

Tipos abstractos de datos, Colas, Pilas y Listas enlazadas.

Guillermo Palma

Universidad Simón Bolívar
Departamento de Computación y T.I.

CI-2612: Algoritmos y Estructuras II



Plan

- 1 Tipos abstractos de datos
- 2 TAD Conjunto
- 3 TAD Cola
- 4 TAD Pila
- 5 Listas enlazadas



Tipo abstractos de datos

Definición 1

Se puede pensar en un tipo abstracto de datos (TAD) como un modelo matemático con una serie de operaciones definidas sobre el modelo [1].

Definición 2

An abstract data type defines a class of abstract objects which is completely characterized by the operations available on those objects. This means that an abstract data type can be defined by defining the characterizing operations for that type. [3]



Características de los tipos abstractos de datos

- Generalizaciones de los tipos primitivos (entero, real, boolean, etc)
- Encapsulación de cierto tipo de dato
- Contiene la definición del tipo y sus operaciones
- Una vez definido se puede usar como un tipo de dato primitivo
- Un TAD puede incluir otro TAD
- Las operaciones de un TAD pueden involucrar otros TADs
- El tipo es definido en base a su comportamiento desde el punto de vista del usuario



Tipo de datos, estructuras de datos y tipos abstractos de datos

Definición de tipo de dato

Es el conjunto de valores que puede tomar una variable en lenguaje un programación [1].

Ejemplos de tipo de datos básicos o primitivos

- Entero (int)
- Real (float, double, real)
- Booleano (bool)
- String
- Caracter (char)
- Void (Vacío significa sin ningún valor)



Tipo de datos, estructuras de datos y tipos abstractos de datos

Definición de estructuras de datos

... son conjuntos de variables, quizás de tipos distintos, conectadas entre sí de diferentes formas [1].

Ejemplos de estructuras de datos

- Arreglo
- Record o Struct
- Lista enlazadas
- Tuplas
- Union



Apuntador

Definición

Un apuntador es una celda cuyo valor indica o señala a otra. ... el hecho de que una celda A sea un apuntador a la celda B se indica con una flecha de A a B. [1].

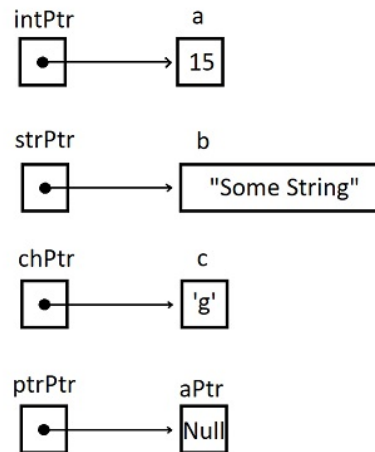


Figura: Ejemplo de apuntadores



Especificación de un tipo abstracto de datos

- Un estilo de especificación de TADs es el llamado *basado en modelos*
- Se especifica un TAD por medio del lenguaje natural y por medio de estructuras matemáticas como, por ejemplo, conjuntos y relaciones, entre otras.
- La especificación por modelos tiene tres componentes:
 - ▶ Modelo Abstracto de Representación
 - ▶ Invariante de Representación
 - ▶ Operaciones



Especificación del TAD Conjunto (por [4])

- Un TAD Conjunto es una estructura de datos que se desea implementar en el computador, que simule las características de la estructura matemática Conjunto.
- Las operaciones del TAD Conjunto son:
 - ▶ agregar un elemento a un conjunto,
 - ▶ eliminar un elemento de un conjunto
 - ▶ extraer un elemento cualquiera de un conjunto no-vacío
 - ▶ determinar si un elemento está o no en un conjunto
 - ▶ determinar si un conjunto está vacío o no
 - ▶ unir dos conjuntos
 - ▶ intersectar dos conjuntos
 - ▶ calcular diferencia de dos conjuntos



Modelo abstracto de representación del TAD Conjunto

Modelo de Representación

```
const MAX : int  
var contenido : set T
```



Invariante de representación del TAD Conjunto

Invariante de Representación

$$MAX > 0 \wedge \# \text{ contenido} \leq MAX$$



Operaciones del TAD Conjunto

Especificación \mathbb{A} de TAD *Conjunto* (T)

