Técnico Informático

ESTRUCTURA DE DATOS

UNIDADES IV: TDA COLA

Escuela de Minas "Dr. Horacio Carrillo"
Universidad Nacional de Jujuy

¿Qué es una cola o fila?

Índice

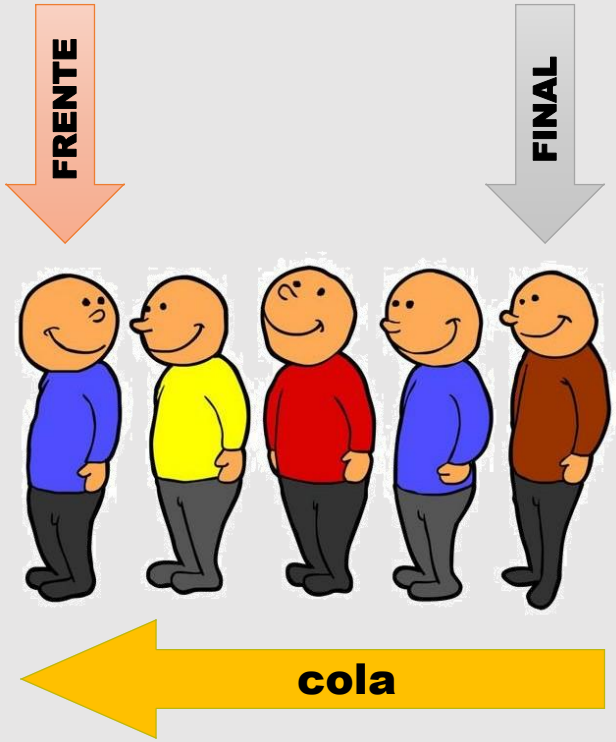
- Definición del TDA cola o fila
- Operaciones fundamentales
- Implementación
- Aplicaciones

Definición (1)

- Una cola o fila es una colección ordenada de elementos, con 3 características:
  - contiene elementos del mismo tipo (estructura homogénea),
  - la recuperación de elementos se realiza según el orden de almacenamiento (acceso FIFO) y
  - la cantidad de elementos almacenados varía durante la ejecución (estructura dinámica).

Definición (2)

- En una computadora, los elementos de una cola pueden almacenarse a partir de una dirección de memoria ocupando posiciones consecutivas y ordenadas.
- La recuperación de elementos se realiza por el frente y la inserción por el final.



Operaciones Fundamentales (1)

- Sobre el TDA cola se definen las siguientes operaciones:
  - Iniciar cola (*Init_Queue*)
  - Agregar elemento (*Push_Queue*)
  - Extraer elemento (*Pop_Queue*)
  - Determinar cola vacía (*Empty_Queue*)
  - Determinar cola llena (*Full_Queue*)
  - Consulta primer elemento (*Top_Queue*)
  - Consulta último elemento (*Bottom_Queue*)

Operaciones Fundamentales (2)

- *Init_Queue (iniciar cola)*
  - Propósito: inicializar la fila o cola (esto genera una cola vacía).
  - Entrada: cola de datos.
  - Salida: cola de datos inicializada (cola vacía).
  - Restricciones: ninguna.

Operaciones Fundamentales (3)

- *Push_Queue (encolar)*
  - Propósito: agregar un elemento al **final** de la fila o cola de datos.
  - Entrada: cola de datos y un nuevo elemento.
  - Salida: cola de datos con un nuevo elemento al final.
  - Restricciones: cola inicializada y contenedor de datos no completo.

Operaciones Fundamentales (4)

- *Full_Queue (cola llena)*
  - Propósito: determinar si el contenedor de datos de la cola está completo.
  - Entrada: cola de datos.
  - Salida: valor lógico *true* si el contenedor está completo o *false* en caso contrario.
  - Restricciones: cola inicializada.

Operaciones Fundamentales (5)

- *Pop_Queue (desencolar)*
  - Propósito: quitar el elemento del **frente** de la fila o cola de datos.
  - Entrada: cola de datos.
  - Salida: cola de datos con un elemento menos, se modifica el frente.
  - Restricciones: cola inicializada y contenedor de datos no vacío.

Operaciones Fundamentales (6)

- *Empty_Queue (cola vacía)*
  - Propósito: determinar si el contenedor de datos de la cola está vacío.
  - Entrada: cola de datos.
  - Salida: valor lógico *true* si el contenedor está vacío o *false* en caso contrario.
  - Restricciones: cola inicializada.

Operaciones Fundamentales (7)

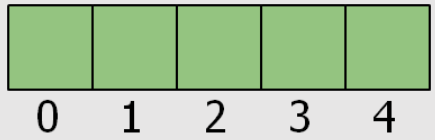
- *Top_Queue (frente o primero de la cola)*
  - Propósito: consultar el primer elemento de la cola de datos (**frente**).
  - Entrada: cola de datos.
  - Salida: primer elemento de la cola de datos.
  - Restricciones: cola inicializada y contenedor de datos no vacío.

