Tema 5
Tipos Abstractos de Datos Lineales

Pablo Sánchez

Dpto. Matemáticas, Estadística y Computación
Universidad de Cantabria
Santander (Cantabria, España)
p.sanchez@unican.es

Objetivos

Objetivos

1. Conocer los tipos abstractos de datos lineales más populares.
2. Saber usar los tipos abstractos de datos lineales más populares.
3. Saber escoger, adoptando las técnicas más adecuadas de acuerdo a criterios de eficiencia temporal y espacial, la implementación más adecuada para un TAD lineal.
4. Ser capaz de usar las bibliotecas de TADs lineales proporcionadas por los lenguajes de programación actuales.

Bibliografía Básica

Lewis, J. and Chase, J. (2006).
Estructuras de Datos con Java : Diseño de Estructuras y Algoritmos.
Peason Education, 2 edition.

Meyer, B. (2000).
Construcción de Software Orientada a Objetos.
Prentice Hall.

Ricardo Peña (2005).
Diseño de Programas: Formalismo y Abstracción.
Pearson Educacion, 3 edition.

Sahni, S. (2000).
Data Structures, Algorithms, And Applications In Java.
McGraw-Hill.

Índice

Concepto de Tipo Abstracto de Datos Lineal

Concepto de Tipo Abstracto de Datos lineal

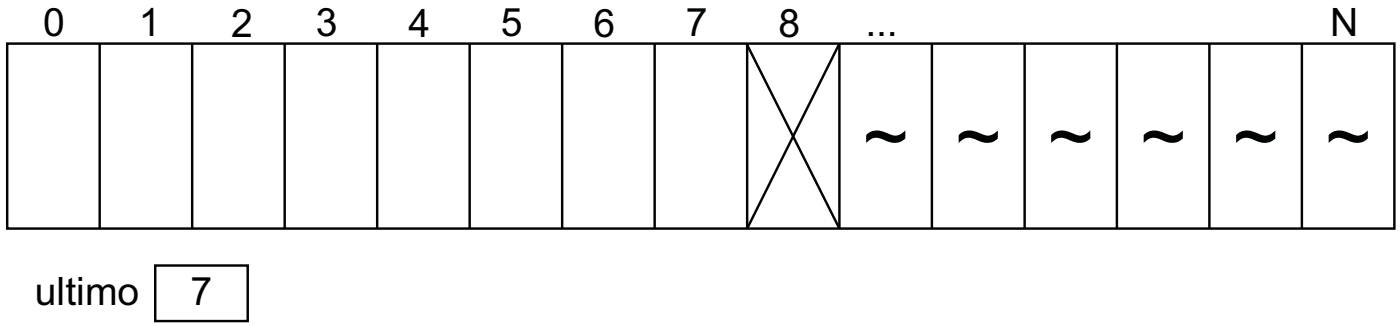
Un tipo abstracto de datos lineal es un tipo abstracto de datos que se puede representar como una secuencia de elementos $\{a_1, \ldots, a_i, \ldots, a_j, \ldots, a_n\}$, donde un conjunto finito (y corto), de elementos son elementos destacados y la única relación entre dos elementos a_i e a_j, si la hubiere, es la de ser el *antecesor de* o el *sucesor de*.

Tipos Abstractos de Datos Lineales más Populares

1. Bolsas (Multiconjuntos)
2. Conjuntos
3. Secuencias (Listas)
4. Secuencias con Prioridad.
5. Conjuntos Ordenados (Secuencias sin repetición).
6. Conjuntos Ordenados con Prioridad.
7. Vectores y Matrices.
8. Pilas.
9. Colas.
10. Colas con Prioridad.
11. Tablas, Mapas, Aplicaciones o Diccionarios.

Índice

Estructuras con Memoria Estática



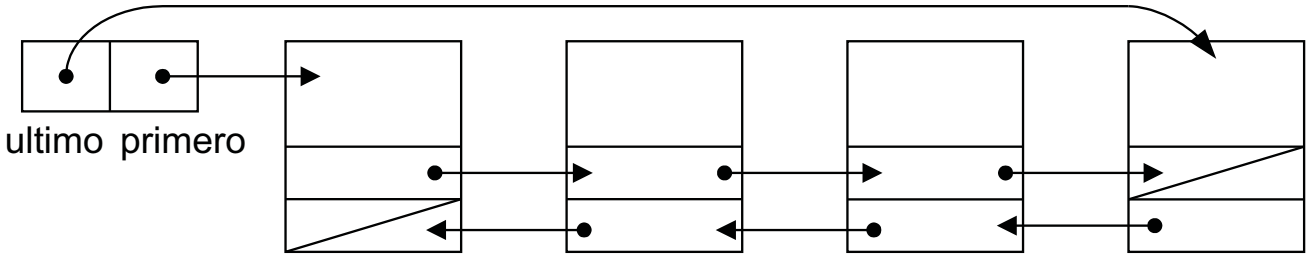
Ventajas

1. Eficientes al estar todos los elementos contiguos en memoria.
2. Fácil acceso posicional.

Inconvenientes

1. Tamaño del vector puede estar determinado en tiempo de compilación.
2. Si se puede cambiar de tamaño, el redimensionamiento suele ser costoso.
3. Desperdicia memoria para estructuras poco llenas.
4. Desplazar subvectores dentro del vector es costoso.

Estructuras con Memoria Dinámica



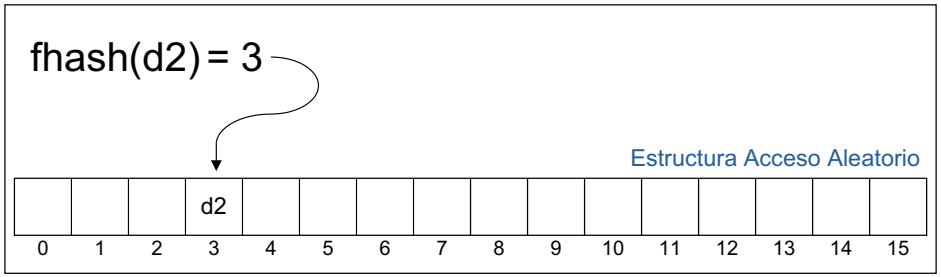
Ventajas

1. Estructuras no acotadas de fácil crecimiento.
2. Sólo se consume la memoria que realmente se usa.
3. Desplazar subestructuras es $\mathcal{O}(1)$.

Inconvenientes

1. Menos eficientes al estar los elementos dispersos en memoria.
2. Acceso posicional más complejo.
3. Puede que tengamos que gestionar la liberación de memoria.

