Universidad
Francisco de Vitoria
UFV Madrid

Estructuras de Datos y Algoritmos

Tema 3.2. Tipos de datos lineales.

Colas

Prof. Dr. P. Javier Herrera

Grado en Ingeniería Informática
Escuela Politécnica Superior

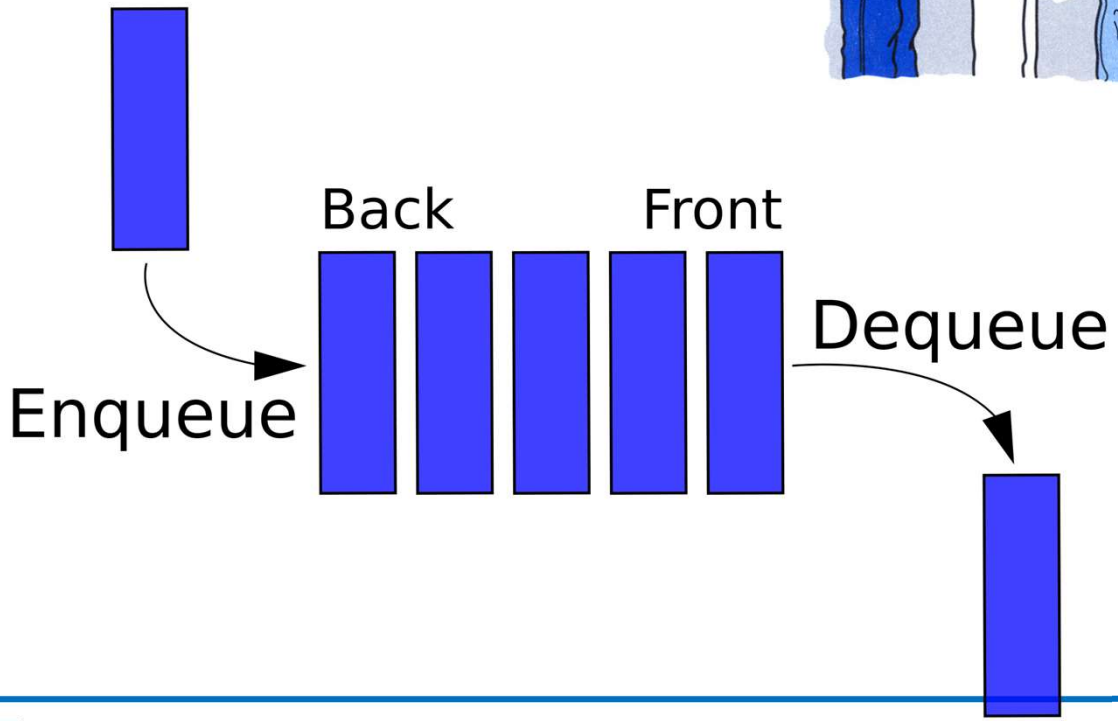
Universidad
Francisco de Vitoria
UFV Madrid

Contenido

- Colas: Conceptos generales
- Operaciones básicas
- Especificación algebraica
- Implementación estática: vector circular
- Implementación dinámica
- Ejercicio: Frases palíndromas

Cola (*queue*)


Rear of Queue
Front of Queue
next...
PAY HERE


Back
Front
Enqueue
Dequeue

Colas: Conceptos generales

- Estructura de datos lineal cuya característica principal es que los elementos se realiza en el mismo orden en que fueron almacenados, siguiendo el criterio de *el primero en entrar es el primero en salir* (FIFO – *First In First Out*).

- El comportamiento de las colas es totalmente independiente del tipo de los datos almacenados en ellas, por lo que se trata de un tipo de datos parametrizado.

- Las colas presentan dos zonas de interés, a partir de donde suele realizarse el acceso:
  - el extremo final: donde se insertan los elementos,
  - la cabecera: donde se consultan y eliminan los elementos.

- El comportamiento FIFO es muy utilizado en el diseño de algoritmos para diversas aplicaciones, sobre todo en simulación de sistemas.

Operaciones básicas

- Una cola puede ser:
  a) La cola vacía, con 0 elementos
  b) Una cola a la que llega un elemento, que pasa a ser el último.