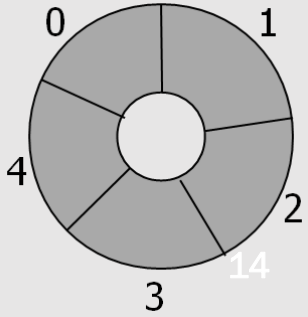
Operaciones Fundamentales (8)

- *Bottom_Queue (final o último de la cola)*
  - Propósito: consultar último elemento de la cola de datos (**final**).
  - Entrada: cola de datos.
  - Salida: último elemento de la cola de datos.
  - Restricciones: cola inicializada y contenedor de datos no vacío.

Alternativas de Implementación (1)

- Un **contenedor de datos** y **un índice.** El frente coincide con la primer posición del contenedor, mientras que el índice apunta al elemento final de la cola.
- Un **contenedor** y **2 índices.** Los índices se utilizan para apuntar, respectivamente, al frente y final de la cola.
  - almacenamiento lineal
  - almacenamiento circular





Alternativas de Implementación (2)

- El TDA cola, implementado utilizando un almacenamiento circular, puede presentar 2 variantes:
  - Implementación que prioriza la **velocidad de procesamiento.**

  - Implementación que prioriza el **espacio de almacenamiento.**

Implementación (1)

- TDA cola: Implementación que prioriza velocidad de procesamiento.

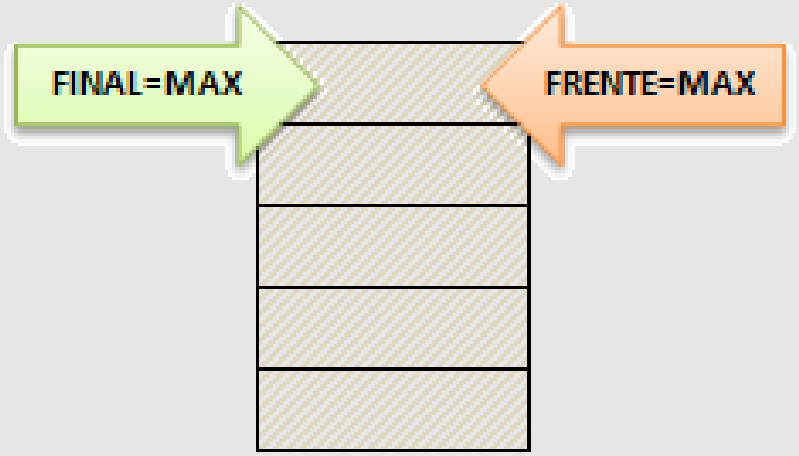
```
contenedor=ARREGLO [1..MAX] de ELEMENTOS
tcola=REGISTRO
        datos:contenedor
        frente,final:ENTERO
    FIN_REGISTRO
```

  - *frente*: indica el último elemento que se extrajo de la cola.
  - *final*: indica el último elemento que se agregó a la cola.

Implementación (2)

- Operación iniciar cola (*init_queue)*

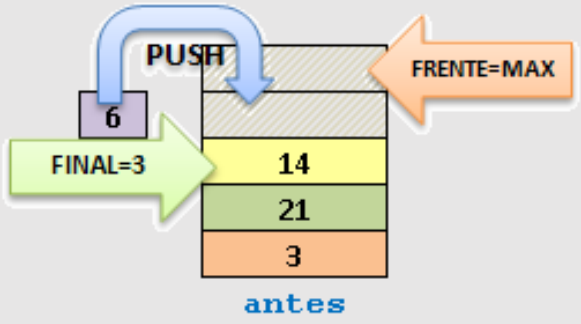
```
PROCEDIMIENTO iniciar_cola(E/S cola: tcola)
INICIO
        cola.frente←MAX
        cola.final←MAX
FIN
```

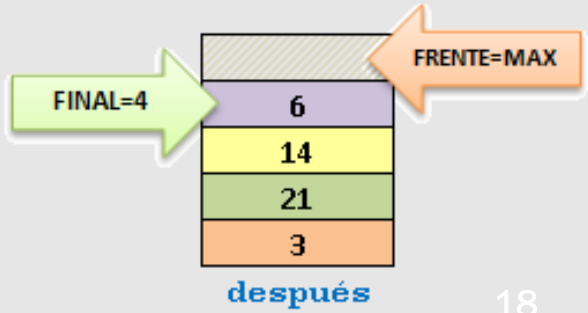


Implementación (3)

- Operación agregar cola (*push_queue*)

```
PROCEDIMIENTO agregar_cola(E/S cola:tcola,E nuevo:ELEMENTO)
INICIO
      SI cola_llena(cola)=VERDADERO ENTONCES
           ESCRIBIR "cola completa"
      SINO
           cola.final←siguiente(cola.final)
           cola.datos[cola.final]←nuevo
      FIN_SI
FIN
```