Tablas de Dispersión



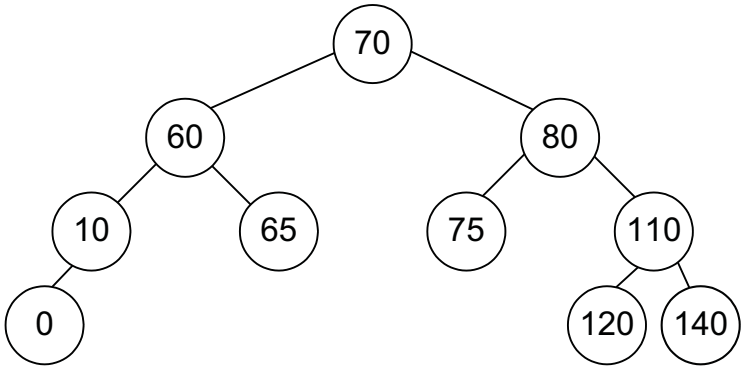
Ventajas

1. Tiempos de búsqueda de elementos de $\mathcal{O}(\approx 1)$.

Inconvenientes

1. Desperdiciar algo de memoria.
2. Ordenar los elementos es algo más complejo y costoso ($\mathcal{O}(n \log n)$).
3. Necesita de una buena función de dispersión.
4. Es una estructura acotada de costosos redimensionamientos.
5. Acceso posicional complejo de mantener.

Árboles Autobalanceados



Ventajas

1. Tiempos de búsqueda de elementos de $\mathcal{O}(\log(n))$.
2. Se pueden ordenar los elementos en $\mathcal{O}(n)$.
3. Estructura no acotada, puede crecer fácilmente en tamaño.
4. No desperdicia memoria.

Inconvenientes

1. Los elementos han de ser comparables por igualdad y orden.

Árboles AVL vs Árboles Rojinegros

	AVL	Rojinegro
Altura	$\mathcal{O}(1{,}44\log(n))$	$\mathcal{O}(2\log(n))$
Búsqueda	$\mathcal{O}(1{,}44\log(n))$	$\mathcal{O}(2\log(n))$
Inserción	$\mathcal{O}(2{,}88\log(n))$	$\mathcal{O}(2\log(n))$
Eliminación	$\mathcal{O}(2{,}88\log(n))$	$\mathcal{O}(2\log(n))$

Índice

Clasificación de Colecciones

	Repetidos	**No Repetidos**
Pos. Relevante	Secuencia	Conjunto Ordenado
Pos. No Relevante	Bolsa	Conjunto

- ¡OJO! Ordenado aquí simplemente significa que el orden en el que aparecen los elementos en la secuencia importan (en inglés, ordered).
- Cuando los elementos estén dispuestos en la colección de acuerdo a alguna relación de orden especial (e.g. lexicográfico) diremos que están *dispuestos u ordenados por prioridad* (en inglés, sorted) o con *orden natural*.

Bolsas

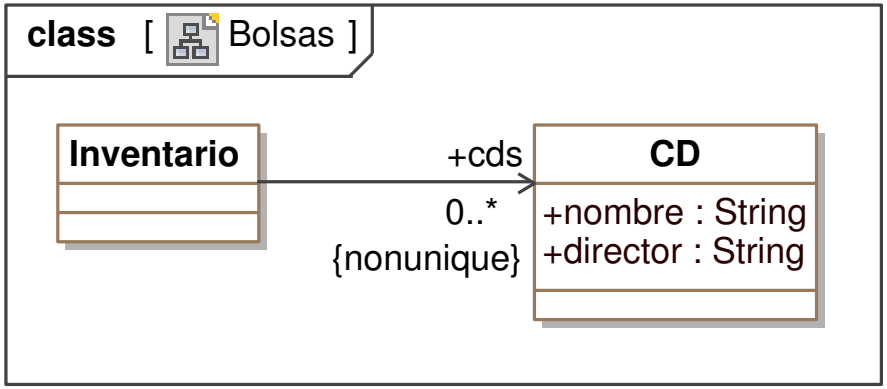
Bolsas (Multiconjuntos)

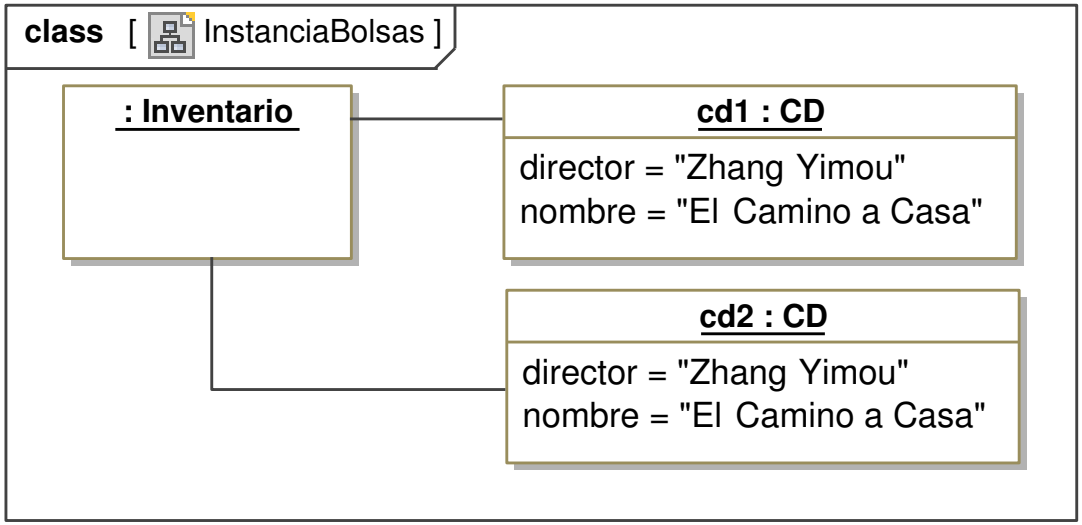
Colecciones de elementos potencialmente repetidos donde el orden de inserción (o posición) de los elementos en la estructura no es relevante.

Operaciones

1. `emptyBag : -> Bag`
2. `add : Bag E -> Bag`
3. `size : Bag -> Natural`
4. `contains/exists : Bag E -> Booleano`
5. `remove : Bag E -> Bag`
6. `removeOne: Bag E -> Bag`
7. `isEmpty : Bag -> Booleano`
8. `number : Bag E -> Natural`

Uso de Bolsas





Implementación de Bolsas

1. Vector (array) con acceso directo al último.
2. Lista enlazada con inserción en la cabeza.
3. Puedo usar un contador para mantener el tamaño calculado.
4. Se pueden hacer implementaciones muy eficientes con tablas de dispersión o árboles autobalanceados.

