CAPITULO 7
OTRAS ESTRUCTURAS DINÁMICAS Y ORDEN DE ALGORITMOS

7.1 Lista Circular

La figura 7.1. presenta un la estructura de la lista circular

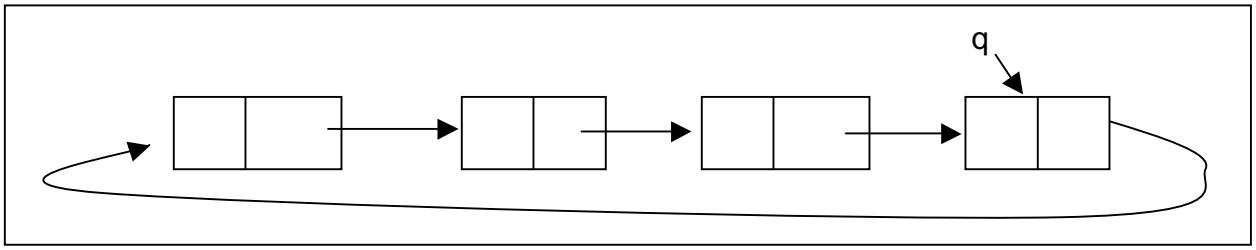

Figura 7.1. Estructura de lista circular

Sus características son:

- Podemos llegar a los nodos que preceden.
- El campo “siguiente” del último nodo contiene un apuntador al primer nodo y no un apuntador nulo (NULL).
- Si recorremos toda la lista llegamos al punto inicial.
- El apuntador externo q apunta al último nodo. Por lo que para obtener el primer nodo se accesa el siguiente a q.
- Un apuntador externo nulo lista circular varia.

Veamos un ejemplo del uso de esta estructura circular en la implementación de una cola.

Si tenemos una cola de dos elementos que queremos se implemente con una cola circular, el segundo (último) elemento de la cola será apuntado por el apuntador externo (q) y el primer elemento por el siguiente a q. esto hace que no sea necesario tener un encabezado con dos apuntadores al primero y último elemento de la cola. Observe la figura

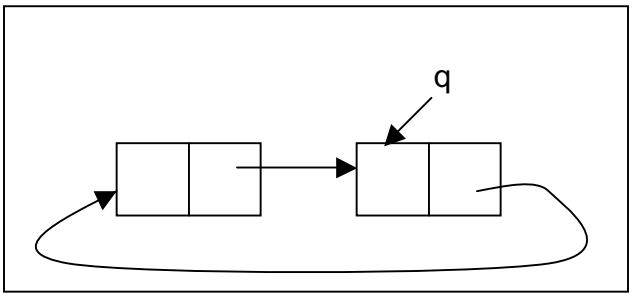

Figura 7.2. Cola de dos elementos

Ahora veamos como cambia la estructura de datos.

```
typedef struct s_nodo
{
   ELEM info;
   struct s_nodo*sig;
} NODO;

typedef NODO *COLA;
```

Veamos como cambian las dos operaciones principales de cola: insert y remove.

