Escuela de Ingeniería Civil Informática
Facultad de Ingeniería

Estructuras de datos

Capítulo II: Estructuras de datos fundamentales

Fabián Riquelme Csori

fabian.riquelme@uv.cl

2017-II

Index

Elementos de un TDA

Un tipo de datos abstracto (TDA o ADT) está conformado por:

- Una colección de datos.
- Un conjunto de operaciones sobre los datos (o sobre un subconjunto de ellos).
- Un conjunto de axiomas que controlan y restringen el uso de las operaciones.

Elementos de un TDA

Un tipo de datos abstracto (TDA o ADT) está conformado por:

- Una colección de datos.
- Un conjunto de operaciones sobre los datos (o sobre un subconjunto de ellos).
- Un conjunto de axiomas que controlan y restringen el uso de las operaciones.

Los TDA son independientes de la implementación, y permiten acceder a sus operaciones mediante una interfaz.

Elementos de un TDA

Un tipo de datos abstracto (TDA o ADT) está conformado por:

- Una colección de datos.
- Un conjunto de operaciones sobre los datos (o sobre un subconjunto de ellos).
- Un conjunto de axiomas que controlan y restringen el uso de las operaciones.

Los TDA son independientes de la implementación, y permiten acceder a sus operaciones mediante una interfaz.
Una estructura de datos es una implementación de un TDA.
Ej: las clases `vector`, `list`, `stack`, ... de C++ Standard Library.

Notación de un TDA

No existe una notación estándar para describir un TDA.
Una notación posible puede ser:

ADT_name	Breve descripción de la colección de datos.
Operations	Lista de operaciones, con parámetros de entrada entre paréntesis, más breves descripciones.
Axioms	Definidos lo más formalmente posible en términos de las operaciones.

Notación de un TDA

No existe una notación estándar para describir un TDA.
Una notación posible puede ser:

ADT_name	Breve descripción de la colección de datos.
Operations	Lista de operaciones, con parámetros de entrada entre paréntesis, más breves descripciones.
Axioms	Definidos lo más formalmente posible en términos de las operaciones.

- El detalle de las operaciones se puede describir con pseudocódigo aquí o después de los axiomas, indicando la salida (output).

Notación de un TDA

No existe una notación estándar para describir un TDA.
Una notación posible puede ser:

ADT_name	Breve descripción de la colección de datos.
Operations	Lista de operaciones, con parámetros de entrada entre paréntesis, más breves descripciones.
Axioms	Definidos lo más formalmente posible en términos de las operaciones.

- El detalle de las operaciones se puede describir con pseudocódigo aquí o después de los axiomas, indicando la salida (output).
- Los axiomas pueden ser (in)consistentes y (in)completos.
  Un conjunto de axiomas completo puede tener axiomas redundantes.

Notación de un TDA

No existe una notación estándar para describir un TDA.
Una notación posible puede ser:

ADT_name	Breve descripción de la colección de datos.
Operations	Lista de operaciones, con parámetros de entrada entre paréntesis, más breves descripciones.
Axioms	Definidos lo más formalmente posible en términos de las operaciones.

- El detalle de las operaciones se puede describir con pseudocódigo aquí o después de los axiomas, indicando la salida (output).
- Los axiomas pueden ser (in)consistentes y (in)completos.
  Un conjunto de axiomas completo puede tener axiomas redundantes.
- Es bueno incluir además la complejidad computacional de cada operación.

Notación de un TDA

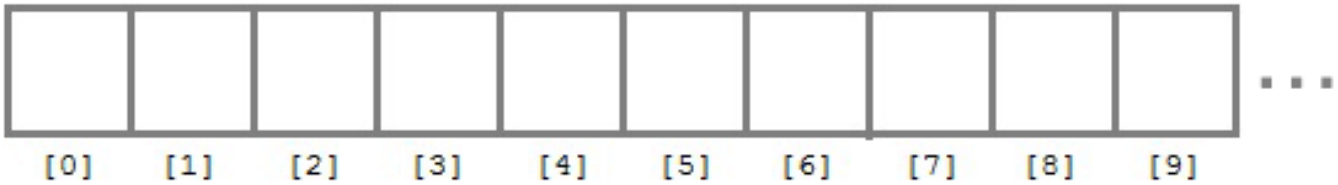
No existe una notación estándar para describir un TDA.
Una notación posible puede ser:

ADT_name	Breve descripción de la colección de datos.
Operations	Lista de operaciones, con parámetros de entrada entre paréntesis, más breves descripciones.
Axioms	Definidos lo más formalmente posible en términos de las operaciones.

- El detalle de las operaciones se puede describir con pseudocódigo aquí o después de los axiomas, indicando la salida (output).
- Los axiomas pueden ser (in)consistentes y (in)completos.
  Un conjunto de axiomas completo puede tener axiomas redundantes.
- Es bueno incluir además la complejidad computacional de cada operación.
- En adelante omitiremos operaciones de constructores y destructores.

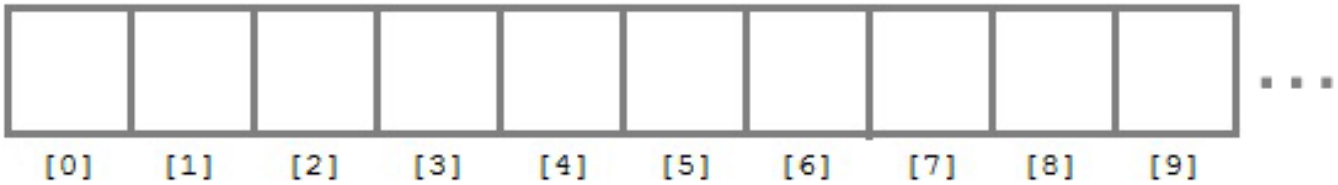
Vector

- A diferencia de un `array`, posee un tamaño variable.
- A cambio, su rendimiento es ligeramente inferior (punteros).
- Aún así, salvo para arreglos muy pequeños cuyo contenido debe modificarse constantemente, es preferible usar una estructura de datos `vector` por sobre los arreglos.