Comparación de Implementaciones

	Vector	Lista Enlazada	Dispersión	Árbol balanceado
emptyBag	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(1)$
add	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$
size	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
contains	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$
remove	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$
removeOne	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$
isEmpty	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

`N es el tamaño de la tabla de dispersión`

`n es el número de elementos en la bolsa`

Conjuntos

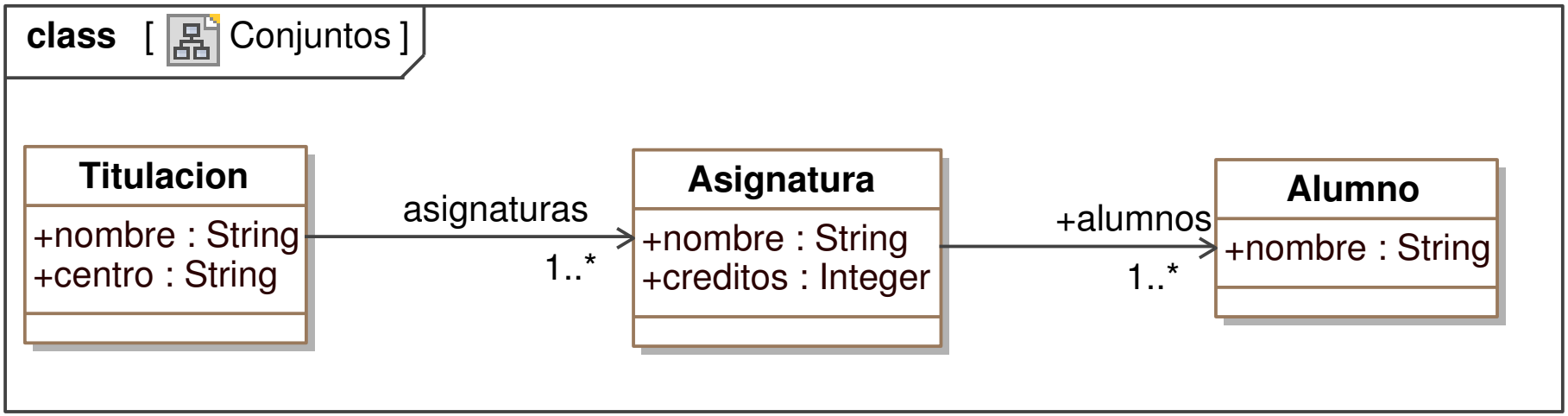
Conjuntos

Colecciones de elementos no repetidos donde el orden de inserción (o posición) de los elementos en la estructura no es relevante.

Operaciones

1. `emptySet : -> Set`
2. `add : Set E -> Set`
3. `size : Set -> Natural`
4. `contains/exists : Set E -> Booleano`
5. `remove : Set E -> Set`
6. `isEmpty : Set -> Booleano`
7. `union, intersection, diff : Set Set -> Set`
8. `subset, equal : Set Set -> Boolean`

Uso de Conjuntos



Implementación de Conjunto

1. Vector (array) con acceso directo al último.
2. Lista enlazada con inserción en la cabeza o al final.
3. Puedo usar un contador para mantener el tamaño calculado.
4. Antes de añadir tengo que comprobar si pertenece.
5. Se pueden hacer implementaciones muy eficientes con tablas de dispersión o árboles binarios de búsqueda autobalanceados.

Comparación de Implementaciones

	Vector	Lista Enlazada	Dispersión	Árbol balanceado
emptySet	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(N)$	$\mathcal{O}(1)$
add	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$
size	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
contains	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$
remove	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
isEmpty	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
union	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n\log(n))$
intersection	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n\log(n))$
diff	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n\log(n))$
subset	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n\log(n))$
equal	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n\log(n))$

`N es el tamaño de la tabla de dispersión`

`n es el tamaño del conjunto`

Secuencias/Listas

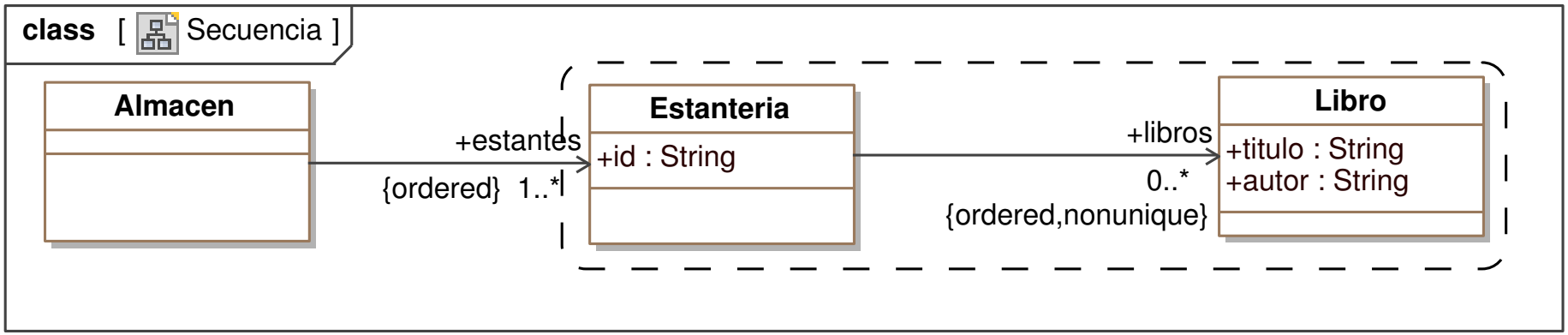
Secuencias/Listas

Colecciones de elementos potencialmente repetidos donde el orden de inserción (o posición) de los elementos en la estructura es relevante.

Operaciones sobre Secuencias

1. `emptyList : -> Seq`
2. `addTail, addHead : Seq E -> Seq`
3. `add : Seq E Natural -> Seq`
4. `getTail, getHead : Seq -> E`
5. `get : Seq Natural -> E`
6. `removeAll, removeFirst, removeLast : Seq E -> Seq`
7. `remove : Seq Natural -> Seq`
8. `replace : Seq Natural E -> Seq`
9. `contains : Seq E -> Booleano`
10. `find[Last, First]Index : Seq E -> Entero`
11. `size : Seq -> Natural`
12. `number : Seq E -> Natural`
13. `isEmpty : Seq -> Booleano`
14. `concat : Seq Seq -> Seq`
15. `equal, subseq : Seq Seq -> Boolean`

Uso de Secuencias



Implementación de Secuencias

1. Vector (array) con acceso directo al último elemento.
2. Lista enlazada con puntero a la cabeza y al final.
3. Puedo usar un contador para mantener el tamaño siempre calculado.