Modelo de Representación

```
const MAX : int
var contenido : set T
```

Invariante de Representación

$$MAX > 0 \wedge \# \text{ contenido} \leq MAX$$

Operaciones

```
proc crear (in m : int ; out c : Conjunto)
{ Pre : m > 0 }
{ Post : c.MAX = m  $\wedge$  c.contenido =  $\emptyset$  }

proc agregar (in-out c : Conjunto ; in x : T)
{ Pre : x  $\notin$  c.contenido  $\Rightarrow$   $\#$  c.contenido < c.MAX }
{ Post : c.contenido = c0.contenido  $\cup$  { x } }

proc eliminar (in-out c : Conjunto ; in x : T)
{ Pre : true }
{ Post : c.contenido = c0.contenido - { x } }
```



Operaciones del TAD Conjunto

```

proc extraer ( in  $c : \text{Conjunto}$  ; out  $x : T$  )
  { Pre :  $c.\text{contenido} \neq \emptyset$  }
  { Post :  $x \in c.\text{contenido}$  }

proc pertenece ( in  $c : \text{Conjunto}$  ; in  $x : T$  ; out  $p : \text{boolean}$  )
  { Pre : true }
  { Post :  $p \equiv (x \in c.\text{contenido})$  }

proc vacio ( in  $c : \text{Conjunto}$  ; out  $v : \text{boolean}$  )
  { Pre : true }
  { Post :  $v \equiv (c.\text{contenido} = \emptyset)$  }

proc unir ( in  $c0, c1 : \text{Conjunto}$  ; in  $m : \text{int}$  ; out  $c : \text{Conjunto}$  )
  { Pre :  $\# c0.\text{contenido} + \# c1.\text{contenido} \leq m$  }
  { Post :  $c.\text{MAX} = m \wedge c.\text{contenido} = c0.\text{contenido} \cup c1.\text{contenido}$  }

proc interseccion ( in  $c0, c1 : \text{Conjunto}$  ; in  $m : \text{int}$  ; out  $c : \text{Conjunto}$  )
  { Pre :  $\# c0.\text{contenido} \leq m \wedge \# c1.\text{contenido} \leq m$  }
  { Post :  $c.\text{MAX} = m \wedge c.\text{contenido} = c0.\text{contenido} \cap c1.\text{contenido}$  }

proc restar ( in  $c0, c1 : \text{Conjunto}$  ; in  $m : \text{int}$  ; out  $c : \text{Conjunto}$  )
  { Pre :  $\# c0.\text{contenido} \leq m$  }
  { Post :  $c.\text{MAX} = m \wedge c.\text{contenido} = c0.\text{contenido} - c1.\text{contenido}$  }

```



Fin TAD

Definición del TAD Cola

Definición

El TAD Cola es una secuencia dinámica de elementos, en el cual el elemento a eliminar está preespecificado. El elemento a eliminar va a ser, el primer elemento insertado en la secuencia. A esto se le conoce como la política first-in, first-out, o FIFO.



Especificación del TAD Cola (por [4])

Especificación \mathbb{A} de TAD *Cola* (T)

Modelo de Representación

```
const MAX : int
var contenido : seq T
```

Invariante de Representación

$$MAX > 0 \wedge \# \text{ contenido} \leq MAX$$

Operaciones

```
proc crear (in m : int; out c : Cola)
{ Pre : m > 0 }
{ Post : c.MAX = m  $\wedge$  c.contenido =  $\langle \rangle$  }

proc encolar (in-out c : Cola; in x : T)
{ Pre :  $\# c.\text{contenido} < c.MAX$  }
{ Post : c.contenido =  $c_0.\text{contenido} \mathbin{++} \langle x \rangle$  }

proc desencolar (in-out c : Cola)
{ Pre : c.contenido  $\neq \langle \rangle$  }
{ Post : c.contenido =  $\text{tl } c_0.\text{contenido}$  }

proc primero (in c : Cola; out x : T)
{ Pre : c.contenido  $\neq \langle \rangle$  }
{ Post :  $x = \text{hd } c.\text{contenido}$  }

proc vacia (in c : Cola; out v : boolean)
{ Pre : true }
{ Post :  $v \equiv (c.\text{contenido} = \langle \rangle)$  }
```

Fin TAD



Implementación del TAD Cola

- Se implementa con un arreglo donde $Q[1..n]$
- $Q.head$ es un apuntador al índice donde fue insertado el más antiguo elemento.
- $Q.tail$ es un apuntador al índice donde va a ser insertado el próximo elemento.
- La cola reside en las posiciones $Q.head, Q.head + 1, \dots, Q.tail - 1$.
- Si $Q.head = Q.tail$ indica que la cola esta vacía.
- Se tiene que $Q.length = n$