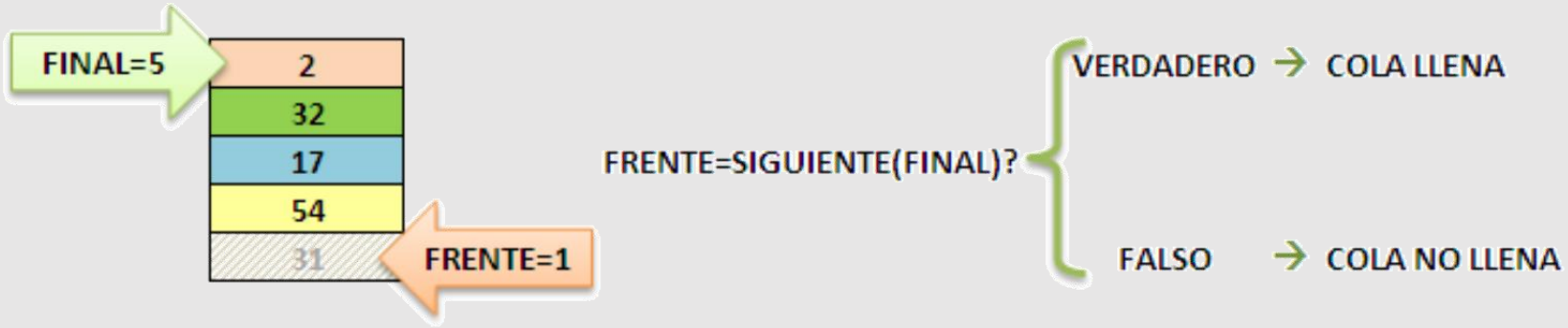




Implementación (4)

- Operación cola llena (*full_queue*)

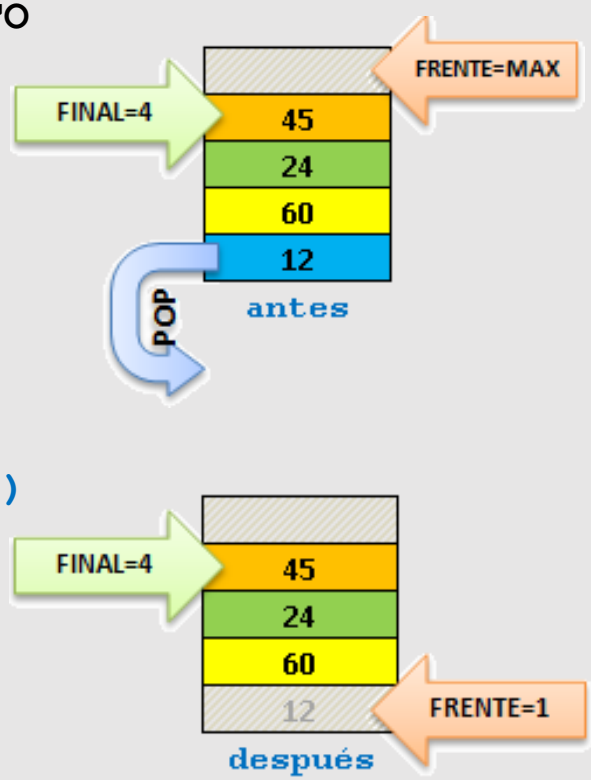
```
FUNCIÓN cola_llena(E cola:tcola):LÓGICO
INICIO
     cola_llena ← cola.frente=siguiente(cola.final)
FIN
```



Implementación (5)

- Operación quitar colar (*pop_queue*)

```
FUNCIÓN quitar_cola(E/S cola:tcola): ELEMENTO
VARIABLES
      extraido: ELEMENTO
INICIO
      SI cola_vacia(cola)=VERDADERO ENTONCES
           extraido←valor arbitrario
      SINO
           cola.frente←siguiente(cola.frente)
           extraido←cola.datos[cola.frente]
      FIN_SI
      quitar_cola←extraido
FIN
```



Implementación (6)

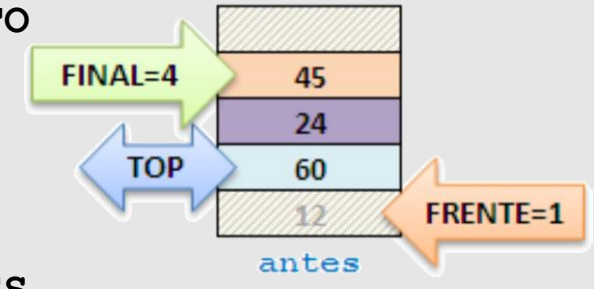
- Operación cola vacía (*empty_queue*)

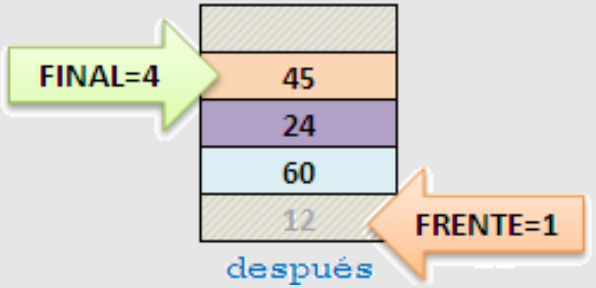
```
FUNCIÓN cola_vacia(E cola:tcola):LÓGICO
INICIO
        cola_vacia← cola.frente=cola.final
FIN
```

Implementación (7)

- Operación primero cola (*top_queue)*

```
FUNCIÓN primero_cola(E cola:tcola): ELEMENTO
VARIABLES
      consultado: ELEMENTO
INICIO
      SI cola_vacia(cola)=VERDADERO ENTONCES
           consultado←valor arbitrario
      SINO
           consultado←cola.datos[siguiente(cola.frente)]
      FIN_SI
      primero_cola←consultado
FIN
```