Comparación de Implementaciones

	Vector	Lista Enlazada
emptyList	$\mathcal{O}(1)$	$\mathcal{O}(1)$
addHead	$\mathcal{O}(n)$	$\mathcal{O}(1)$
addTail	$\mathcal{O}(1)$	$\mathcal{O}(1)$
add	$\mathcal{O}(1)$	$\mathcal{O}(1)$
getTail, getHead	$\mathcal{O}(1)$	$\mathcal{O}(1)$
get	$\mathcal{O}(1)$	$\mathcal{O}(n)$
remove[All, First, Last]	$\mathcal{O}(n)$	$\mathcal{O}(n)$
remove	$\mathcal{O}(n)$	$\mathcal{O}(n)$
contains	$\mathcal{O}(n)$	$\mathcal{O}(n)$
find[Last, First]Index	$\mathcal{O}(n)$	$\mathcal{O}(n)$
size	$\mathcal{O}(1)$	$\mathcal{O}(1)$
number	$\mathcal{O}(n)$	$\mathcal{O}(n)$
isEmpty	$\mathcal{O}(1)$	$\mathcal{O}(1)$
concat	$\mathcal{O}(n)$	$\mathcal{O}(1)$
equal	$\mathcal{O}(n)$	$\mathcal{O}(n)$

`n es el tamaño de la secuencia`

Secuencias sin Repeticiones

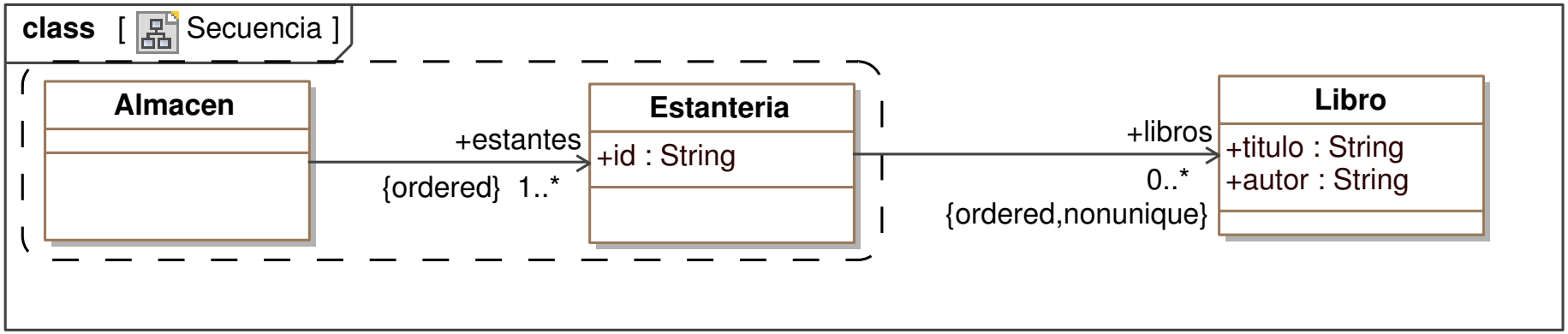
Secuencias sin Repeticiones

Colecciones de elementos no repetidos donde el orden de inserción (o posición) de los elementos en la estructura es relevante.

Operaciones sobre Secuencias sin Repeticiones

1. `emptyOrderSet : -> OrderSet`
2. `addTail, addHead : OrderSet E -> OrderSet`
3. `add : OrderSet E Natural -> OrderSet`
4. `getTail, getHead : OrderSet -> E`
5. `get : OrderSet Natural -> E`
6. `remove : OrderSet E -> OrderSet`
7. `removeAt : OrderSet Natural -> OrderSet`
8. `replace : OrderSet Natural E -> OrderSet`
9. `contains : OrderSet E -> Booleano`
10. `findIndex : OrderSet E -> Entero`
11. `size : OrderSet -> Natural`
12. `isEmpty : OrderSet -> Booleano`
13. `concat : OrderSet OrderSet -> OrderSet`
14. `equal, subseq : OrderSet OrderSet -> Boolean`

Uso de Secuencias sin Repeticiones



Implementación de Secuencias sin Repeticiones

1. Vector (array) con acceso directo al último elemento.
2. Lista enlazada con puntero a la cabeza y al final.
3. Puedo usar un contador para mantener el tamaño siempre calculado.

Comparación de Implementaciones

	Vector	Lista Enlazada
emptyOrderSet	$\mathcal{O}(1)$	$\mathcal{O}(1)$
addTail, addHead	$\mathcal{O}(n)$	$\mathcal{O}(n)$
add	$\mathcal{O}(n)$	$\mathcal{O}(n)$
getTail, getHead	$\mathcal{O}(1)$	$\mathcal{O}(1)$
get	$\mathcal{O}(1)$	$\mathcal{O}(n)$
remove	$\mathcal{O}(n)$	$\mathcal{O}(n)$
removeAt	$\mathcal{O}(n)$	$\mathcal{O}(n)$
contains	$\mathcal{O}(n)$	$\mathcal{O}(n)$
findIndex	$\mathcal{O}(n)$	$\mathcal{O}(n)$
size	$\mathcal{O}(1)$	$\mathcal{O}(1)$
isEmpty	$\mathcal{O}(1)$	$\mathcal{O}(1)$
concat	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
equal	$\mathcal{O}(n)$	$\mathcal{O}(n)$
subseq	$\mathcal{O}(n)$	$\mathcal{O}(n)$

`n es el tamaño del vector de la lista`

Índice

Vectores

Vectores

Colección de elementos (potencialmente repetidos), usualmente de *exactamente el mismo tipo*, donde la posición de cada elemento en la estructura es especialmente relevante; y donde el acceso tanto para escritura como para lectura se hace a través del índice de la posición de dicho elemento en la estructura.

1. `newVector : Natural -> Vector(E)`
2. `set : Vector(E) Natural E -> Vector(E)`
3. `get : Vector(E) Natural -> E`
4. `equal : Vector(E) Vector(E) -> Boolean`
5. `size : Vector(E) -> Natural`
6. `isSet : Vector(E) Natural -> Boolean`
7. `resize : Vector(E) Natural -> Vector(E)`

Matrices

Matrices

Colección bidimensional de elementos (potencialmente repetidos), usualmente de *exactamente el mismo tipo*, donde la posición de cada elemento en la estructura es especialmente relevante; y donde el acceso tanto para escritura como para lectura se hace a través de las coordenadas de la posición de dicho elemento en la estructura.

1. `newMatrix : Natural Natural -> Matrix(E)`
2. `set : Matrix(E) Natural Natural E -> Matrix(E)`
3. `get : Matrix(E) Natural Natural -> E`
4. `equal : Matrix(E) Matrix(E) -> Boolean`
5. `rows, columns : Matrix(E) -> Natural`
6. `isSet : Matrix(E) Natural Natural -> Boolean`
7. `resize : Matrix(E) Natural Natural -> Matrix(E)`

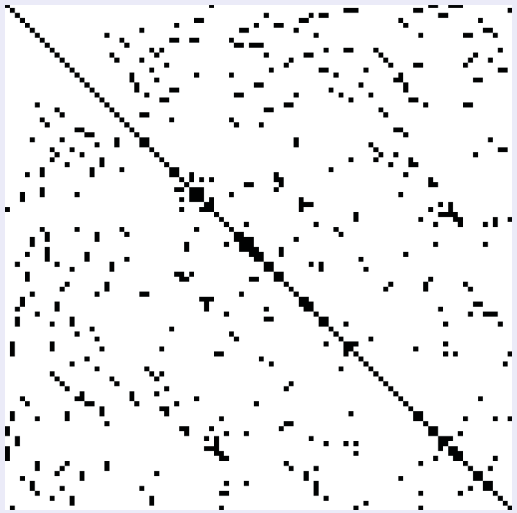
Uso de Matrices y Vectores

- Cálculo Científico en general.
- Software de Simulación.
- Resolución de problemas reducibles a grafos. (Ejemplo: Extracción de grupos perceptuales en imágenes.)

