

Estructuras de datos: Pilas, Colas, Listas

Algoritmos

Facultad de Informática
Universidad de A Coruña



UNIVERSIDADE DA CORUÑA

Table of Contents

1 Pilas

2 Colas

3 Listas

Table of Contents

1 Pilas

2 Colas

3 Listas

Pilas

- Acceso limitado al último elemento insertado
- Operaciones básicas: apilar, desapilar y cima.
 - desapilar o cima en una pila vacía es un error en el TDA pila.
 - Quedarse sin espacio al apilar es un error de implementación.
- Cada operación debería tardar una cantidad **constante** de tiempo en ejecutarse.
 - Con independencia del número de elementos apiladas.

Pseudocódigo: Implementación a base de vectores (i)

```
tipo Pila = registro  
  Cima_de_pila : 0..Tamaño_máximo_de_pila  
  Vector_de_pila : vector [1..Tamaño_máximo_de_pila]  
                  de Tipo_de_elemento  
  
fin registro  
  
procedimiento Crear Pila ( P )  
  P.Cima_de_pila := 0  
fin procedimiento  
  
función Pila Vacía ( P ) : test  
  devolver P.Cima_de_pila = 0  
fin función
```

Pseudocódigo: Implementación a base de vectores (ii)

```
procedimiento Apilar ( x, P )  
  si P.Cima_de_pila = Tamaño_máximo_de_pila entonces  
    error Pila llena  
  sino  
    P.Cima_de_pila := P.Cima_de_pila + 1;  
    P.Vector_de_pila[P.Cima_de_pila] := x  
fin procedimiento  
función Cima ( P ) : Tipo_de_elemento  
  si Pila Vacía (P) entonces error Pila vacía  
  sino devolver P.Vector_de_pila[P.Cima de Pila]  
fin función  
procedimiento Desapilar ( P )  
  si Pila Vacía (P) entonces error Pila vacía  
  sino P.Cima_de_pila := P.Cima_de_pila - 1  
fin procedimiento
```

Table of Contents

1 Pilas

2 Colas

3 Listas

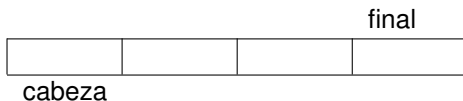
Colas

- Operaciones básicas: insertar, quitarPrimero y primero.
- Cada rutina debería ejecutarse en tiempo constante.

Implementación circular a base de vectores (i)

- La implementación circular devuelve **cabeza** y **fin** al principio del vector cuando rebasan la última posición.

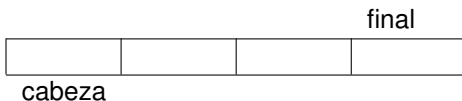
1) Crear_Cola (C)



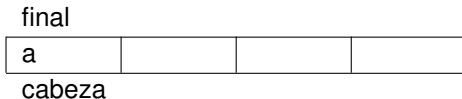
Implementación circular a base de vectores (i)

- La implementación circular devuelve **cabeza** y **fin** al principio del vector cuando rebasan la última posición.

1) Crear_Cola (C)



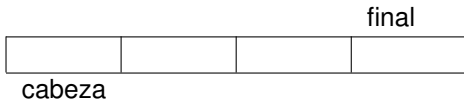
2) Insertar_en_Cola (a,C)



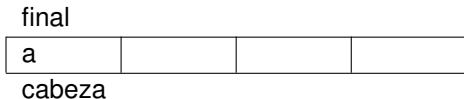
Implementación circular a base de vectores (i)

- La implementación circular devuelve **cabeza** y **fin** al principio del vector cuando rebasan la última posición.

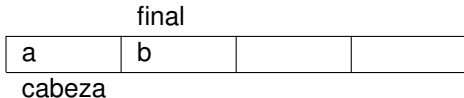
1) Crear_Cola (C)



2) Insertar_en_Cola (a,C)

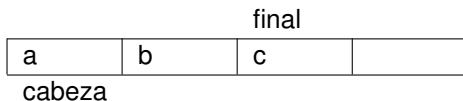


3) Insertar_en_Cola (b,C)