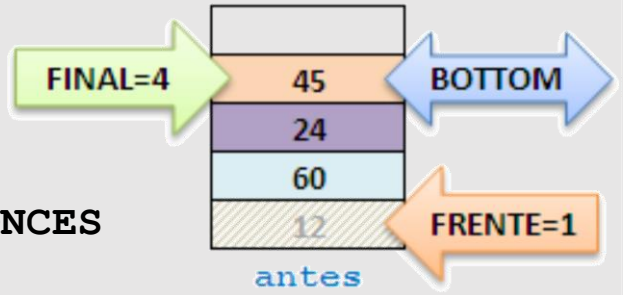


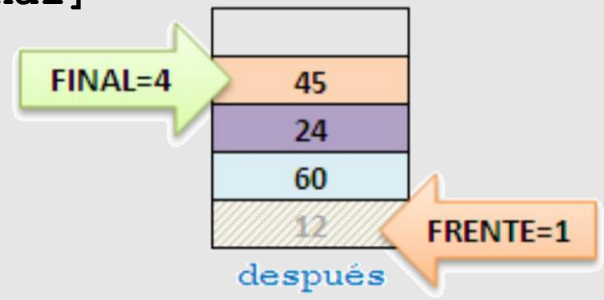


Implementación (8)

- Operación último cola (*bottom_queue)*

```
FUNCIÓN ultimo_cola(E cola:tcola): ELEMENTO
VARIABLES
      consultado: ELEMENTO
INICIO
      SI cola_vacia(cola)=VERDADERO ENTONCES
           consultado←valor arbitrario
      SINO
           consultado←cola.datos[cola.final]
      FIN_SI
      ultimo_cola←consultado
FIN
```

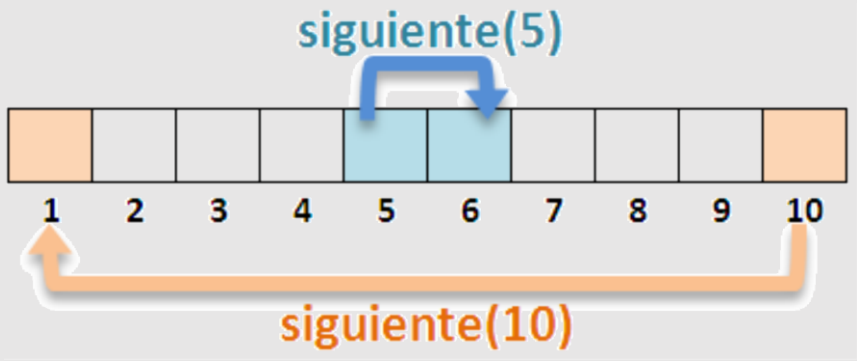




Implementación (9)

- Operación *next*

```
FUNCIÓN siguiente(E indice:entero):ENTERO
INICIO
      SI indice=MAX ENTONCES
          indice←1
      SINO
          indice←indice + 1
      FIN_SI
      siguiente←indice
FIN
```



Implementación (10)

- TDA cola: Implementación que prioriza espacio de almacenamiento en memoria.

```
contenedor=ARREGLO [1..MAX] de ELEMENTOS
tcola=REGISTRO
        datos:contenedor
        frente,final:ENTERO
        cantidad:ENTERO
    FIN_REGISTRO
```

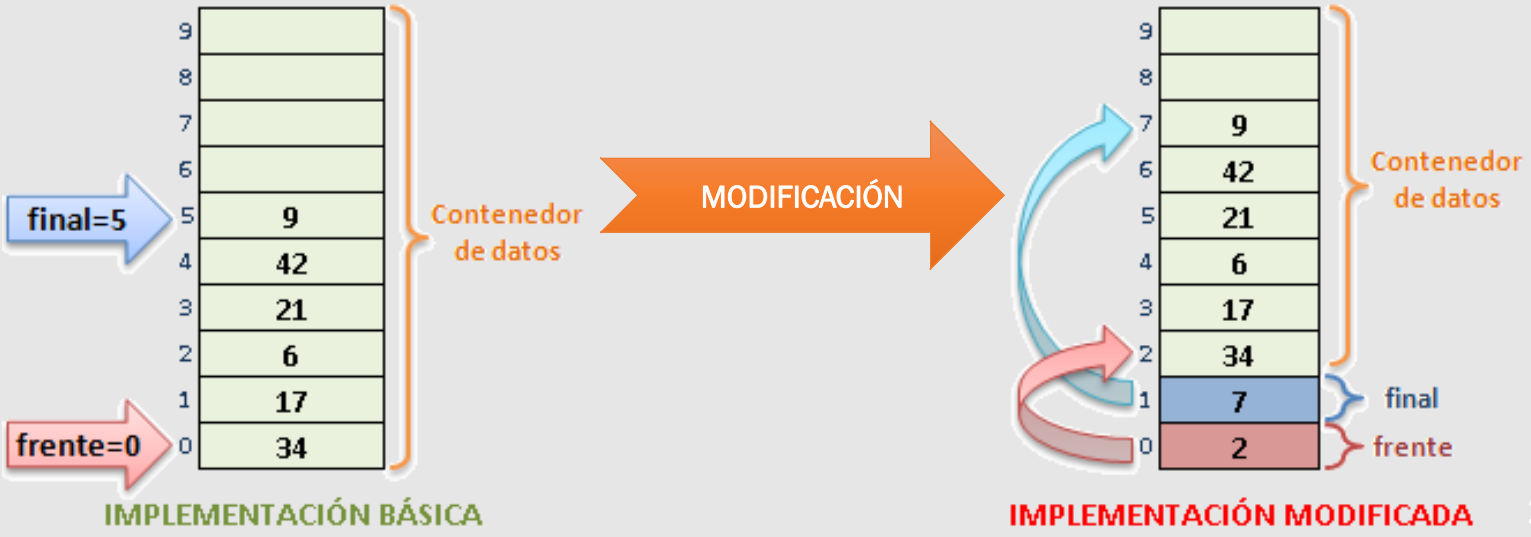
- ¿Cómo se modifican las operaciones básicas?