- El TAD de las colas cuenta con las siguientes operaciones:
  - crear la cola vacía,
  - añadir un elemento al final de la cola,
  - eliminar el primer elemento en la cola,
  - consultar el primer elemento, y
  - determinar si la cola es vacía.

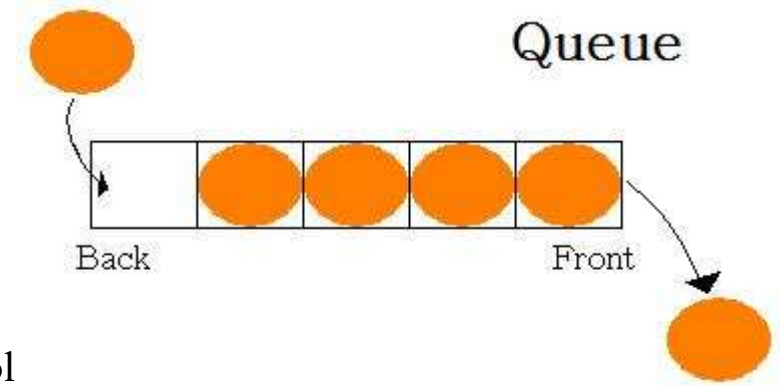
Especificación algebraica

especificación *COLAS*[*ELEM*]
usa *BOOLEANOS*
tipos *cola*
operaciones

cola-vacía	:	$\rightarrow$ *cola*	{ constructora }
pedir-vez	:	*cola elemento* $\rightarrow$ *cola*	{ constructora }
avanzar	:	*cola* $\rightarrow_p$ *cola*	
primero	:	*cola* $\rightarrow_p$ *elemento*	
es-cola-vacía?	:	*cola* $\rightarrow$ *bool*	

- El orden de inserción de los elementos en la cola determina la cola, por lo que las constructoras son **libres**. No son necesarias ecuaciones de equivalencia.

Especificación algebraica



variables

e : elemento

c : cola

ecuaciones

avanzar(cola-vacía) = error

avanzar(pedir-vez(*c*, *e*)) = cola-vacía ⇐ es-col

avanzar(pedir-vez(*c*, *e*)) = pedir-vez(avanzar(*c*), *e*) ⇐ ¬es-cola-vacía?(*c*)

primero(cola-vacía) = error

primero(pedir-vez(*c*, *e*)) = *e* ⇐ es-cola-vacía?(*c*)

primero(pedir-vez(*c*, *e*)) = primero(*c*) ⇐ ¬es-cola-vacía?(*c*)

es-cola-vacía?(cola-vacía) = cierto

es-cola-vacía?(pedir-vez(*c*, *e*)) = falso

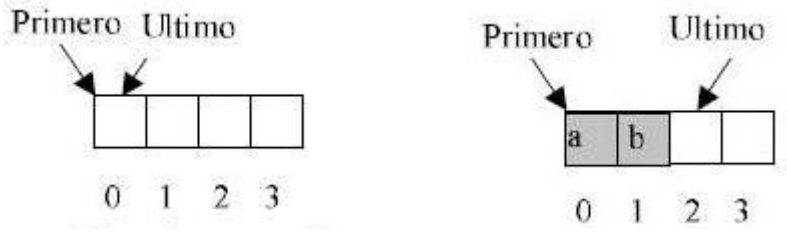
fespecificación

Implementación estática: vector circular

Representación

- Almacenamos los elementos de la cola en un vector, de izquierda a derecha según se vayan añadiendo.

- Mediante dos índices: *primero* y *último* accedemos a los puntos de interés:
  - Para insertar moveremos *último* hacia la derecha.
  - Para avanzar moveremos *primero* hacia la derecha.