Implementación de Matrices/Vectores

Matriz Dispersa

Una matriz M se dice que es dispersa cuando la mayoría de sus elementos ($\approx 80\,\%$) son ceros.



Vector/Matriz Denso

Una matriz M se dice que es densa cuando no es dispersa.

Implementación de Matrices/Vectores

Vector/Matriz Denso

- Los vectores y matrices densos se implementan, salvo causa de fuerza mayor, usando los vectores (arrays) proporcionados por el lenguaje de programación.

Vector/Matriz Disperso

- Lista/Vector de listas enlazadas con los elementos no nulos de cada fila.
- Tablas de dispersión usando las coordenadas de un elemento como llave.
- Secuencia sin repeticiones ordenadas por filas y columnas de tuplas (*fila*, *columna*, *valor*).
- Formato Yale

1 2 0 0 0 3 9 0 0 1 4 0	Elements = [1 2 3 9 1 4] RowsBegin = [0 2 4 6] Columns = [0 1 1 2 1 2]

Índice

Clasificación por Tipo de Acceso

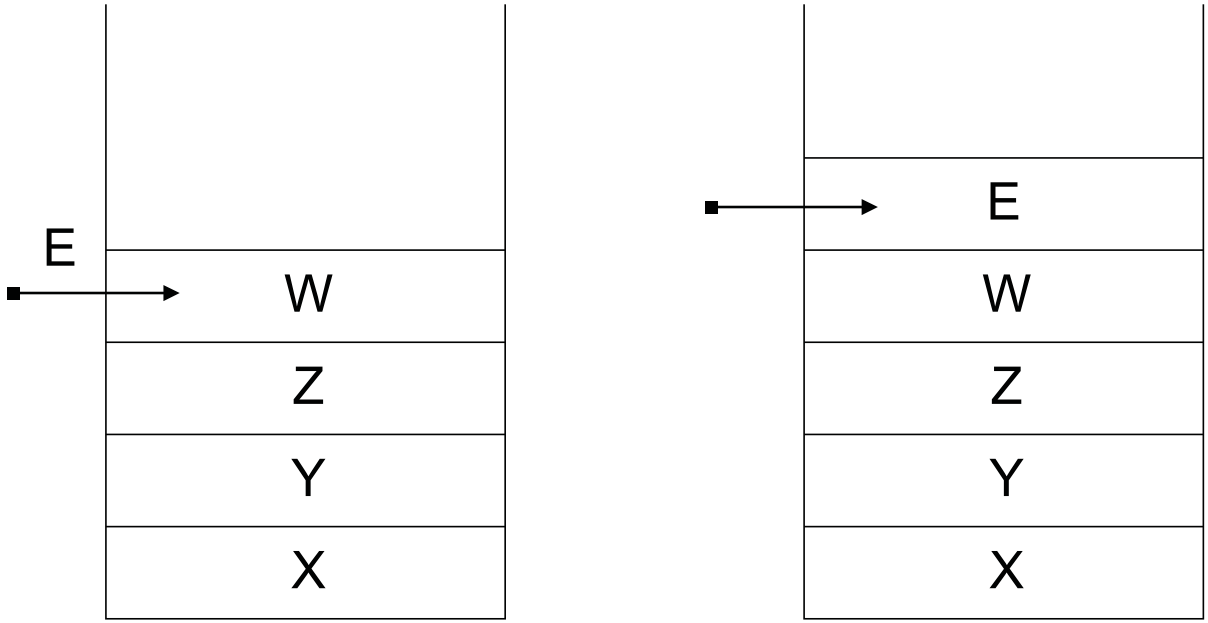
LIFO (Last In First Out): El elemento que recuperamos es el último elemento que hemos introducido en la estructura.

FIFO (First In First Out): El elemento que recuperamos es el primer elemento que hemos introducido en la estructura.

Pilas

Pilas

Colección de elementos (potencialmente repetidos) donde el orden de inserción de los elementos es relevante y sólo tenemos acceso al último elemento añadido. Es una estructura de tipo LIFO (*Last In, First Out*), donde el único elemento que podemos obtener es el último elemento añadido.



Operaciones sobre Pilas

1. `emptyStack : -> Stack`
2. `push : Stack E -> Stack`
3. `parcial peek : Stack -> E`
4. `parcial pop : Stack -> Stack`
5. `isEmpty : Stack -> Boolean`
6. `size : Stack -> Natural`

Uso de Pilas

1. Cálculo de expresiones en notación polaca inversa.
2. Cálculo de expresiones en lenguajes de programación.
3. Simulación de programas recursivos sobre programas iterativos
4. Ej: Determinación de palíndromos.

Implementación de Pilas

1. Vector (array) con acceso directo al último elemento.
2. Lista enlazada con puntero a la cabeza.
3. Puedo usar un contador para mantener el tamaño siempre calculado.

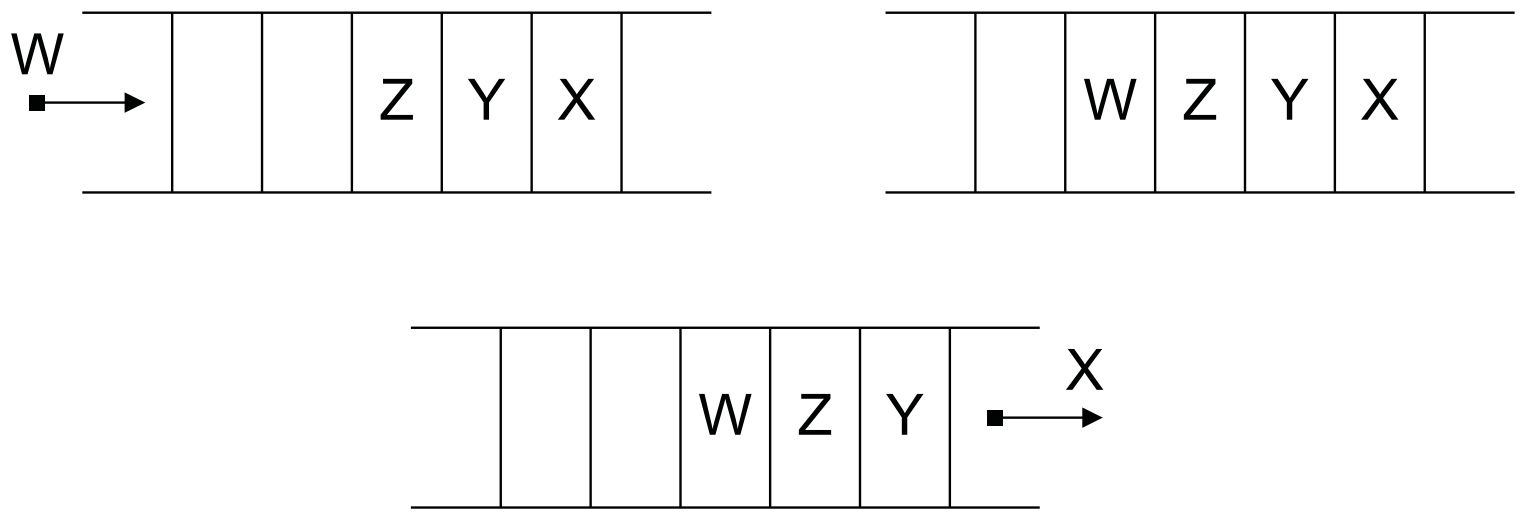
Comparación de Implementaciones

	Vector	Lista Enlazada
emptyStack	$\mathcal{O}(1)$	$\mathcal{O}(1)$
push, pop	$\mathcal{O}(1)$	$\mathcal{O}(1)$
peek	$\mathcal{O}(1)$	$\mathcal{O}(1)$
isEmpty	$\mathcal{O}(1)$	$\mathcal{O}(1)$
size	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Colas