Implementación (11)

- Modificaciones en las operaciones del TDA cola:
  - *iniciar_cola*: se inicia el contador de elementos.
  - *agregar_cola*: se actualiza la cantidad de elementos de la cola (incremento).
  - *cola_llena*: se verifica si el contador alcanzó la **máxima capacidad** del contenedor.
  - *quitar_cola*: se actualiza la cantidad de elementos de la cola (decremento).
  - *cola_vacia*: se verifica si el contador alcanzó el valor de inicialización (cero).
  - *primero_cola* y *ultimo_cola*: no se modifican.

Implementación Modificada (1)

- Modifique la implementación básica del **TDA cola** (que prioriza velocidad de proceso) de forma que el **contenedor de datos** y los **índices** de la cola se almacenen en un único arreglo.
- ¿Cómo se modifican las **operaciones** de cola para esta implementación?



Implementación Modificada (2)

- TDA Cola modificado

```
const int MAX=10;
typedef int tcola[MAX];
```

- Operaciones modificadas: *iniciar_cola*

```
void iniciar_cola (tcola &q)
{
     q[0]←MAX-1; // frente
     q[1]←MAX-1; // final
}
```

Implementación Modificada (3)

- TDA Cola modificado

```
const int MAX=10;
typedef int tcola[MAX];
```

- Operaciones modificadas: *cola_vacia*

```
bool cola_vacia (tcola q)
{
    return q[1]==q[0];
}
```

Implementación Modificada (4)

- Operaciones modificadas: *agregar_cola*

```
void agregar_cola(tcola q,int nuevo)
{
      if (cola_llena(q)==true)
        cout << "COLA LLENA" << endl;
      else
        { q[1]=siguiente(q[1]);
          q[q[1]]=nuevo;
        }
}
```

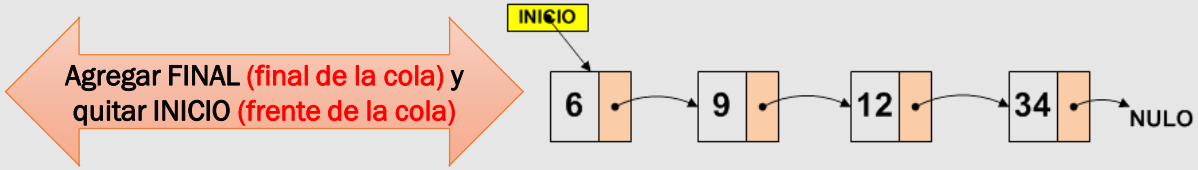
Implementación Modificada (5)

- Operaciones modificadas: *siguiente*

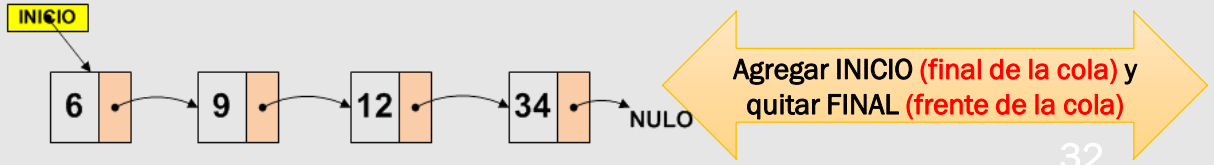
```
int siguiente(int indice)
{
  if (indice==MAX-1)
     indice=2;
  else
     indice++;
  return indice;
}
```

Implementación: Listas (1)

- Implementación del TDA Cola mediante **listas simples**
  - Alternativa 1:
    - Utilizar las operaciones *agregar_final* y *quitar_inicio* para representar el comportamiento de la cola.

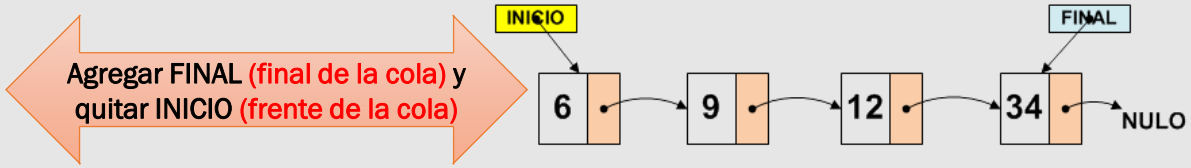


  - Alternativa 2:
    - Utilizar las operaciones *agregar_inicio* y *quitar_final* para representar el comportamiento de la cola.