Vector

- A diferencia de un `array`, posee un tamaño variable.
- A cambio, su rendimiento es ligeramente inferior (punteros).
- Aún así, salvo para arreglos muy pequeños cuyo contenido debe modificarse constantemente, es preferible usar una estructura de datos `vector` por sobre los arreglos.



Nota: Python al definir "`var=[a,b]`" en realidad usa su clase `list`. Para este lenguaje existe el paquete NumPy que utiliza arreglos multi-dimensionales eficientes.

Vector: TDA

vector	Sequential, random-access, variable-size elements container (of the same type).
operations	
size()	How many elements are in the collection?
isEmpty()	Is the collection empty?
contains(element)	Does the collection contain the given element?
elementAt(index)	Access the element at the given index.
clear()	Make the collection empty.
append(element)	Add a new element to the end of the collection.
insertAt(index, element)	Insert a new element at the given index.
remove(element)	Remove the given element from the collection.
removeAt(index)	Remove the element at the given index.
replace(index, element)	Replace the element at the given index with a new element.

Vector: TDA

vector	Sequential, random-access, variable-size elements container (of the same type).
operations	
size()	How many elements are in the collection?
isEmpty()	Is the collection empty?
contains(element)	Does the collection contain the given element?
elementAt(index)	Access the element at the given index.
clear()	Make the collection empty.
append(element)	Add a new element to the end of the collection.
insertAt(index, element)	Insert a new element at the given index.
remove(element)	Remove the given element from the collection.
removeAt(index)	Remove the element at the given index.
replace(index, element)	Replace the element at the given index with a new element.

- ¿Cuál sería el output de cada operación?

Vector: TDA

vector	Sequential, random-access, variable-size elements container (of the same type).
operations	
size()	How many elements are in the collection?
isEmpty()	Is the collection empty?
contains(element)	Does the collection contain the given element?
elementAt(index)	Access the element at the given index.
clear()	Make the collection empty.
append(element)	Add a new element to the end of the collection.
insertAt(index, element)	Insert a new element at the given index.
remove(element)	Remove the given element from the collection.
removeAt(index)	Remove the element at the given index.
replace(index, element)	Replace the element at the given index with a new element.

- ¿Cuál sería el output de cada operación?
- ¿Qué operadores son de capacidad, acceso, modificadores, comparadores?

Vector: TDA

Sea n el tamaño de un vector y n' su tamaño luego de modificarse:

Operación	**Complejidad**
size()	$O(1)$
isEmpty()	$O(1)$
contains(element)	$O(n)$
elementAt(index)	$O(1)$
clear()	$O(n)$
append(element)	$O(1)$; $O(n')$ si hay reasignación de memoria
insertAt(index, element)	$O(m) \leq O(n)$; m son los elementos desde index+1
remove(element)	$O(n)$ en el peor caso
removeAt(index)	$O(m) \leq O(n)$; m son elementos después de los borrados
replace(index, element)	$O(1)$

Vector: TDA

Sea n el tamaño de un vector y n' su tamaño luego de modificarse:

Operación	**Complejidad**
size()	$O(1)$
isEmpty()	$O(1)$
contains(element)	$O(n)$
elementAt(index)	$O(1)$
clear()	$O(n)$
append(element)	$O(1)$; $O(n')$ si hay reasignación de memoria
insertAt(index, element)	$O(m) \leq O(n)$; m son los elementos desde index+1
remove(element)	$O(n)$ en el peor caso
removeAt(index)	$O(m) \leq O(n)$; m son elementos después de los borrados
replace(index, element)	$O(1)$

- ¿Qué complejidad tendría una operación indexOf(element), que retorna el índice de un elemento dado?

Vector: TDA

axioms

For any vector V, element a, index i, integer n:

1. V.size(), V.isEmpty(), V.contains(a), V.clear() and V.append(a) are always defined
2. (V.append(a)).isEmpty() = false
3. (V.append(a)).size() = V.size()+1
4. (V.insertAt(i,a)).size() = V.size()+1
5. (V.remove(a)).size() $\leq$ V.size()
6. (V.removeAt(i)).size() = V.size()-1
7. (V.insertAt(i,a)).removeAt(i) = V
8. (V.insertAt(i,a)).elementAt(i) = a

Vector: TDA

axioms

For any vector V, element a, index i, integer n:

1. V.size(), V.isEmpty(), V.contains(a), V.clear() and V.append(a) are always defined
2. (V.append(a)).isEmpty() = false
3. (V.append(a)).size() = V.size()+1
4. (V.insertAt(i,a)).size() = V.size()+1
5. (V.remove(a)).size() $\leq$ V.size()
6. (V.removeAt(i)).size() = V.size()-1
7. (V.insertAt(i,a)).removeAt(i) = V
8. (V.insertAt(i,a)).elementAt(i) = a

- Este conjunto de axiomas es consistente. ¿Será completo? ¿será redundante?

