarle **m** a los valores de **p2** y **u2** antes de aplicar la operación módulo y después incrementar el resultado en **m** para que el valor obtenido quede en el rango correcto: desde **m** hasta **n – 1**.

Las instrucciones para incrementar **p2** y **u2** son:

```
p2 = (p2 – m + 1) % (n – m) + m
u2 = (u2 – m + 1) % (n – m) + m
```

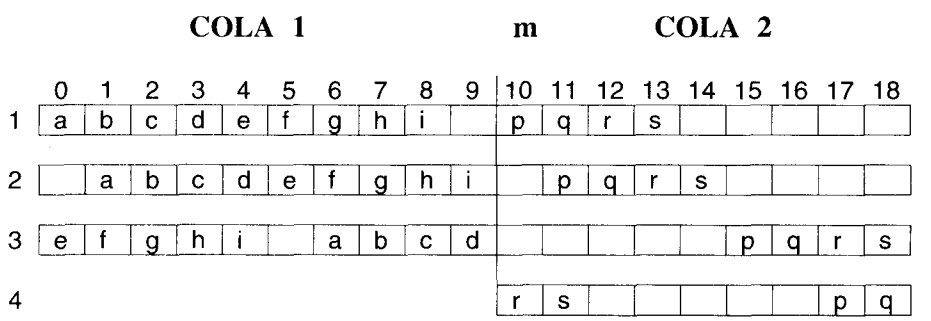
Teniendo definido la forma de incremento de las variables de manejo de cada cola, nuestros algoritmos para encolar y desencolar en cada una de ellas serán:

```
sub_programa encolar(V, m, n, p1, u1, p2, u2, cola, d)
    if cola = 1 then
        u1 = (u1 + 1) % m
        if u1 = p1 then
            cola_llena(cola)
        end(if)
        V(u1) = d
    else
        u2 = (u2 – m + 1) % (m – n) + m
```

```
            if  u2 = p2  then
                  cola_llena(cola)
            end(if)
            V(u2) = d
      end(if)
fin(encolar)

sub_programa desencolar(V, m, n, p1, u1, p2, u2, cola, d)
      if  cola = 1  then
            p1 = (p1 + 1) % m
            if  u1 = p1  then
                  write('cola 1 vacía')
            else
                  d = V(p1)
            end(if)
      else
            p2 = (p2 – m + 1) % (m – n) + m
            if  u2 = p2  then
                  write('cola 2 vacía')
            else
                  d = V(p2)
            end(if)
      end(if)
fin(desencolar)
```

Consideremos el algoritmo cola_llena



El subprograma cola_llena se invoca cuando se desee encolar en una de las dos colas y ésta está llena. En general, el subprograma cola_llena deberá averiguar si hay espacio disponible en la otra cola y efectuar los movimientos apropiados para abrir espacio en la cola que se llenó y poder encolar.

Comencemos considerando la situación en la cual es la cola 1 la que está llena, la cola 2 no está llena y necesitamos abrir espacio para poder encolar en la cola 1.

Para lograr este objetivo debemos hacer algún tratamiento tanto a la cola 1 como a la cola 2, dependiendo de las situaciones de cola llena y de espacio disponible respectivamente

Si la **cola 1** está llena se pueden presentar tres formas diferentes, las cuales llamaremos 1, 2 y 3. En cualquier situación tenemos **p1 = u1**.

En la situación 1, la cual identificamos porque **p1** y **u1** son iguales a **m – 1**, la acción a tomar es simplemente incrementar **p1** en 1.

En la situación 2, la cual identificamos porque **p1** y **u1** son iguales a cero, la acción a tomar es simplemente hacer **u1** igual a **m**.

En la situación 3, la cual identificamos porque **p1** y **u1** están entre **1** y **m – 2**, hay que mover los datos desde la posición **p1 + 1** hasta la posición **m – 1** una posición hacia la derecha y actualizar **p1**.

Las instrucciones correspondientes a estas situaciones son:

```
casos
    :p1 = m1:                              // situación 1
          p1 = p1 + 1
    :u1 = 0:                               // situación 2
          u1 = m
    :else:                                 // situación 3
          for i = p1+1 to m – 1 do
              v(i – 1) = v(i)
          end(for)
fin(casos)
```

Las situaciones de espacio disponible en la cola 2 las llamaremos 1, 2, 3, y 4.