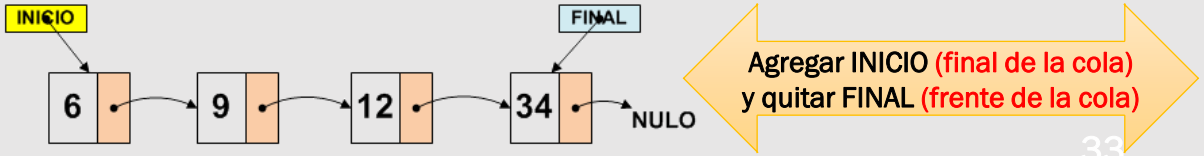


Implementación: Listas (2)

- Implementación del TDA Cola mediante **listas simples**
  - Alternativa 3:
    - Utilizar las operaciones *agregar_final* y *quitar_inicio* para representar el comportamiento de la cola.

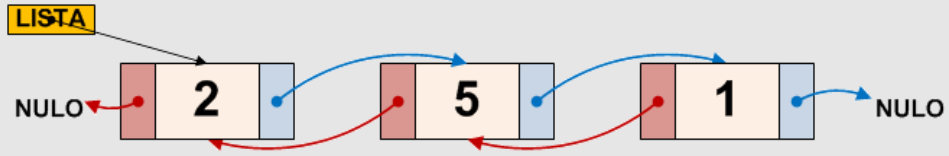


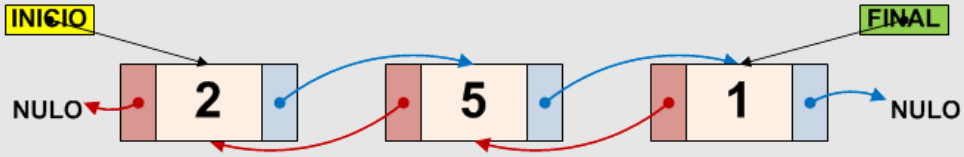
  - Alternativa 4:
    - Utilizar las operaciones *agregar_inicio* y *quitar_final* para representar el comportamiento de la cola.



Implementación: Listas (3)

- Implementación del TDA Cola mediante **listas dobles**
  - ¿Cuáles son las alternativas de implementación al utilizar listas dobles?





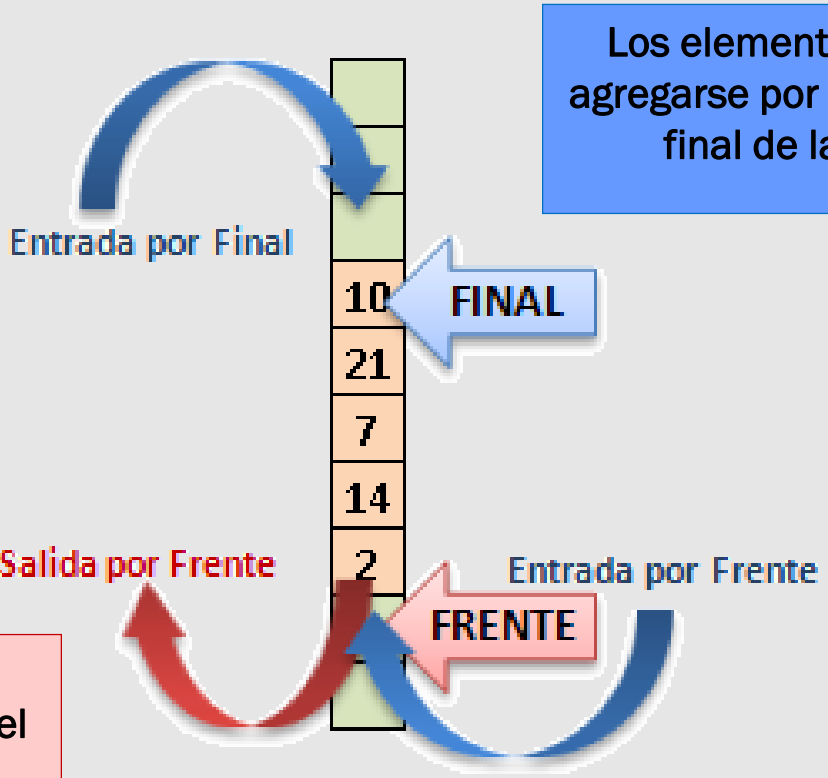
  - ¿Cuáles son las operaciones de listas dobles que pueden utilizarse para implementar las operaciones de cola para cada alternativa?