Vector: TDA

axioms

For any vector V, element a, index i, integer n:

1. V.size(), V.isEmpty(), V.contains(a), V.clear() and V.append(a) are always defined
2. (V.append(a)).isEmpty() = false
3. (V.append(a)).size() = V.size()+1
4. (V.insertAt(i,a)).size() = V.size()+1
5. (V.remove(a)).size() $\leq$ V.size()
6. (V.removeAt(i)).size() = V.size()-1
7. (V.insertAt(i,a)).removeAt(i) = V
8. (V.insertAt(i,a)).elementAt(i) = a

- Este conjunto de axiomas es consistente. ¿Será completo? ¿será redundante?
- Verificar qué operaciones están implementadas en la estructura de datos `vector` de la C++ Standard Library, y cuáles no:
  `http://www.cplusplus.com/reference/vector/vector/`

Vector: TDA

axioms

For any vector V, element a, index i, integer n:

1. V.size(), V.isEmpty(), V.contains(a), V.clear() and V.append(a) are always defined
2. (V.append(a)).isEmpty() = false
3. (V.append(a)).size() = V.size()+1
4. (V.insertAt(i,a)).size() = V.size()+1
5. (V.remove(a)).size() $\leq$ V.size()
6. (V.removeAt(i)).size() = V.size()-1
7. (V.insertAt(i,a)).removeAt(i) = V
8. (V.insertAt(i,a)).elementAt(i) = a

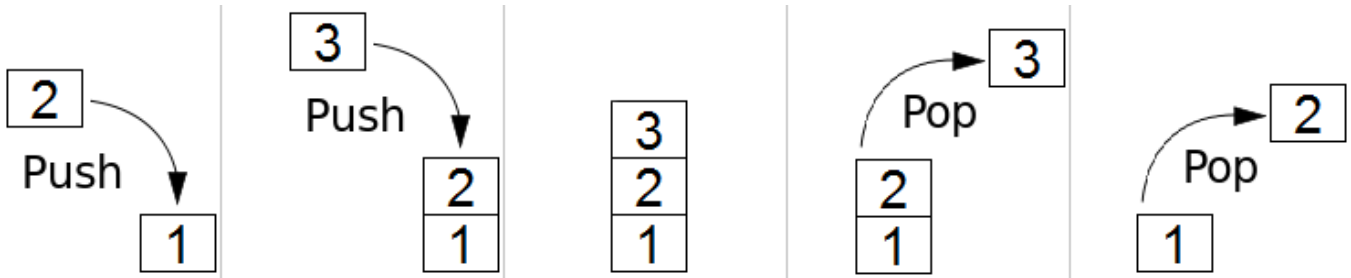
- Este conjunto de axiomas es consistente. ¿Será completo? ¿será redundante?
- Verificar qué operaciones están implementadas en la estructura de datos `vector` de la C++ Standard Library, y cuáles no:
  `http://www.cplusplus.com/reference/vector/vector/`
- Ejercicio: Definir nuevos axiomas, e implementar las operaciones que falten.

Pila/Stack

- Como el vector, es una estructura secuencial y de tamaño variable, pero donde solo se puede agregar o quitar elementos desde uno de sus extremos.

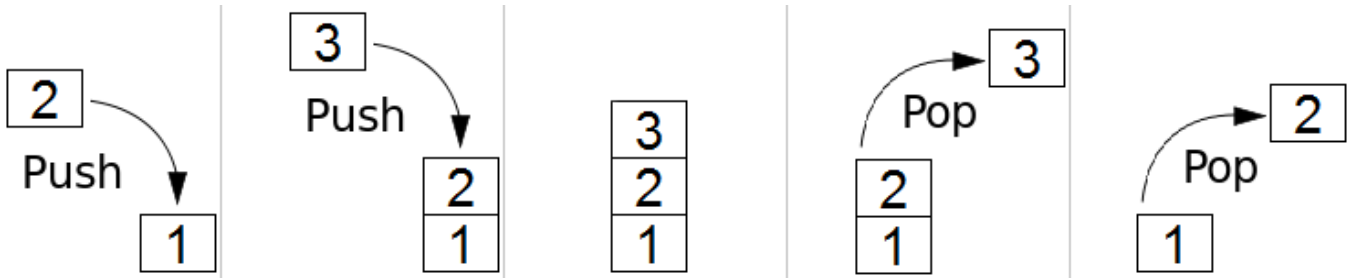
Pila/Stack

- Como el vector, es una estructura secuencial y de tamaño variable, pero donde solo se puede agregar o quitar elementos desde uno de sus extremos.
- Los elementos se eliminan en el orden inverso al que se insertaron $\rightarrow$ estructura LIFO (last-in, first-out).



Pila/Stack

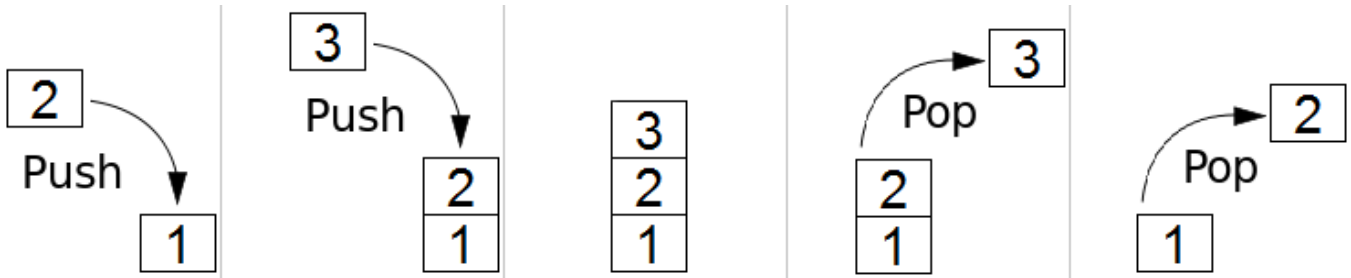
- Como el vector, es una estructura secuencial y de tamaño variable, pero donde solo se puede agregar o quitar elementos desde uno de sus extremos.
- Los elementos se eliminan en el orden inverso al que se insertaron $\rightarrow$ estructura LIFO (last-in, first-out).



- ¿Aplicaciones?