Colas

Colección de elementos (potencialmente repetidos) donde el orden de inserción de los elementos es relevante y sólo tenemos acceso al elemento más antiguo añadido. Es una estructura de tipo FIFO (*First In, First Out*), donde el único elemento que podemos obtener es el elemento más antiguo añadido.



Operaciones sobre Colas

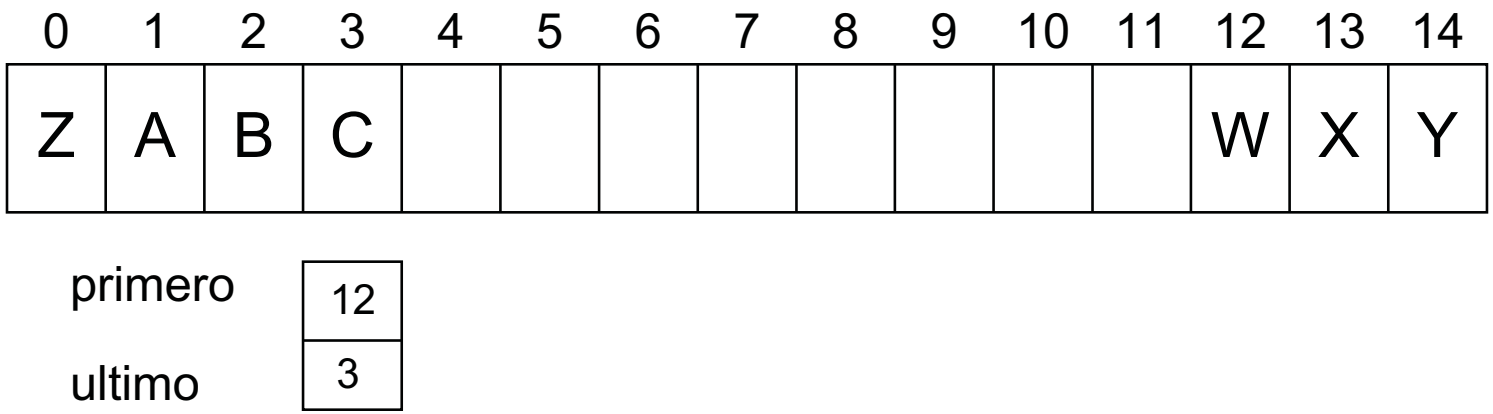
1. `emptyQueue : -> Queue`
2. `add : Queue E -> Queue`
3. `peek : Queue -> E`
4. `remove : Queue -> Queue`
5. `isEmpty : Queue -> Boolean`
6. `size : Queue -> Natural`

Uso de Colas

1. Sistemas Operativos (planificadores de procesos, colas de eventos).
2. Software de Telemática (colas de paquetes).
3. Streaming (vídeo por internet, TDT).
4. Programas de Simulación de Procesos.
5. Interfaces Gráficas (colas de eventos).

Implementación de Colas

1. Lista enlazada con puntero al primero y al ultimo.
2. Vector (array) circular con indicadores de primero y ultimo.
3. Puedo usar un contador para mantener el tamaño siempre calculado.



Comparación de Implementaciones

	Vector	Lista Enlazada
emptyQueue	$\mathcal{O}(1)$	$\mathcal{O}(1)$
add	$\mathcal{O}(1)$	$\mathcal{O}(1)$
peek	$\mathcal{O}(1)$	$\mathcal{O}(1)$
remove	$\mathcal{O}(1)$	$\mathcal{O}(1)$
isEmpty	$\mathcal{O}(1)$	$\mathcal{O}(1)$
size	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Índice

Nociones sobre Relaciones de Orden

Relación de Orden Estricto

Dada una relación binaria R sobre un conjunto S, dicha relación es de orden estricto si es:

1. **Irreflexiva**: $\forall x \in S, (x, x) \notin R$
2. **Transitiva**: $(x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$

Relación de Orden Total No Estricto

Dada una relación binaria R sobre un conjunto S, dicha relación es de orden total estricto si es una relación de orden parcial estricto y es además total, es decir $\forall x, y \in Sx \neq y \in R(x, y) \notin R \Rightarrow (y, x) \in R$

Nociones sobre Relaciones de Orden

Relación de Orden Parcial No Estricto

Dada una relación binaria R sobre un conjunto S, dicha relación es de orden no estricto si es:

1. **Reflexiva**: $\forall x \in S, (x, x) \in R$
2. **Transitiva**: $(x, y) \in R \wedge (y, z) \in R \Rightarrow (x, z) \in R$
3. **Antisimétrica**: $(x, y) \in R \wedge (y, x) \in R \Rightarrow x = y$

Relación de Orden Total No Estricto

Dada una relación binaria R sobre un conjunto S, dicha relación es de orden total (no estricto) si es una relación de orden parcial (no estricto) y es además total, es decir $\forall x, y \in S, (x, y) \notin R \Rightarrow (y, x) \in R$

TADs Lineales con Orden Natural

TADs Lineales con Orden Natural

Un tipo de datos abstracto lineal $T = \{a_1, \ldots, a_i, \ldots, a_j, \ldots, a_n\}$ se dice que está ordenado topológicamente con respecto a una relación de orden total R, si, dadas dos funciones $suc : T \to T$, que devuelve el sucesor de cada elemento en T menos del último, y $last : T \to Boolean$, que indica si un elemento del TAD es el último, se cumple que
$\forall x \in T, ultimo(x) \neq TRUE; (x, suc(x)) \in R$