Implementación del TAD Cola

Función Empty(Q)

inicio

└ **devolver** $Q.elems = 0$;

Procedimiento Enqueue(Q, x)

inicio

└ **si** $Q.elems = Q.length$ **entonces**

└└ **devolver** "Error, overflow" ;

$Q[Q.tail] \leftarrow x$;

└ **si** $Q.tail = Q.length$ **entonces**

└└ $Q.tail \leftarrow 1$;

└ **en otro caso**

└└ $Q.tail \leftarrow Q.tail + 1$;

└ $Q.elems \leftarrow Q.elems + 1$;



Continuación de la implementación del TAD Cola

Función Dequeue(Q)

inicio

└ **si** $Empty(Q)$ **entonces**

└└ **devolver** "Error, underflow" ;

$x \leftarrow Q[Q.head]$;

└ **si** $Q.head = Q.length$ **entonces**

└└ $Q.head \leftarrow 1$;

└ **en otro caso**

└└ $Q.head \leftarrow Q.head + 1$;

└ $Q.elems \leftarrow Q.elems - 1$;

└ **devolver** x ;



Ejemplo del TAD Cola

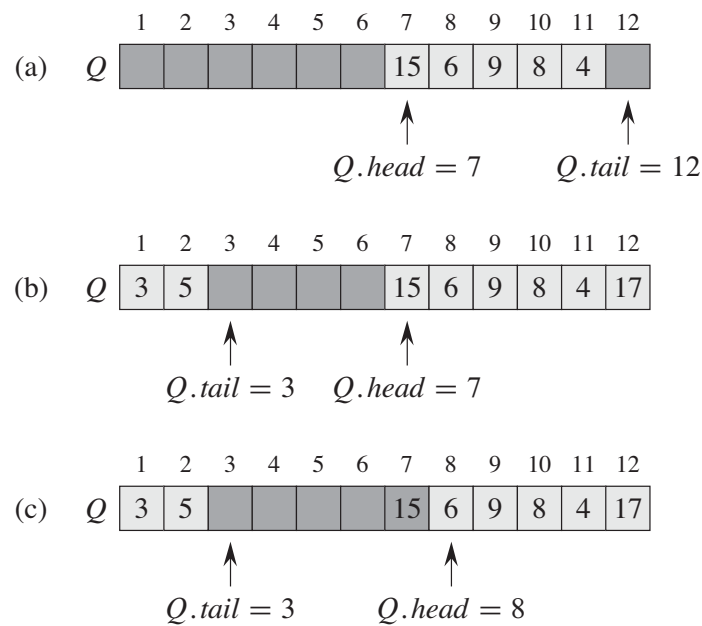


Figura: Ejemplo de implementación de una cola como un arreglo. Fuente [2]



Definición del TAD Pila

Definición

El TAD Pila es una secuencia dinámica de elementos, en el cual el elemento a eliminar está preespecificado. El elemento a eliminar va a ser, el último elemento insertado en la secuencia. A esto se le conoce como la política last-in, first-out, o LIFO.



Implementación del TAD Pila

- Se implementa con un arreglo donde $S[1..n]$
- $S.top$ es un apuntador al índice donde fue insertado el más reciente elemento.
- La pila consiste en $S[1..S.top]$, donde $S[1]$ es el último elemento y $S[S.top]$ es el primero.
- $S.top = 0$ indica que la pila esta vacía.



Implementación del TAD Pila

Función Empty(S)

inicio

└ **devolver** $S.elems = 0$;

Procedimiento Push(S, x)

inicio

└ **si** $S.top = S.length$ **entonces**
 └ **devolver** "Error, overflow";
 $S.top \leftarrow S.top + 1$;
 $S[S.top] \leftarrow x$;



Continuación de la implementación del TAD Pila

Función Pop(S)

inicio

```

si Empty(S) entonces
  | devolver "Error, underflow" ;
en otro caso
  |  $S.top \leftarrow S.top - 1$  ;
  | devolver  $S[S.top + 1]$  ;