Pila/Stack

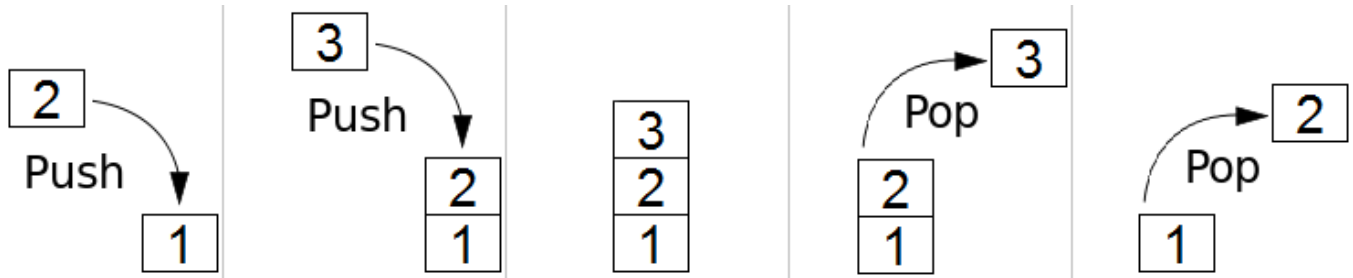
- Como el vector, es una estructura secuencial y de tamaño variable, pero donde solo se puede agregar o quitar elementos desde uno de sus extremos.
- Los elementos se eliminan en el orden inverso al que se insertaron $\rightarrow$ estructura LIFO (last-in, first-out).



- ¿Aplicaciones? Cadenas de producción, análisis sintáctico, programación lógica, etc.

Pila/Stack

- Como el vector, es una estructura secuencial y de tamaño variable, pero donde solo se puede agregar o quitar elementos desde uno de sus extremos.
- Los elementos se eliminan en el orden inverso al que se insertaron $\rightarrow$ estructura LIFO (last-in, first-out).



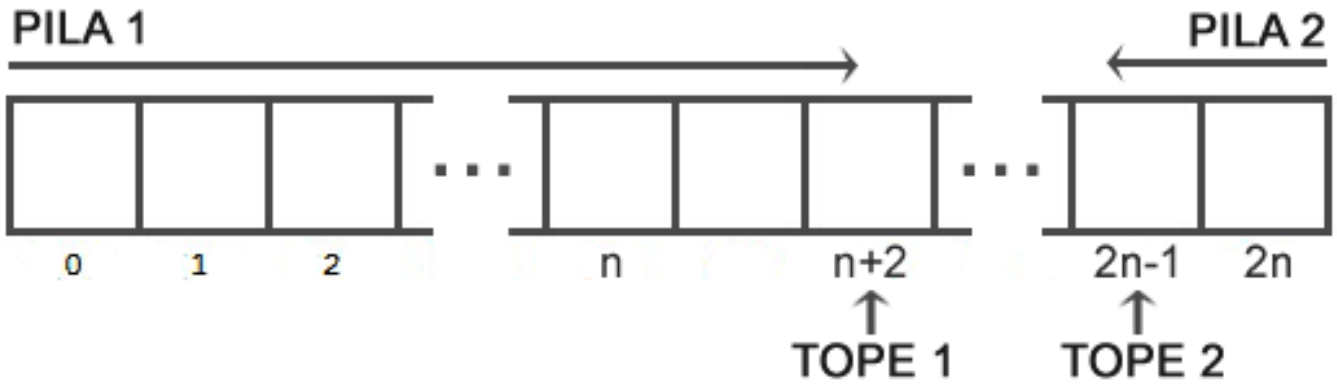
- ¿Aplicaciones? Cadenas de producción, análisis sintáctico, programación lógica, etc.
- El tope es el único elemento visible, i.e., el último agregado.

Pila/Stack: errores usuales

- Una pila, como un vector, puede definirse con un tamaño máximo. Si se agregan más elementos que este máximo, se produce un error conocido como desbordamiento u overflow.

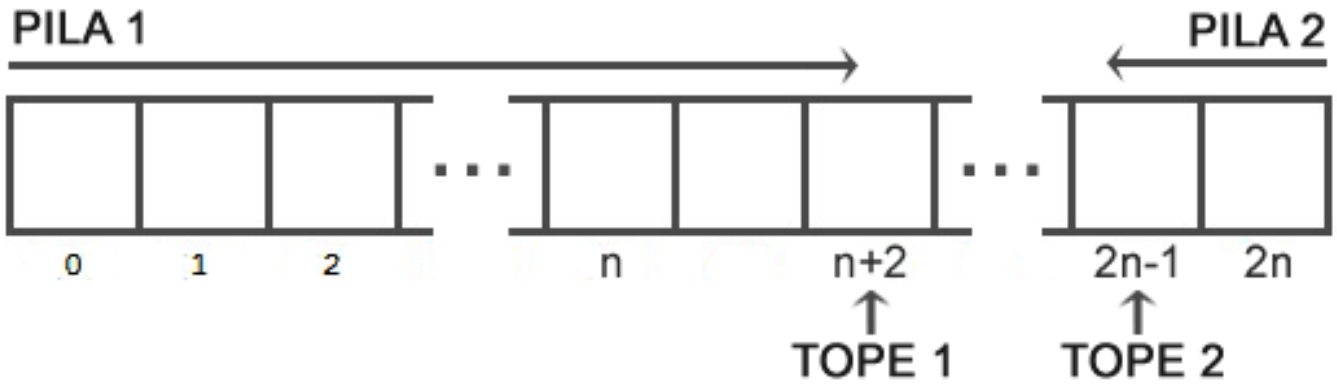
Pila/Stack: errores usuales

- Una pila, como un vector, puede definirse con un tamaño máximo. Si se agregan más elementos que este máximo, se produce un error conocido como desbordamiento u overflow.
  - Para evitar overflow sin usar un máximo muy grande se pueden definir pilas con espacio compartido:



Pila/Stack: errores usuales

- Una pila, como un vector, puede definirse con un tamaño máximo. Si se agregan más elementos que este máximo, se produce un error conocido como desbordamiento u overflow.
  - Para evitar overflow sin usar un máximo muy grande se pueden definir pilas con espacio compartido:



- Al contrario, si se intenta quitar un elemento de una pila vacía, se produce un error de subdesbordamiento o underflow.