Bicolas (1)

- Una variante del TDA cola es la *bicola* o *cola doble* que permite la inserción y eliminación de elementos por ambos extremos del contenedor de datos.
- De acuerdo al extremo (frente, final) que admita la inserción/eliminación de elementos la *bicola* puede ser:
  - Con entrada restringida (se permite eliminar por *frente* y *final*; y agregar sólo por *final*)
  - Con salida restringida (se permite agregar por *frente* y *final;* y eliminar sólo por *frente*;)

Bicolas (2)

- Bicola con salida restringida



Bicolas (3)

- Bicola con salida restringida

```
const int MAX=10;
typedef int contenedor[MAX];
typedef struct tcola {
                    contenedor datos;
                    int frente;
                    int final;
                    };
```

Para implementar la bicola se utilizan las mismas estructuras de datos que para la cola estándar

Bicolas (4)

- Bicola con salida restringida

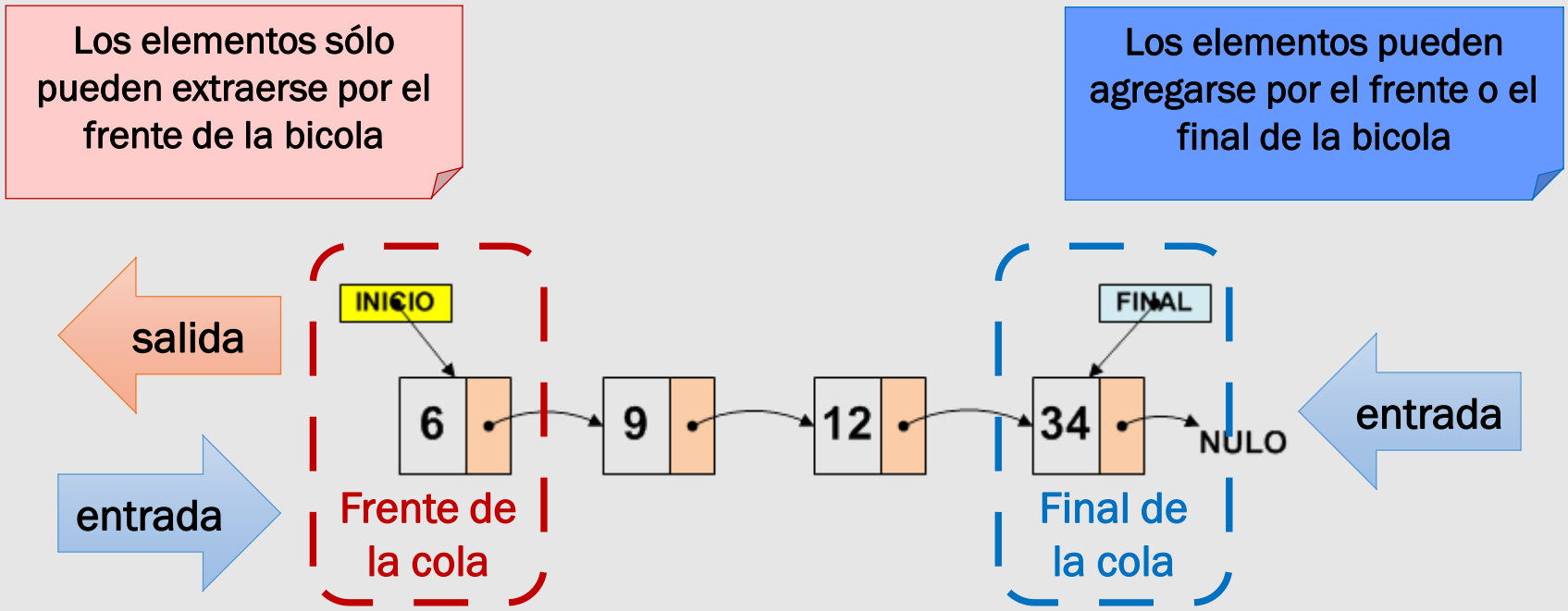
```
void agregar_bicola(tcola q,int nuevo,bool ultimo)
{ if (cola_llena(q)==true)
        cout << "No hay espacio" << endl;
  else
    if (ultimo==true)
      {q.final=siguiente(q.final);
       q.datos[q.final]=nuevo;}
    else
      {q.datos[q.frente]=nuevo;
       q.frente=anterior(q.frente);}
}
```

Agrega elementos al final de la fila

Agrega elementos al frente de la fila

Bicolas con Lista Simples

- Bicola con salida restringida



Bicolas (3)

- Bicola con salida restringida

```
typedef struct tnodo *pnodo;
typedef struct tnodo {
                    int dato;
                    pnodo sig;
                    };
typedef struct tbicola {
                    pnodo inicio; // frente
                    pnodo final; // final
                    };
```

Para implementar la bicola pueden utilizarse listas simples o dobles, con uno o 2 punteros a la estructura.