Secuencias/Listas con Orden Natural

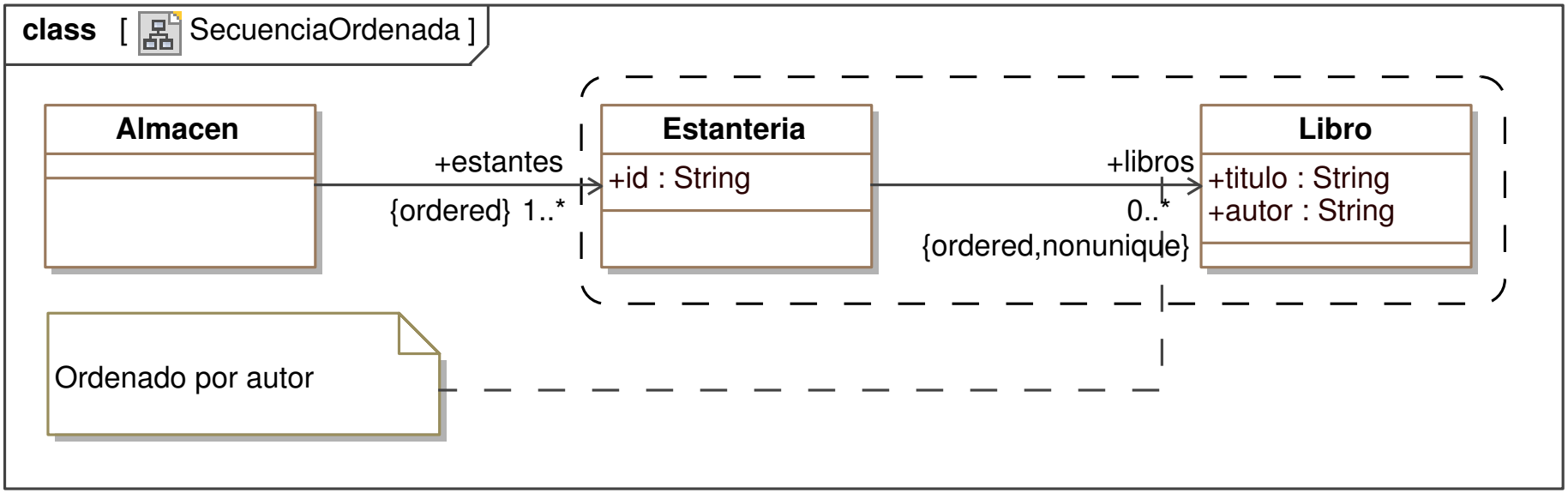
Secuencia/Lista con Orden Natural

Colección de elementos potencialmente repetidos donde la posición de los elementos en la estructura es relevante, y donde la colección está ordenada con respecto a alguna relación de orden total.

Operaciones sobre Secuencias con Orden Natural

1. `emptySortedSeq : -> SortedSeq`
2. `add : SortedSeq E -> SortedSeq`
3. `getTail, getHead : SortedSeq -> E`
4. `get : SortedSeq Natural -> E`
5. `remove[All, One] : SortedSeq E -> SortedSeq`
6. `remove : SortedSeq Natural -> SortedSeq`
7. `contains : SortedSeq E -> Boolean`
8. `find[First, Last]Index : SortedSeq E -> Integer`
9. `size : SortedSeq -> Natural`
10. `number : SortedSeq E -> Natural`
11. `isEmpty : SortedSeq -> Boolean`
12. `mix : SortedSeq SortedSeq -> SortedSeq`
13. `subseq, equal : SortedSeq SortedSeq -> Boolean`
14. `min, max : SortedSeq -> Element`

Uso de Secuencias Ordenadas



Implementación de Secuencias Ordenadas

1. Vector (array) con acceso directo al último elemento.
2. Lista enlazada con puntero a la cabeza y al final.
3. Puedo usar un contador para mantener el tamaño siempre calculado.
4. En todos los casos mantengo la estructura siempre ordenada (si interesa).

Búsqueda Binaria sobre TADs Acceso Aleatorio

```
// Pre: inicio <= fin) => (0 <= inicio, fin < size(t))
FUNCION contains(t : ITAD_AA(E), e : E,
                 inicio : NATURAL, fin : NATURAL) : BOOLEAN

    result : BOOLEAN;  medio : NATURAL;
    result := FALSE;

    SI (inicio <= fin) ENTONCES
        medio := (inicio + fin) div 2;
        SI (equal(get(t,medio),e)) ENTONCES
            result := TRUE;
        SINO
            result := contains(t,e,inicio,medio-1) OR
                      contains(t,e,medio+1,fin);
        FINSI
    FINSI

    DEVOLVER result;
FINFUNCION
```

Búsqueda Binaria sobre TADs Acceso Aleatorio y Ordenados

```
// Pre: inicio <= fin) => (0 <= inicio, fin < size(t))
FUNCION contains(t : ISortedTAD_AA(E), e : E,
                 inicio : NATURAL, fin : NATURAL) : BOOLEAN

    result : BOOLEAN;  medio : NATURAL;
    result := FALSE;

    SI (inicio <= fin) ENTONCES
        medio := (min + max) div 2;
        SI (equal(get(t,medio),e)) ENTONCES
            result := TRUE;
        SINOSI (lessThan(get(t,medio),e)) ENTONCES
            result := contains(t,e,medio+1,fin)
        SINO
            result := contains(t,e,inicio,medio-1)
        FINSI
    FINSI

    DEVOLVER result;
FINFUNCION
```

Comparación de Implementaciones

	Vector	Lista Enlazada	Arbol Balanceado
emptySortedSeq	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
add	$\mathcal{O}(\log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
getTail, getHead	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$
get	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
removeAll	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(m\log(n))$
removeOne	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
remove	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
contains	$\mathcal{O}(\log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
find[Last, First]Index	$\mathcal{O}(max(m, \log(n)))$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
size	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
number	$\mathcal{O}(max(m, \log(n)))$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
isEmpty	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
mix	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n\log(n))$
equal, subseq	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
min, max	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$