Pila/Stack: TDA

stack	LIFO, variable-size elements container (of the same type).
operations	
size()	How many elements are in the stack?
isEmpty()	Is the stack empty?
top()	Access the element at the top of the stack.
push(element)	Add a new element at the top of the stack.
pop()	Remove the top element of the stack.

Pila/Stack: TDA

stack	LIFO, variable-size elements container (of the same type).
operations	
size()	How many elements are in the stack?
isEmpty()	Is the stack empty?
top()	Access the element at the top of the stack.
push(element)	Add a new element at the top of the stack.
pop()	Remove the top element of the stack.

- ¿Cuál sería el output de cada operación?
- ¿Qué operadores son de capacidad, acceso, modificadores, comparadores?

Pila/Stack: TDA

Operación	Complejidad
size()	$O(1)$
isEmpty()	$O(1)$
top()	$O(1)$
push(element)	$O(1)$
pop()	$O(1)$

Pila/Stack: TDA

axioms

For any stack S, element a:

1. S.size(), S.IsEmpty(), S.push(a) are always defined
2. If (S.isEmpty() = true) then (S.top( ) = error)
3. If (S.isEmpty() = true) then (S.pop( ) = error)
4. (S.push(a)).isEmpty() = false
5. (S.push(a)).top( ) = a
6. (S.push(a)).pop( ) = S

Pila/Stack: TDA

axioms

For any stack S, element a:

1. S.size(), S.IsEmpty(), S.push(a) are always defined
2. If (S.isEmpty() = true) then (S.top( ) = error)
3. If (S.isEmpty() = true) then (S.pop( ) = error)
4. (S.push(a)).isEmpty() = false
5. (S.push(a)).top( ) = a
6. (S.push(a)).pop( ) = S

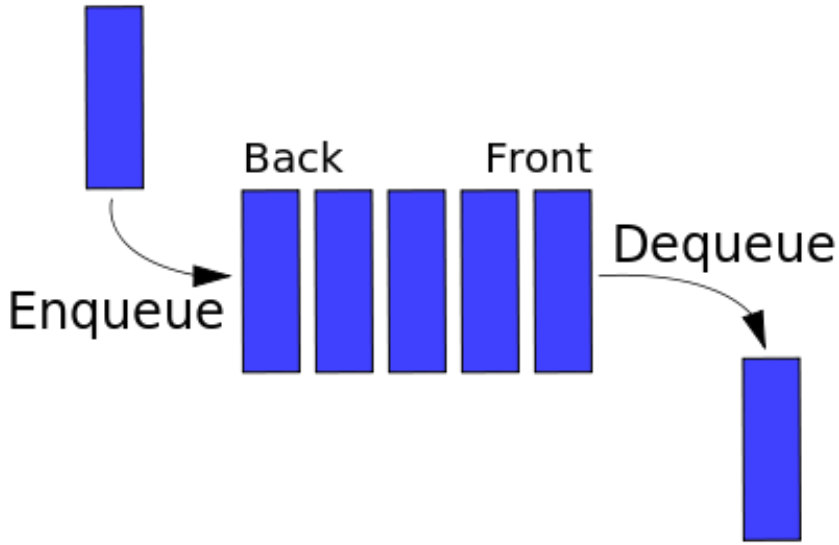
- Verificar operaciones implementadas en estructura de datos `stack` de C++ Standard Library: `http://www.cplusplus.com/reference/stack/stack/`
- Ejercicio: Definir nuevos axiomas (puede utilizar operaciones de constructores).

Cola/Fila/Queue

- Como las anteriores, es una estructura secuencial y de tamaño variable, pero donde solo se puede agregar por un extremo y quitar por el otro.

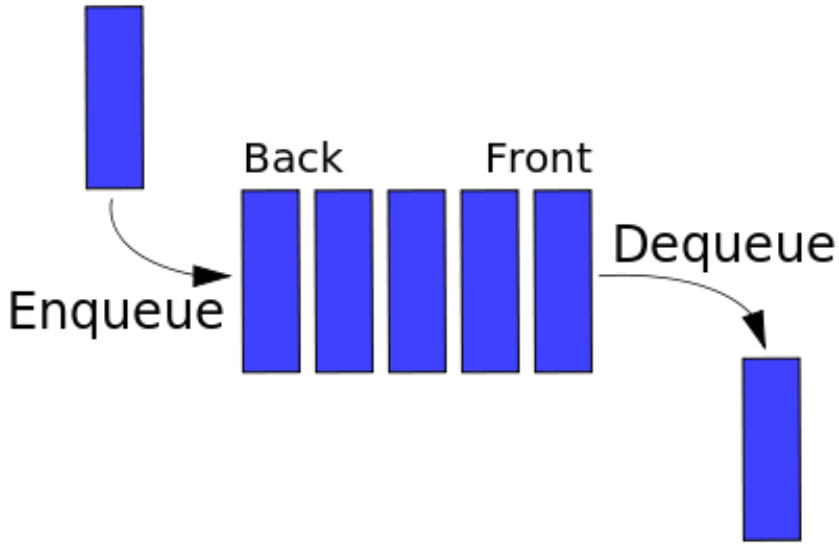
Cola/Fila/Queue

- Como las anteriores, es una estructura secuencial y de tamaño variable, pero donde solo se puede agregar por un extremo y quitar por el otro.
- Los elementos se eliminan en el mismo orden en el que se insertaron $\rightarrow$ estructura FIFO (first-in, first-out).