En situaciones 1 y 4, las cuales identificamos porque **p2** es mayor que **u2**, hay que mover los datos desde la posición **m** hasta la posición **u2** una posición hacia la derecha y actualizar **u2**.

La situación 2, la cual identificamos porque **p2** es igual a **m** basta con asignar a **p2** el valor de **n – 1**. Pero antes de hacer esto debemos considerar el hecho de que de pronto la cola 2 está vacía y entonces habrá que asignar **n – 1** tanto a **p2** como a **u2**.

En la situación 3 no hay que efectuar operación alguna.

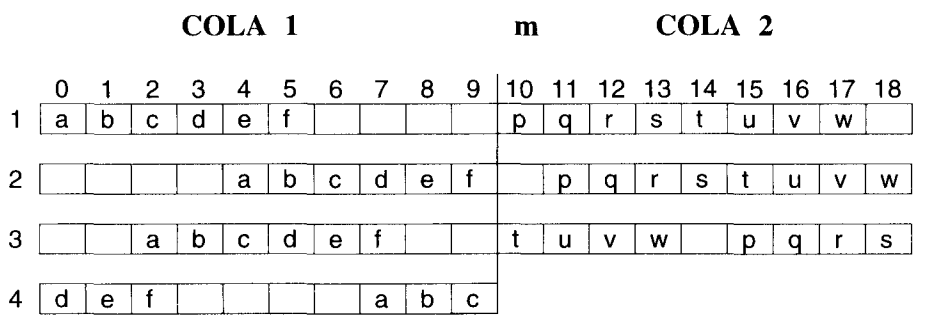
Las instrucciones correspondientes son:

```
casos
      :p2 > u2:
            for i = u2 downto m do
                V(i + 1) = V(i)
            end(for)
            u2 = u2 + 1
      :p2 = m:
            if p2 = u2 then
                u2 = n - 1
            end(if)
            p2 = n - 1
fin(casos)
```

Cualquiera que hubieran sido los tratamientos que se hayan hecho sobre ambas colas el valor de **m** hay que incrementarlo en 1.

Analicemos ahora el caso en que hubiera sido la cola 2 la que está llena y la cola 1 tiene espacio.

Consideremos las diferentes situaciones para este caso.



Las situaciones de la cola 2 llena son 3, las cuales identificaremos como 1, 2 y 3. En cualquier situación **p2 = u2**.

La situación 1, la cual identificamos porque **p2 = n – 1**, basta con hacer **p2 = m – 1**

La situación 2, la cual identificamos porque **u2 = m**, basta con hacer **u2 = m –1**.

La situación 3 es cuando no se cumple ninguna de las dos condiciones anteriores y será necesario mover los datos en el vector desde la posición **m** hasta la posición **u2** una posición hacia la izquierda.

Las instrucciones correspondientes son:

```
casos
      :p2 = n – 1:
            p2 = m – 1
      :u2 = m:
            u2 = m – 1
      :else:
            for i = m to u2 do
                V(i – 1) = V(i)
            end(for)
fin(casos)
```

Consideremos ahora el tratamiento a la cola 1. Hay 4 situaciones de cola 1 con espacio:

Situaciones 1 y 4, las cuales identificamos porque **p1 > u1** se solucionan así: si **p1** es igual a **m – 1** basta con restar 1 a **p1**, de lo contrario hay que mover los datos en el vector, desde la posición **p1** hasta la posición **m – 1**, una posición hacia la izquierda y restarle 1 a **p1**.

Situaciones 2 y 3, las cuales identificamos porque **p1 < u1** se solucionan así: si **u1** es menor que **m – 1** no hay que tomar ninguna acción, de lo contrario hay que mover todos los datos de la cola uno una posición hacia la izquierda y restar uno a **p1** y a **u1**.

Cualquiera que hubieran sido los tratamientos a las dos colas hay que restar 1 a **m**.