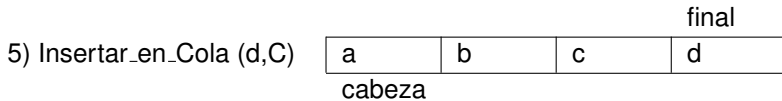
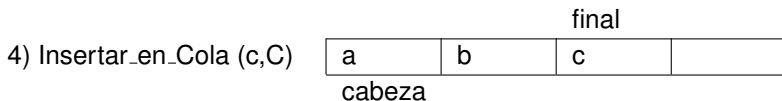


Implementación circular a base de vectores (i)

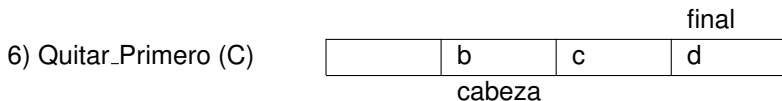
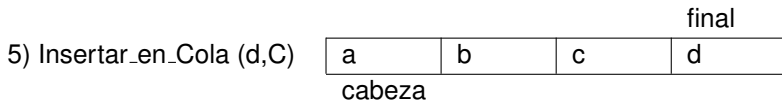
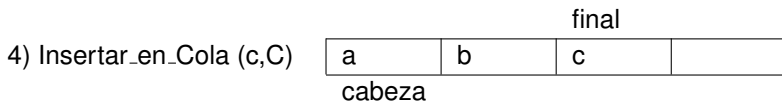
4) Insertar_en_Cola (c,C)



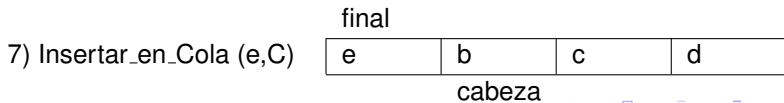
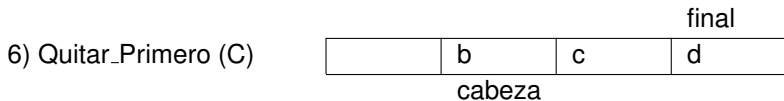
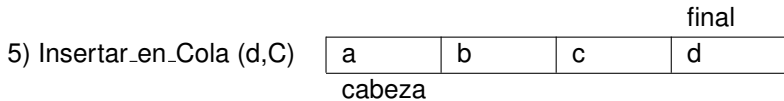
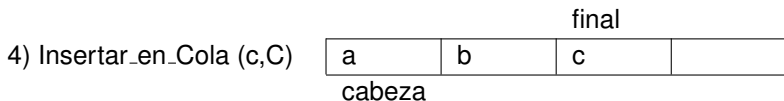
Implementación circular a base de vectores (i)



Implementación circular a base de vectores (i)



Implementación circular a base de vectores (i)



Pseudocódigo (i)

```
tipo Cola = registro  
  Cabeza_deCola, Final_deCola: 1..Tamaño_máximo_deCola  
  Tamaño_deCola : 0..Tamaño_máximo_deCola  
  Vector_deCola : vector [1..Tamaño_máximo_deCola]  
                  de Tipo_de_elemento  
  
fin registro  
procedimiento Crear_Cola ( C )  
  C.Tamaño_deCola := 0;  
  C.Cabeza_deCola := 1;  
  C.Final_deCola := Tamaño_máximo_deCola  
fin procedimiento  
función Cola_Vacía ( C ) : test  
  devolver C.Tamaño_deCola = 0  
fin función
```


Pseudocódigo (ii)

```
procedimiento incrementar ( x ) (* privado *)  
  si x = Tamaño_máximo_de_cola entonces x := 1  
  sino x := x + 1  
fin procedimiento
```

```
procedimiento Insertar_en_Cola ( x, C )  
  si C.Tamaño_de_Cola = Tamaño_máximo_de_cola entonces  
    error Cola llena  
  sino  
    C.Tamaño_de_cola := C.Tamaño_de_cola + 1;  
    incrementar(C.Final_de_cola);  
    C.Vector_de_cola[C.Final_de_cola] := x;  
fin procedimiento
```

Pseudocódigo (iii)

```
función Quitar_Primeros ( C ) : Tipo_de_elemento
  si Cola_Vacía ( C ) entonces
    error Cola vacía
  sino
    C.Tamaño_de_cola := C.Tamaño_de_cola - 1;
    x := C.Vector_de_cola[C.Cabeza_de_cola];
    incrementar(C.Cabeza_de_cola);
    devolver x
fin función

función Primeros ( C ) : Tipo_de_elemento
  si Cola_Vacía ( C ) entonces
    error Cola vacía
  sino
    devolver Vector_de_cola[C.Cabeza_de_cola]
fin función
```