Cola/Fila/Queue

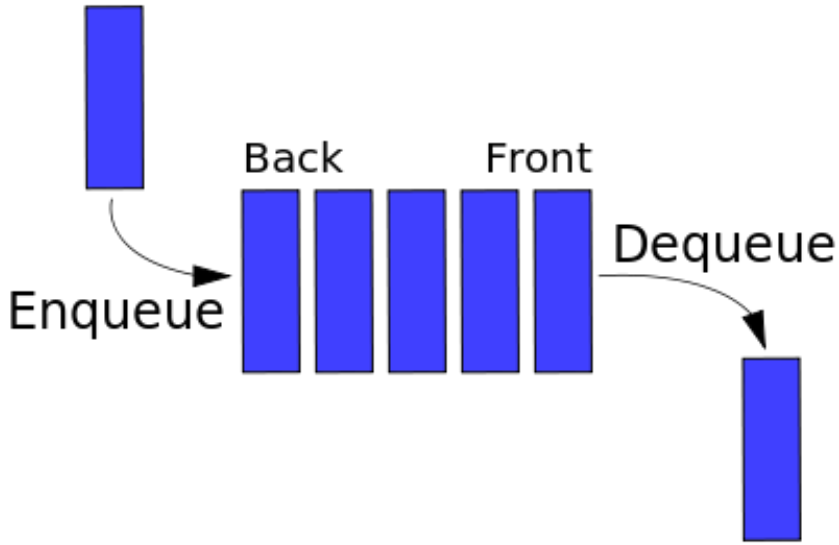
- Como las anteriores, es una estructura secuencial y de tamaño variable, pero donde solo se puede agregar por un extremo y quitar por el otro.
- Los elementos se eliminan en el mismo orden en el que se insertaron $\rightarrow$ estructura FIFO (first-in, first-out).



- ¿Aplicaciones?

Cola/Fila/Queue

- Como las anteriores, es una estructura secuencial y de tamaño variable, pero donde solo se puede agregar por un extremo y quitar por el otro.
- Los elementos se eliminan en el mismo orden en el que se insertaron $\rightarrow$ estructura FIFO (first-in, first-out).



- ¿Aplicaciones? Cadenas de producción, listas de espera en sistemas operativos, etc.

Cola/Fila/Queue: TDA

queue	FIFO, variable-size elements container (of the same type).
operations	
size()	How many elements are in the queue?
isEmpty()	Is the queue empty?
front()	Access the element at the front of the queue.
push(element)	Add a new element at the back of the queue.
pop()	Remove the element at the front of the queue.

Cola/Fila/Queue: TDA

queue	FIFO, variable-size elements container (of the same type).
operations	
size()	How many elements are in the queue?
isEmpty()	Is the queue empty?
front()	Access the element at the front of the queue.
push(element)	Add a new element at the back of the queue.
pop()	Remove the element at the front of the queue.

- ¿Cuál sería el output de cada operación?
- ¿Qué operadores son de capacidad, acceso, modificadores, comparadores?

Cola/Fila/Queue: TDA

Operación	**Complejidad**
size()	$O(1)$
isEmpty()	$O(1)$
front()	$O(1)$
push(element)	$O(1)$
pop()	$O(1)$

Cola/Fila/Queue: TDA

---

axioms

For any queue Q, element a:

1. Q.size(), Q.IsEmpty(), Q.push(a) are always defined
2. (Q.push(a)).isEmpty() = false
3. If (Q.isEmpty() = true) then (Q.front( ) = error)
4. If (Q.isEmpty() = true) then (Q.pop( ) = error)
5. if (Q.isEmpty() = true) then (Q.push(a)).front( ) = a
6. if (Q.isEmpty() = true) then (Q.push(a)).pop( ) = Q

Cola/Fila/Queue: TDA

---

axioms

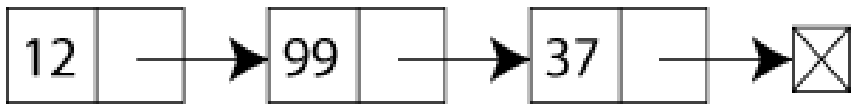
For any queue Q, element a:

1. Q.size(), Q.IsEmpty(), Q.push(a) are always defined
2. (Q.push(a)).isEmpty() = false
3. If (Q.isEmpty() = true) then (Q.front( ) = error)
4. If (Q.isEmpty() = true) then (Q.pop( ) = error)
5. if (Q.isEmpty() = true) then (Q.push(a)).front( ) = a
6. if (Q.isEmpty() = true) then (Q.push(a)).pop( ) = Q

- Verificar operaciones implementadas en estructura de datos queue de C++ Standard Library: `http://www.cplusplus.com/reference/queue/queue/`
- Ejercicio: Definir nuevos axiomas (puede utilizar operaciones de constructores).

Lista/List

- Como las estructuras anteriores (vectores, pilas y colas) es un contenedor secuencial y de tamaño variable.
- Sus implementaciones utilizan uno o dos "cabezales" móviles, para recorrerla de izquierda a derecha.
  - Los vectores no usan cabezales (acceso directo/aleatorio).
  - Las pilas tienen solo un cabezal fijo en el tope.
  - Las colas tienen un cabezal fijo en el back y otro en el front $\rightarrow$ este es el caso de las listas.



Lista/List

- Ventajas
  - Son más eficientes en operaciones de inserción, extracción y mover elementos $\rightarrow$ de $O(n)$ en peor caso se pasa a $O(1)$.
  - Preferibles en algoritmos que usan mucho estas operaciones, como algoritmos de ordenamiento.