```



Ejemplo del TAD Pila

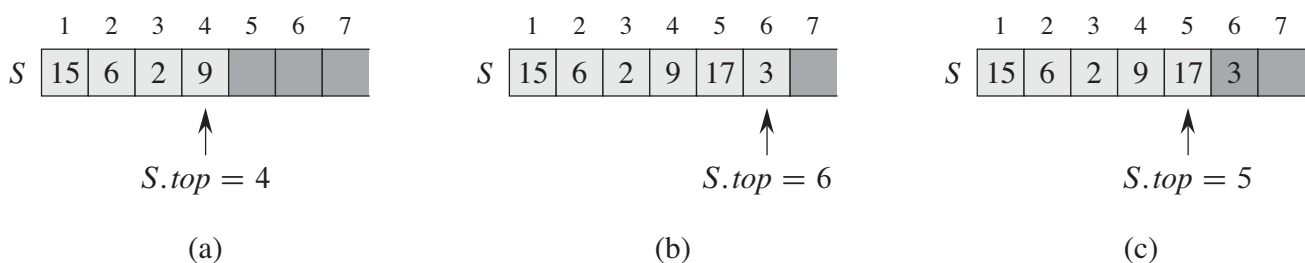


Figura: Ejemplo de implementación de una Pila como un arreglo. Fuente [2]



Definición del Listas enlazadas

Definición

Una lista enlazada es una estructura de datos en donde los elementos se encuentran en orden lineal. La lista enlazada esta constituida por celdas. Cada elemento contiene un elemento de la lista y un apuntador a la siguiente celda. Una lista doblemente enlazada además contiene un apuntador a la celda previa.



Ejemplo de listas enlazadas

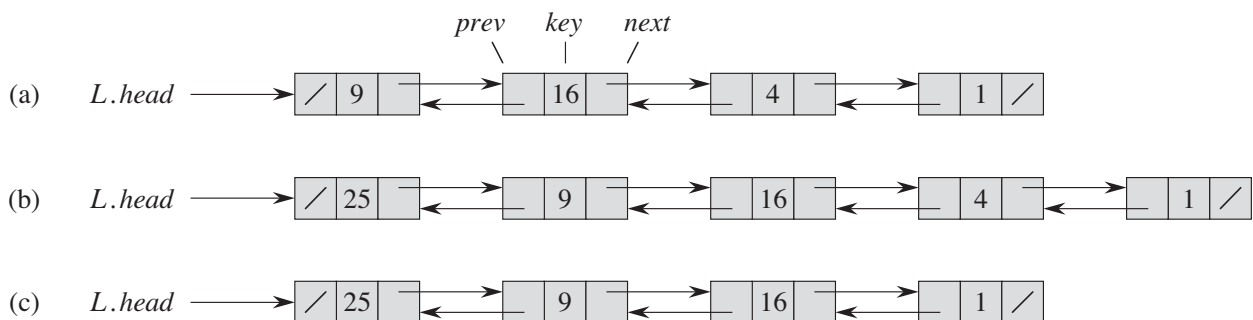


Figura: Ejemplo de lista enlazada implementando la secuencia 1,4,9,16.
Fuente [2]



Implementación de las listas doblemente enlazadas

Función List-Search(L, k)

inicio

```

 $x \leftarrow L.head$  ;
mientras  $x \neq NIL \wedge x.key \neq k$  hacer
     $x \leftarrow x.next$  ;
devolver  $x$ ;

```

Procedimiento List-Insert(L, x)

inicio

```

 $x.next \leftarrow L.head$  ;
si  $L.head \neq NIL$  entonces
     $L.head.prev \leftarrow x$  ;
 $L.head \leftarrow x$  ;
 $x.prev \leftarrow NIL$  ;

```



Continuación de la implementación de las listas doblemente enlazadas

Procedimiento List-Delete(L, x)

inicio





```

si  $x.prev \neq NIL$  entonces
     $x.prev.next \leftarrow x.next$  ;
en otro caso
     $L.head \leftarrow x.next$  ;
si  $x.next \neq NIL$  entonces
     $x.next.prev = x.prev$  ;

```



Referencias

-  A. Aho, J. Hopcroft, and J. Ullman.
Estructuras de Datos y Algoritmos.
Addison-Wesley, 1998.
-  T. Cormen, C. Leirserson, R. Rivest, and C. Stein.
Introduction to Algorithms.
McGraw Hill, 3ra edition, 2009.
-  Barbara Liskov and Stephen Zilles.
Programming with abstract data types.
In *ACM Sigplan Notices*, volume 9, pages 50–59. ACM, 1974.
-  J.N. Ravelo.
Especificacion e implementacion de tipos abstractos de datos.
<https://ldc.usb.ve/~jravelo/docencia/algoritmos/material/tads.pdf>, 2009.