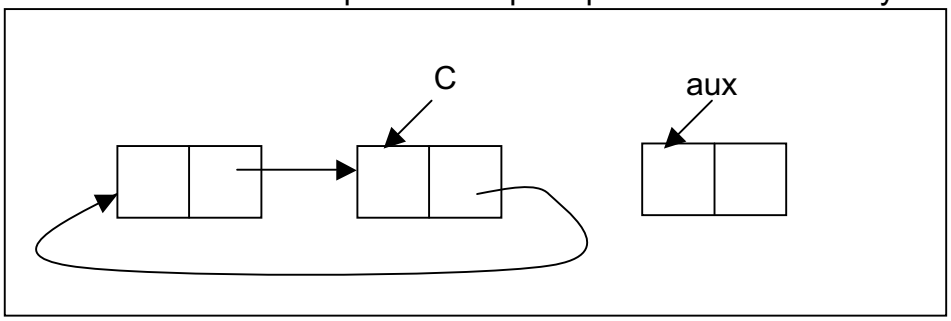

Figura 7.3. Insertando un elemento en la cola circular

Note que en la inserción el siguiente al último de la cola (apuntado por C) pasa a ser aux y luego debe actualizarse el siguiente de aux para que apunte al primero de la cola.

```
COLA P_Insert(COLA C,ELEM e)
 /* PRE: dada una cola creada y un elemento
    POST: coloca el elemento como siguiente al último
 */
{
   NODO *aux;

   if ((aux=(NODO *)malloc(sizeof(NODO))) != NULL)
   {
      aux->info = e;
      aux->sig = C->sig;
      C->sig = aux;
      C = aux;
   }
   return(C);
}
```

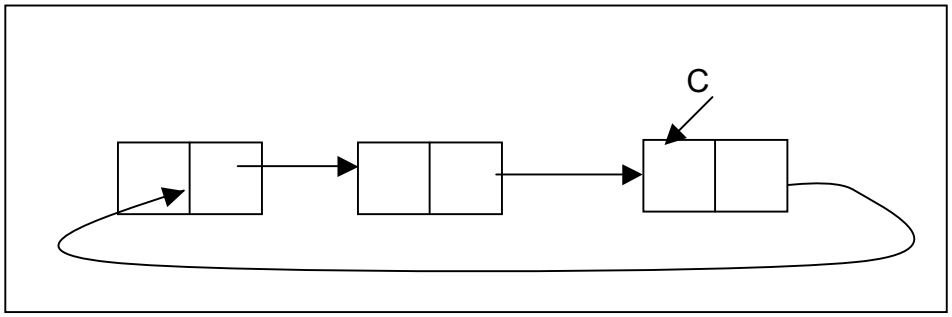

Figura 7.4. Resultado del insert

Si queremos eliminar el primer elemento de la cola que aparece en la figura 7.4 la operación de Remove quedaría.

```
COLA C_Remove(COLA C)
 /* PRE: dada una una cola creada no vacía
    POST: remueve el primer elemento de la cola
 */
 {
    NODO *aux;

    aux = C->sig;
    C->sig = aux->sig;
    free(aux);
    return(C);
 }
```

7.2 Listas Doblemente enlazadas.

Se conoce listas doblemente enlazadas a la estructura dinámica que contiene dos apuntadores: uno hacia el siguiente nodo y el otro hacia el nodo anterior. observe lo que ocurre en la figura 7.5.

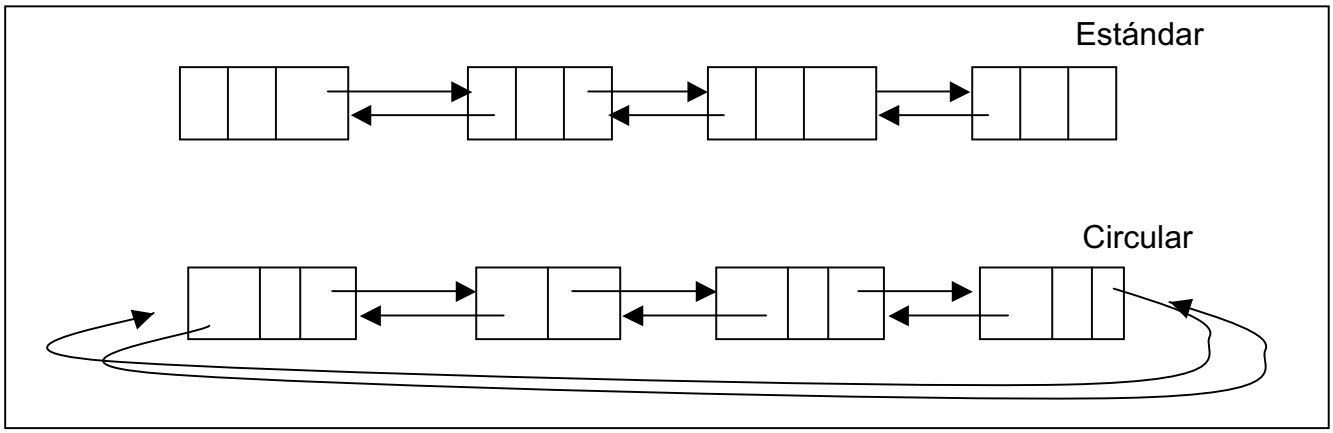

Figura 6.5. Estructura de listas doblemente enlazadas

Sus características son:

- Permite recorrer una lista en dos sentidos: hacia atrás y hacia adelante.
- Cada nodo contiene dos apuntadores uno al predecesor y otro al sucesor.

Veamos como cambia la estructura del nodo.

```
typedef struct s_nodo
{
   ELEM info;
   struct s_nodo *ant, *sig;
} NODO;
```

7.3 METRICA Y COMPLEJIDAD DE ALGORITMOS

Muchas veces tenemos diferentes algoritmos que solucionan un mismo problema. La idea es buscar un mecanismo que permita compararlos para determinar el mejor.

También es importante tener alguna manera de determinar la eficiencia de un algoritmo. Hay diferentes herramientas una de ellas es el orden O grande.

7.3.1 Orden O grande

El orden O grande es una medida que aproxima el comportamiento de un algoritmo a una función continua. La idea es comparar la cola de la función con la cantidad de cálculos realizados por el algoritmo en términos del volumen de datos.

No quiere decir que efectivamente o exactamente que es esa la función, pero si que se acerca mucho. Es decir, la función viene a ser una cota superior "cercana" al algoritmo. En la medida que crece el volumen de datos el comportamiento del algoritmo se acerca más a dicha función.

Se dice que esa función (f(n)) es el orden del algoritmo y se denota O(f(n)). Para ello se dispone de una escala de comparación que permite decidir cuál función es mejor que otra. Entonces tratamos de buscar la función, dentro de esa escala de comparación, que más se parezca a mi algoritmo.

1	$\propto$	$\log n$	$\propto$	n	$\propto$	$n\log n$	$\propto$	n^k	$\propto$	e^n
orden ctte		orden logarítmico		orden lineal				orden polinómico		orden exponencial

El símbolo $\propto$ representa que la función es menor a la siguiente en "la cola " es decir para un tamaño suficientemente grande de n. La intención es buscar algoritmos cuyo orden sea menor que polinómico.

7.3.2 Ejemplos

- Búsqueda Secuencial, se basa en buscar un elemento desde el inicio hasta el final del arreglo.
  Mejor caso: encontrar en la primera un solo calculo O (1)
  Peor caso: Encontrarlo en el último todos los cálculos O (n)
  Caso Promedio: O (n/2) que es el mismo O (n)

  De esta forma concluimos que el orden de la búsqueda secuencia es O(n).

- Búsqueda Binaria, se basa en partir el arreglo tantas mitades como sea necesario hasta encontrar el elemento buscado o no poder seguir realizando particiones.
  Mejor Caso: encontrarlo en la primera comparación O(1)
  Peor Caso: recorrer toda una rama del árbol que se va desminuyendo en la mitad. Se realizan k comparaciones, con k el tamaño de la rama del árbol, hasta llegar a una hoja.

Es decir , $1 = n / 2^k$, donde 1 representa la hoja, n el tamaño de los datos y la k el número de particiones realizadas. Nos interesa conocer k, por lo que despejándola, ya que todos los demás datos son conocidos, quedaría $k = \log_2 n$. Note que estos es una cota superior pues en el caso promedio, el algoritmo se detiene mucho antes.

Por lo tanto hemos concluido que el algoritmo de búsqueda binaria es orden logarítmico.

Si podemos escoger entre usar búsqueda secuencial y búsqueda binaria es mucho mejor usar el algoritmo de búsqueda binaria como lo indica el orden.