Lista/List

- Ventajas
  - Son más eficientes en operaciones de inserción, extracción y mover elementos $\rightarrow$ de $O(n)$ en peor caso se pasa a $O(1)$.
  - Preferibles en algoritmos que usan mucho estas operaciones, como algoritmos de ordenamiento.
- Desventajas
  - No permiten acceso directo a sus elementos (para acceder a un elemento, hay que recorrer la lista desde la ubicación actual del cabezal) $\rightarrow$ de $O(1)$ se pasa a $O(n)$ en el peor caso.
  - Usan un poco más de memoria para mantener a los elementos asociados (punteros).
  - No recomendadas si serán listas muy grandes con elementos muy pequeños.

Lista/List: TDA

list	Sequential, variable-size elements container (of same type).
operations	
size()	How many elements are in the list?
isEmpty()	Is the list empty?
front()	Access the element at the front of the list.
back()	Access the element at the back of the list.
get(position)	Access the element at the given position.
clear()	Make the list empty.
push_back(element)	Add a new element to the back of the list.
push_front(element)	Add a new element to the front of the list.
insert(position, element)	Insert a new element at the given position in the list.
pop_back()	Remove the element at the back of the list.
pop_front()	Remove the element at the front of the list.
remove(element)	Remove the given element from the coalition.
remove(position)	Remove the element at the given position.

Lista/List: TDA

list	Sequential, variable-size elements container (of same type).
operations	
size()	How many elements are in the list?
isEmpty()	Is the list empty?
front()	Access the element at the front of the list.
back()	Access the element at the back of the list.
get(position)	Access the element at the given position.
clear()	Make the list empty.
push_back(element)	Add a new element to the back of the list.
push_front(element)	Add a new element to the front of the list.
insert(position, element)	Insert a new element at the given position in the list.
pop_back()	Remove the element at the back of the list.
pop_front()	Remove the element at the front of the list.
remove(element)	Remove the given element from the coalition.
remove(position)	Remove the element at the given position.

- ¿Cuál sería el output de cada operación?
- ¿Qué operadores son de capacidad, acceso, modificadores, comparadores?

List: TDA

Sea n el tamaño de un vector:

Operación	Complejidad
size()	$O(n)$
isEmpty()	$O(1)$
front()	$O(1)$
back()	$O(1)$ (con dos cabezales)
get(position)	$O(n)$
clear()	$O(n)$
push_back(element)	$O(1)$
push_front(element)	$O(1)$
insert(position, element)	$O(1)$ + search time
pop_back()	$O(1)$
pop_front()	$O(1)$
remove(element)	$O(n)$ en el peor caso
remove(position)	$O(1)$ + search time

Lista/List: TDA

axioms

For any list L, element a, position p:

1. L.size(), L.isEmpty(), L.clear(), L.push_back(a), L.push_front(a) are always defined
2. (L.push_back(a)).IsEmpty() = false
3. (L.insert(p,a)).IsEmpty() = false
4. (L.push_back(a)).size() = L.size()+1
5. (L.insert(p,a)).size() = L.size()+1
6. (L.pop_back()).size() = L.size()-1
7. (L.remove(a)).IsEmpty() $\leq$ L.size()
8. If (L.isEmpty() = true) then (L.back() = error)
9. If (L.isEmpty() = true) then (L.pop_back() = error)
10. if (L.isEmpty() = true) then (L.push_back(a)).back() = a
11. if (L.isEmpty() = true) then (L.push_back(a)).pop_back() = L
12. (L.insert(p,a)).remove(p) = L
13. L.get(p) = (L.insert(p,a)).get(p+1)
14. L.get(p+1) = (L.remove(p)).get(p)

idem reemplazando 'back' por 'front' donde corresponda.

Lista/List: TDA

axioms

For any list L, element a, position p:

1. L.size(), L.isEmpty(), L.clear(), L.push_back(a), L.push_front(a) are always defined
2. (L.push_back(a)).IsEmpty() = false
3. (L.insert(p,a)).IsEmpty() = false
4. (L.push_back(a)).size() = L.size()+1
5. (L.insert(p,a)).size() = L.size()+1
6. (L.pop_back()).size() = L.size()-1
7. (L.remove(a)).IsEmpty() $\leq$ L.size()
8. If (L.isEmpty() = true) then (L.back() = error)
9. If (L.isEmpty() = true) then (L.pop_back() = error)
10. if (L.isEmpty() = true) then (L.push_back(a)).back() = a
11. if (L.isEmpty() = true) then (L.push_back(a)).pop_back() = L
12. (L.insert(p,a)).remove(p) = L
13. L.get(p) = (L.insert(p,a)).get(p+1)
14. L.get(p+1) = (L.remove(p)).get(p)

idem reemplazando 'back' por 'front' donde corresponda.

- Verificar operaciones implementadas en estructura de datos `list` de C++ Standard Library: `http://www.cplusplus.com/reference/list/list/`
- Ejercicio: Definir nuevos axiomas.

Listas enlazadas

- Una lista enlazada es una estructura de datos, i.e., una implementación posible del TDA lista.