Las instrucciones correspondientes son;

```
casos
    :p1 > u1:
          if p1 < m – 1 then
              for i = p1 to m – 2 do
                  V(i) = V(i+1)
              end(for)
          end(if)
```

```
            p1 = p1 – 1
    :p1 < u1:
            if  u1 = m – 1   then
                    for  i = p1  to  m – 2   do
                            V(i) = V(i+1)
                    end(for)
                    p1 = p1 – 1
                    u1 = u1 – 1
            end(if)
fin(casos)
```

Agrupando todas las situaciones planteadas nuestro algoritmo de cola_llena queda así:

```
Subprograma cola_llena(cola)
    if  cola = 1  then
      if  ((u2 – m + 1) % (m – n) + m) <> p2  then   // hay espacio en cola 2
                                                     // tratamiento cola 1
          casos
              :p1 = m1:
                    p1 = p1 + 1
              :u1 = 0:
                    u1 = m
              :else:
                    for  i = p1+1  to  m – 1   do
                        V(i – 1) = V(i)
                    end(for)
          fin(casos)
                                                       // tratamiento cola 2
          casos
              :p2 > u2:
                     for  i = u2  downto  m   do
                         V(i + 1) = V(i)
                    end(for)
                    u2 = u2 + 1
              :p2 = m:
                    if  p2 = u2  then
                        u2 = n – 1
                    end(if)
                    p2 = n – 1
          fin(casos)
          m = m + 1
          return
```

```
      end(if)
    else
      if (u2 + 1) % m <> p2  then              // hay espacio en cola 1
        casos                                  // tratamiento cola 2
          :p2 = n – 1:
                p2 = m – 1
          :u2 = m:
                u2 = m – 1
          :else:
                for i = m to u2 do
                   V(i – 1) = V(i)
                end(for)
        fin(casos)
        casos
          :p1 > u1:
                if p1 < m – 1 then
                  for i = p1 to m – 2 do
                     V(i) = V(i+1)
                  end(for)
                end(if)
                p1 = p1 – 1
          :p1 < u1:
                if u1 = m – 1 then
                   for i = p1 to m – 2 do
                      V(i) = V(i+1)
                   end(for)
                   p1 = p1 – 1
                   u1 = u1 – 1
                end(if)
        fin(casos)
        m = m – 1
        return
    end(if)
    write('ambas colas están llenas')
    stop
fin(cola_llena)
```

Manejo de N colas en un vector

De la misma forma que puede suceder que haya que manejar **n** pilas, también puede suceder que haya que manejar **n** colas. De una forma similar podemos manejar **n** colas en un vector de **m** elementos. En el manejo de colas necesitaríamos tres vectores auxiliares: uno que indique en cuál posición del vector empieza cada

cola, otro que indique en cuál posición del vector se halla el primer elemento de cada cola y otro que indique en cuál posición del vector se halla el último elemento de cada cola.

Manejo de N colas como listas ligadas

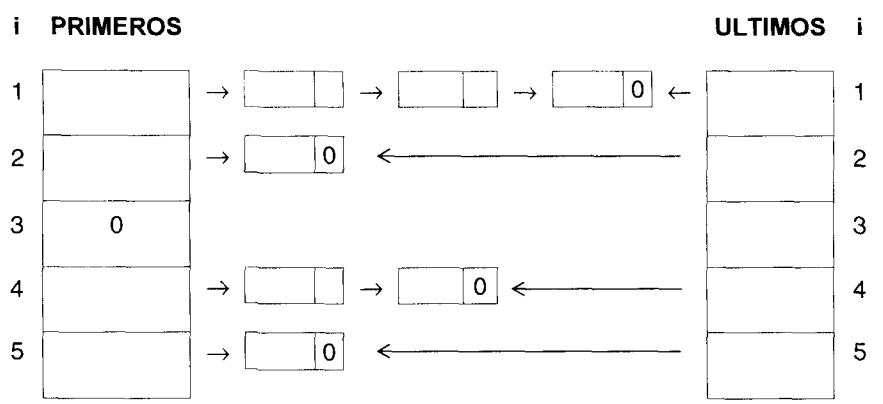
Equivale a manejar **n** listas, una lista por cada cola.

Para manejar **n** colas como listas ligadas simplemente manejamos dos vectores, los cuales llamaremos primeros y ultimos.

Primeros(i): Apunta hacia el primer registro de la lista ligada que representa la cola **i**.

Ultimos(i): Apunta hacia el último registro de la lista ligada que representa la cola **i**.

Los subprogramas para ENCOLAR Y DESENCOLAR son los mismos que manejando una sola cola, con la inclusión del parámetro **i**, el cual indica en cuál cola voy a encolar (o desencolar), y cambiando PRIMERO por PRIMEROS(i) y ULTIMOS por ULTIMOS(i).



Representación gráfica cada una como lista ligada del manejo de N colas.