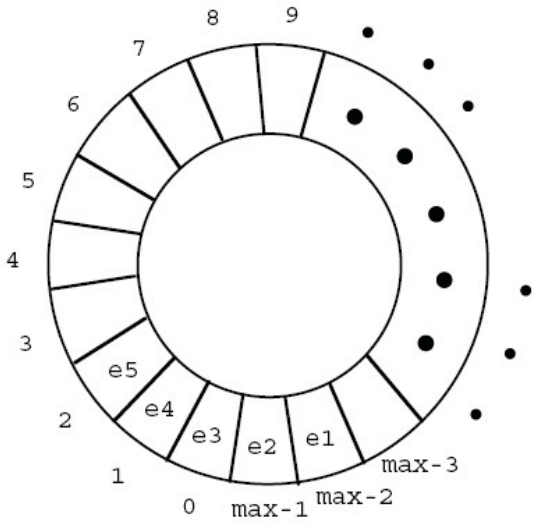
Implementación estática: vector circular

- Gestionamos el vector de forma “circular” para no tener que desplazar los elementos cuando el vector esté ocupado hasta la última posición pero haya posiciones libres al principio del mismo.

- El elemento siguiente al elemento que esta en la posición i se define mediante la función sig:

$$sig(i) = \begin{cases} i+1, & 1 \le i < N \\ 0, & i = N \end{cases}$$

donde N es el tamaño del vector utilizado.



Implementación estática: vector circular

- En el caso vacío, todavía no hay ni primero ni último. Hay que decidir cuáles pueden ser los valores apropiados de los índices.

- Para que la operación pedir-vez no necesite tratar por separado el caso en que se añade por primera vez un elemento a la cola vacía, conviene que *último* sea igual a N para que al pasar al siguiente ya valga 1, y *primero* sea igual a 1 para que se quede apuntando al primero.

- sig(*último*) = *primero* en dos ocasiones diferentes:
  - Cuando la cola está vacía
  - Cuando la cola está completamente llena.

- Para poder distinguir ambas situaciones, necesitamos añadir al registro también el número de elementos en la cola.

Implementación estática: vector circular

```
tipos
    cola  =  reg
               contenido[1..N] de elemento
               primero, último : 1..N
               tamaño : 0..N
             freg
ftipos
```

Implementación estática: vector circular

fun sig(i : 1..N) **dev** s : 1..N { }
    s := (i mod N) + 1
ffun

$O(1)$

- La cola se inicializa con tamaño igual a cero, señalando *primero* la primera posición del vector y *último* la última.

fun cola-vacía() **dev** c : cola { }
    $c.tamaño$:= 0
    $c.primero$:= 1 ; $c.último$:= N
ffun

$O(1)$

Implementación estática: vector circular

- Una vez se ha comprobado que hay hueco para un nuevo elemento, este se añade en la posición siguiente al último elemento en la cola, y el tamaño se incrementa en uno.

proc pedir-vez(*c* : cola, **e** *e* : elemento) { $O(1)$ }
    si c.tamaño = *N* **entonces** error(Espacio insuficiente)
    si no
        c.último := sig(*c.último*)
        c.contenido[*c.último*] := *e*
        c.tamaño := *c.tamaño* + 1
    fsi
fproc

Implementación estática: vector circular

- Una vez se ha comprobado que la cola no es vacía, se avanza haciendo que *primero* señale a la siguiente posición del primer elemento actual, y el tamaño se decrementa en uno.

proc avanzar(*c* : cola) { } $O(1)$

si *c.tamaño* = 0 **entonces** error(Cola vacía)

si no *c.primero* := sig(*c.primero*) ; *c.tamaño* := *c.tamaño* − 1

fsi

fproc

Implementación estática: vector circular

- El primer elemento es el valor en la posición *primero* de *contenido*.

```
fun primero(c : cola) dev e : elemento     {       }
    si c.tamaño = 0 entonces error(Cola vacía)
    si no e := c.contenido[c.primero]
    fsi
ffun
```

$O(1)$

- La cola está vacía si el tamaño es cero.

```
fun es-cola-vacía?(c : cola) dev b : bool  {       }
    b := (c.tamaño = 0)
ffun
```

$O(1)$

Implementación dinámica

tipos

enlace-cola = **puntero a** nodo-cola

nodo-cola = **reg**

valor : elemento

sig : enlace-cola

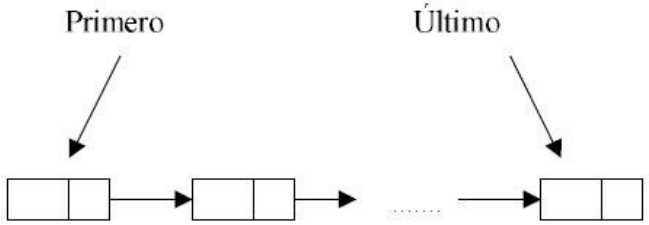
freg

cola = **reg**

primero, *último* : enlace-cola

freg

ftipos



- Utilizamos una estructura lineal enlazada.
- Mediante el acceso directo a ambos extremos de la cola garantizamos un tiempo constante para todas las operaciones.
- Se utilizan punteros para señalar el primer elemento (extremo por donde se consultan y eliminan los elementos) y al último (extremo por donde se introducen los elementos).