Listas enlazadas

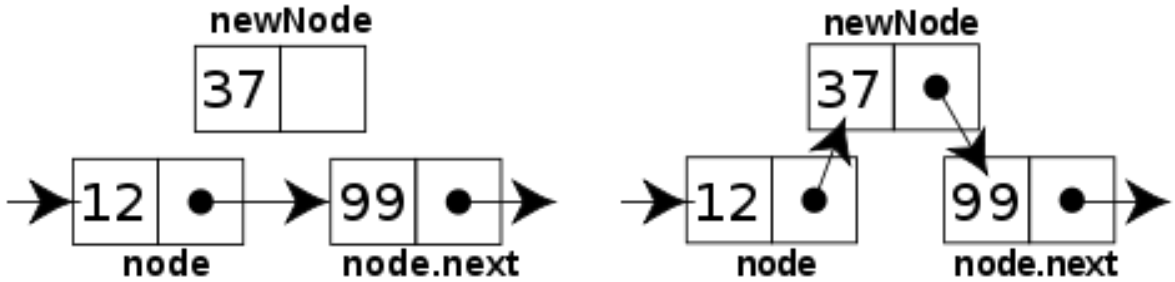
- Una lista enlazada es una estructura de datos, i.e., una implementación posible del TDA lista.
- Ventajas
  - Son estructuras dinámicas, i.e., pueden ocupar y desechar memoria en tiempo de ejecución.
  - Inserción y eliminación están implementadas inteligentemente, y se pueden hacer en cualquier parte de la lista.
  - Otras estructuras dinámicas como pilas y colas pueden implementarse como listas enlazadas.
  - No necesitan definir un tamaño inicial.

Listas enlazadas

- Una lista enlazada es una estructura de datos, i.e., una implementación posible del TDA lista.
- Ventajas
  - Son estructuras dinámicas, i.e., pueden ocupar y desechar memoria en tiempo de ejecución.
  - Inserción y eliminación están implementadas inteligentemente, y se pueden hacer en cualquier parte de la lista.
  - Otras estructuras dinámicas como pilas y colas pueden implementarse como listas enlazadas.
  - No necesitan definir un tamaño inicial.
- Desventajas
  - Mayor uso de memoria que arreglos y vectores (punteros).
  - Los nodos deben leerse en orden, secuencialmente.
  - Pequeño costo asociado para acceder al contenido de un nodo.

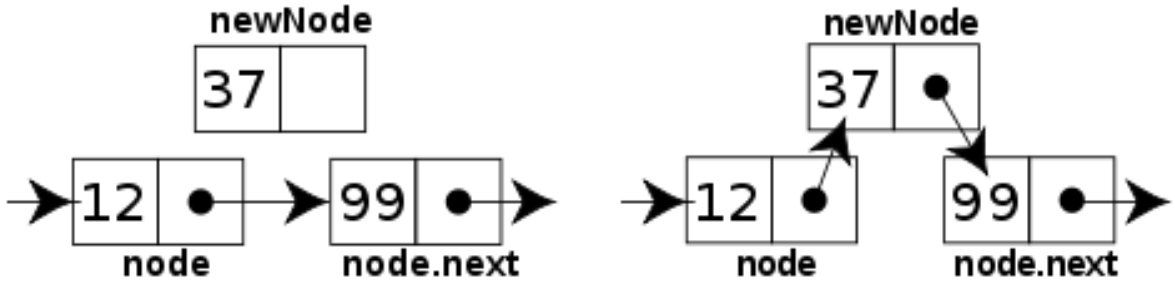
Operaciones

- Insertion

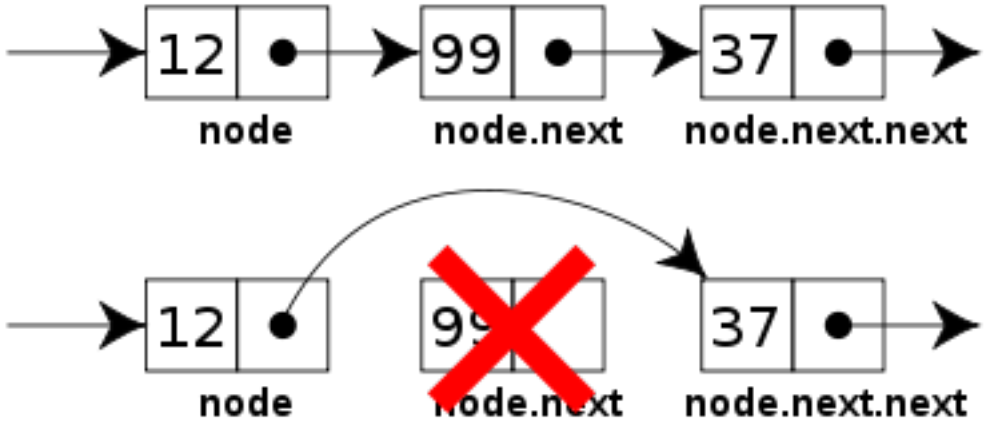


Operaciones

- Insertion



- Deletion



Ejercicio: Listas enlazadas

- Implemente una lista enlazada que reciba como entrada una lista de las notas de un curso de 10 alumnos: 10 números decimales (puede restringirse a un decimal) entre 1.0 y 7.0. Idealmente use C o C++, pero sin la STL o C++ Standard Library.
  1. Reordene dicha secuencia de números de menor a mayor.
  2. Copie aquellos valores de la lista que son mayores que 4.0 en otra lista enlazada, y elimínelos de la lista original.
  3. Añada entre el quinto y el sexto nodo de la primera lista enlazada un nodo adicional, cuyo valor corresponda al promedio de notas en dicha lista.
- Repita el ejercicio utilizando la clase List de C++.

Ejercicio: Vectores, Pilas y Colas

- Utilizando la filosofía del TDA Vector, defina un TDA Matriz.
- Implemente dos pilas con memoria compartida.
- A partir de la estructura de datos Fila de algún lenguaje de programación, implemente dos operaciones:
  - front2(): hace lo mismo que front(), pero no accede al primero, sino al segundo elemento encolado.
  - pop2(): hace lo mismo que pop(), pero no elimina el primero, sino el segundo elemento encolado.

Bibliografía recomendada

- Weiss, M., Estructura de datos y algoritmos, Addison-Wesley, 1995.
- Aho, Hopcroft y Ullman, Estructuras de datos y algoritmos, Addison-Wesley, 1988.

Recursos

- Wikimedia Commons.
- `http://www.cplusplus.com`