Table of Contents

1 Pilas

2 Colas

3 Listas

Listas

- Operaciones básicas:
 - Visualizar su contenido.
 - Buscar la posición de la primera ocurrencia de un elemento.
 - Insertar y Eliminar un elemento en alguna posición.
 - Buscar_k_esimo, que devuelve el elemento de la posición indicada

Implementación de listas a base de vectores

- Tiene que declararse el tamaño de la lista.
 - Exige sobrevaloración.
 - Consume mucho espacio.
- Complejidad computacional de las operaciones:
 - `Buscar_k_esimo`, tiempo constante
 - `Visualizar` y `Buscar`, tiempo lineal.
 - `Insertar` y `Eliminar` son costosas.
 - Insertar o eliminar un elemento exige, en promedio, desplazar la mitad de los valores, $O(n)$.
 - La construcción de una lista o la eliminación de todos sus elementos podría exigir un tiempo cuadrático.

Implementación de listas a base de apuntadores

- Cada nodo apunta al siguiente; el último no apunta a nada.
- La lista es un puntero al primer nodo (y al último).
- Complejidad computacional de las operaciones:
 - Visualizar y Buscar, tiempo lineal.
 - Buscar_k_esimo, tiempo lineal.
 - Eliminar realiza un cambio de apuntadores y una orden `dispose`, $O(1)$.
 - Usa `Buscar_anterior` cuyo tiempo de ejecución es lineal.
 - Insertar tras una posición `p` requiere una llamada a `new` y dos maniobras con apuntadores, $O(1)$.
 - Buscar la posición `p` podría llevar tiempo lineal.
 - Un nodo **cabecera** facilita la inserción y la eliminación al comienzo de la lista.

Implementación de listas doblemente enlazadas

- Cada nodo apunta al siguiente y al anterior.
- Duplica el uso de la memoria necesaria para los punteros.
- Duplica el coste de manejo de punteros al insertar y eliminar.
- La eliminación se simplifica.
 - No es necesario buscar el elemento anterior.

Pseudocódigo: Implementación con un nodo cabecera (i)

```
tipo PNode = puntero a Node
  Lista = PNode
  Posición = PNode
  Node = registro
    Elemento : Tipo_de_elemento
    Siguiente : PNode
fin registro
procedimiento Crear Lista ( L )
  nuevo ( tmp );
  si tmp = nil entonces error Memoria agotada
  sino
    tmp^.Elemento := { nodo cabecera };
    tmp^.Siguiente := nil;
    L := tmp
fin procedimiento
```


Pseudocódigo: Implementación con un nodo cabecera (ii)

```
función Lista Vacía ( L ) : test  
    devolver L^.Siguiente = nil  
fin función
```

```
función Buscar ( x, L ) : posición de la 1ª ocurrencia  
    o nil  
    p := L^.Siguiente;  
    mientras p <> nil y p^.Elemento <> x hacer  
        p := p^.Siguiente;  
    devolver p  
fin función
```

```
función Último Elemento ( p ) : test { privada }  
    devolver p^.Siguiente = nil  
fin función
```

Pseudocódigo: Implementación con un nodo cabecera (iii)

```
función Buscar Anterior ( x, L ) : posición anterior a x  
                                o a nil { privada }  
  
    p := L;  
    mientras p^.Siguiente <> nil y  
                p^.Siguiente^.Elemento <> x hacer  
        p := p^.Siguiente;  
    devolver p  
fin función  
  
procedimiento Eliminar ( x, L )  
    p := Buscar Anterior ( x, L );  
    si Último Elemento ( p ) entonces error No encontrado  
    sino tmp := p^.Siguiente;  
        p^.Siguiente := tmp^.Siguiente;  
        liberar ( tmp )  
fin procedimiento
```

Pseudocódigo: Implementación con un nodo cabecera (iv)

```
procedimiento Insertar ( x, L, p )  
  nuevo ( tmp );   { Inserta después de la posición p }  
  si tmp = nil entonces  
    error Memoria agotada  
  sino  
    tmp^.Elemento := x;  
    tmp^.Siguiente := p^.Siguiente;  
    p^.Siguiente := tmp  
fin procedimiento
```