Implementación dinámica

- La cola vacía corresponde al caso en el que los enlaces a los extremos no apuntan a ninguna estructura.

fun cola-vacía() **dev** *c* : cola { }
    c.primero := nil ; *c.último* := nil $O(1)$
ffun

Implementación dinámica

- En la implementación de la operación de añadir se utiliza un enlace auxiliar para hacer la reserva de la memoria dinámica. Como el elemento se introduce por el final de la cola, el nuevo nodo se enlaza con el último. Sólo en el caso de que la cola estuviera vacía es necesario modificar el enlace *primero*.

```
proc pedir-vez(c : cola, e e : elemento)    {      }
var p : enlace-cola                                  O(1)
    reservar(p)
    p↑.valor := e ; p↑.sig := nil
    si c.primero = nil entonces c.primero := p
    si no (c.último)↑.sig := p
    fsi
    c.último := p
fproc
```

Implementación dinámica

- En la implementación de la operación de avanzar en la cola también se utiliza un enlace auxiliar, en este caso para liberar la memoria dinámica asignada al primer elemento (cuando existe). En el caso de que la cola se quede vacía, es también necesario anular el enlace *último*.

```
proc avanzar(c : cola)                    {      }
var p : enlace-cola                                          O(1)
    si c.primero = nil entonces error(Cola vacía)
    si no
       p := c.primero ; c.primero := p↑.sig
       si c.primero = nil entonces
          c.último := nil
       fsi
       liberar(p)
    fsi
fproc
```

Implementación dinámica

fun primero(*c* : cola) **dev** *e* : elemento { }
  si *c.primero* = nil **entonces** error(Cola vacía)
  si no *e* := (*c.primero*)↑.*valor*
  fsi
ffun

$O(1)$

- Para determinar si la cola está vacía, basta comprobar si el enlace al primer elemento es nulo (el enlace al último lo será también).

fun es-cola-vacía?(*c* : cola) **dev** *b* : bool { }
  b := (*c.primero* = nil)
ffun

$O(1)$

Ejercicio: Frases palíndromas

- Desarrollar una función iterativa de coste lineal en tiempo que decida si una frase dada como sucesión de caracteres (leída desde el teclado) es o no palíndroma, utilizando como estructuras auxiliares una pila y una cola.

```
proc palíndroma?()
var c : cola[car], p : pila[car]
    { leer desde la entrada hacia la cola y la pila }
    c := cola-vacía()
    p := pila-vacía()
    leer(x)
    mientras x ≠ fin hacer
       si x ≠ ' ' entonces pedir-vez(c, x) ; apilar(x, p) fsi
       leer(x)
    fmientras
```

Ejercicio: Frases palíndromas

```
    { comparar la cola con la pila }
    b := cierto
    mientras b ∧ ¬es-cola-vacía?(c) hacer
       b := (primero(c) = cima(p))
       avanzar(c) ; desapilar(p)
    fmientras
    si b entonces imprimir(Es palíndroma)
    si no
       imprimir(No es palíndroma)
       anular-cola(c) ; anular-pila(p)
    fsi
fproc
```

Bibliografía

- Martí, N., Ortega, Y., Verdejo, J.A. *Estructuras de datos y métodos algorítmicos*. Ejercicios resueltos. Pearson/Prentice Hall, 2003. Capítulo 4

- Peña, R.; *Diseño de programas. Formalismo y abstracción*. Tercera edición. Prentice Hall, 2005. Capítulo 6

(Estas transparencias se han realizado a partir de aquéllas desarrolladas por los profesores Clara Segura, Alberto Verdejo y Yolanda García de la UCM, y la bibliografía anterior)