Bicolas (4)

- Bicola con salida restringida

```
void agregar_bicola(tbicola q,pnodo nuevo,bool ultimo)
{ if (q.inicio==NULL)
      { q.inicio=nuevo;
        q.final=nuevo; }
  else
    if (ultimo==true)
       {q.final->sig=nuevo;
        q.final=nuevo;}
    else
       {nuevo->sig=q.inicio;
        q.inicio=nuevo;}
}
```

Agrega elementos al final de la cola

Agrega elementos por el frente de la cola

Aplicaciones

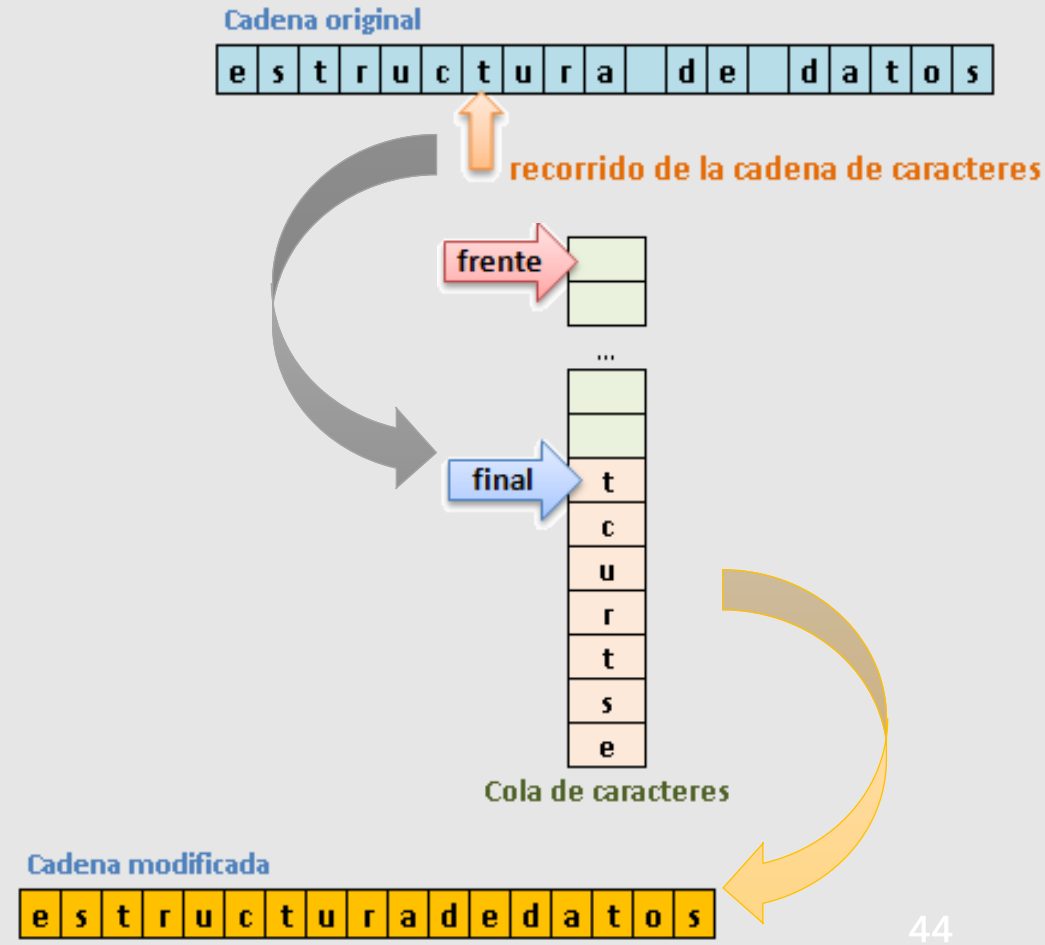
- El concepto de cola puede aplicarse para resolver:
  - simulación (teoría de colas)
  - algoritmos de reemplazo
  - colas de impresión,
  - acceso (escritura) almacenamiento secundario
  - sistemas de tiempo compartido
  - uso de la unidad central de proceso (UCP)

Ejemplo de Aplicación (1)

- Diseñe un algoritmo que elimine los espacios en blanco de una cadena de caracteres. Utilice el TDA cola en la propuesta de solución.
- **Propuesta de Solución**
  - El algoritmo recorrerá, carácter a carácter, la cadena de entrada guardando estos caracteres en una cola, excepto los espacios en blanco.
  - Luego, el contenido de la cola sobre-escribirá la cadena original, obteniéndose una cadena sin espacios.

Ejemplo de Aplicación (2)

- Conforme se recorre la cadena de entrada se guardan los caracteres leídos en una cola, salvo aquellos que sean espacios en blanco.
- Finalizado el recorrido, se inicia el vaciado de la cola, guardándose cada valor extraído en la cadena original. Se obtiene así una cadena sin espacios en blanco.



Ejemplo de Aplicación (3)

- Algoritmo para eliminar espacios usando TDA cola.

```
void eliminar_blancos(tcad &frase)
{ tcola cola;
  int i;
  iniciar_cola(cola);
  for(i=0;i<strlen(frase);i++)
     if (frase[i]!=' ')
        agregar_cola(cola,frase[i]);
  for(i=0;cola_vacia(cola)==false;i++)
     frase[i]=quitar_cola(cola);
  frase[i]='\0';
}
```

Inicialización de la cola

Inserción de datos

Extracción de datos

Bibliografía

- Joyanes Aguilar *et al*. Estructuras de Datos en C++. Mc Graw Hill. 2007.
- De Giusti, Armando *et al.* Algoritmos, datos y programas, conceptos básicos. Editorial Exacta. 1998.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.
- Hernández, Roberto *et al*. Estructuras de Datos y Algoritmos. Prentice Hall. 2001.