`n es el tamaño de la secuencia, m es el número de copias de un elemento`

Secuencias Ordenadas sin Repeticiones

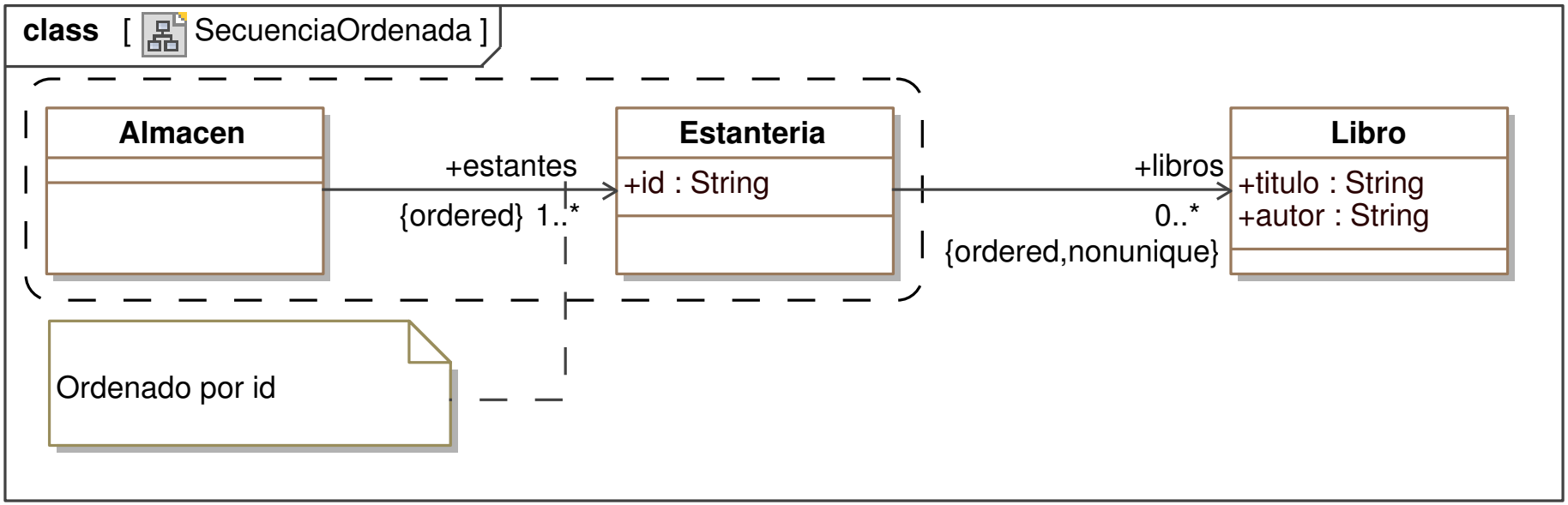
Secuencia Ordenadas sin Repeticiones

Colección de elementos no repetidos donde el orden de inserción (o posición) de los elementos en la estructura es relevante, y donde la colección está ordenada con respecto a alguna relación de orden total.

Operaciones sobre Secuencias Ordenadas sin Repeticiones

1. `emptySortedOrSet : -> SortedOrSet`
2. `add : SortedOrSet E -> SortedOrSet`
3. `get[Tail, Head] : SortedOrSet -> E`
4. `get : SortedOrSet Natural -> E`
5. `remove : SortedOrSet E -> SortedOrSet`
6. `removeAt : SortedOrSet Natural -> SortedOrSet`
7. `contains : SortedOrSet E -> Boolean`
8. `findIndex : SortedOrSet E -> Integer`
9. `size : SortedOrSet -> Natural`
10. `isEmpty : SortedOrSet -> Booleano`
11. `mix : SortedOrSet SortedOrSet -> SortedOrSet`
12. `equal, subseq : SortedOrSet SortedOrSet -> Boolean`
13. `min, max : SortedSeq -> Element`

Uso de Secuencias Ordenadas sin Repeticiones



Implementación de Secuencias Ordenadas sin Repeticiones

1. Vector (array) con acceso directo al último elemento.
2. Lista enlazada con puntero a la cabeza y al final.
3. Puedo usar un contador para mantener el tamaño siempre calculado.
4. En todos los casos mantengo la estructura siempre ordenada (si interesa).

Comparación de Implementaciones

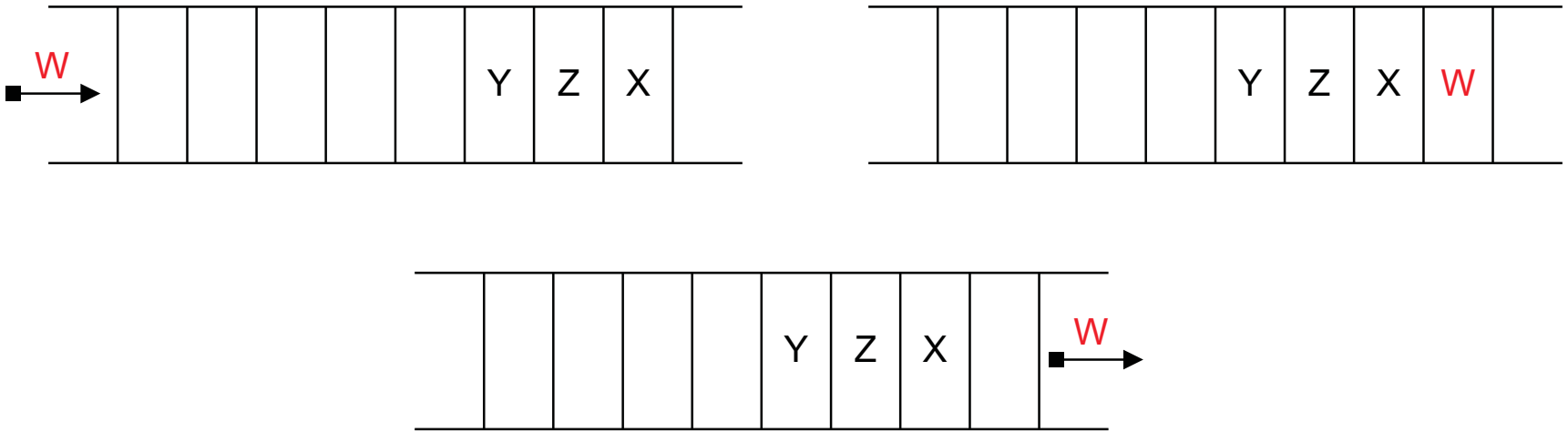
	Vector	Lista Enlazada	Árbol Balanceado
emptySortedOrSet	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
add	$\mathcal{O}(\log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
getTail, getHead	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$
get	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
remove	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
removeAt	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
contains	$\mathcal{O}(\log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
findIndex	$\mathcal{O}(\log(n))$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
size	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
isEmpty	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
mix	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n\log(n))$
equal, subseq	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
min, max	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$

`n es el tamaño del vector de la lista`

Colas con Prioridad

Colas con Prioridad

Colección de elementos (potencialmente repetidos) donde el orden de inserción de los elementos es relevante y sólo tenemos acceso al elemento más antiguo y de más alta prioridad añadido. Es una estructura de tipo FIFO con Prioridad (*First In, First Out*), donde el único elemento que podemos obtener es el elemento de más alta prioridad más antiguo añadido.



Operaciones sobre Colas con Prioridad

1. `emptyQueue : -> Queue`
2. `add : Queue E -> Queue`
3. `peek : Queue -> E`
4. `remove : Queue -> Queue`
5. `isEmpty : Queue -> Boolean`
6. `size : Queue -> Natural`

Uso de Colas con Prioridad

1. Sistemas Operativos (planificadores de procesos, colas de eventos).
2. Software de Telemática (colas de paquetes).
3. Programas de Simulación de Procesos.

Implementación de Colas con Prioridad

1. Lista enlazada con puntero al primero y al ultimo.
2. Vector (array) circular con indicadores de primero y ultimo.
3. Montículos (Heaps) (no preserva FIFO entre elementos de igual prioridad).
4. Puedo usar un contador para mantener el tamaño siempre calculado.
5. En todos los casos mantengo la estructura siempre ordenada (si interesa).

Comparación de Implementaciones

	Vector	Lista Enlazada	Montículo
emptyQueue	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
add	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
peek	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
remove	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n))$
isEmpty	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
size	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Índice

Tablas

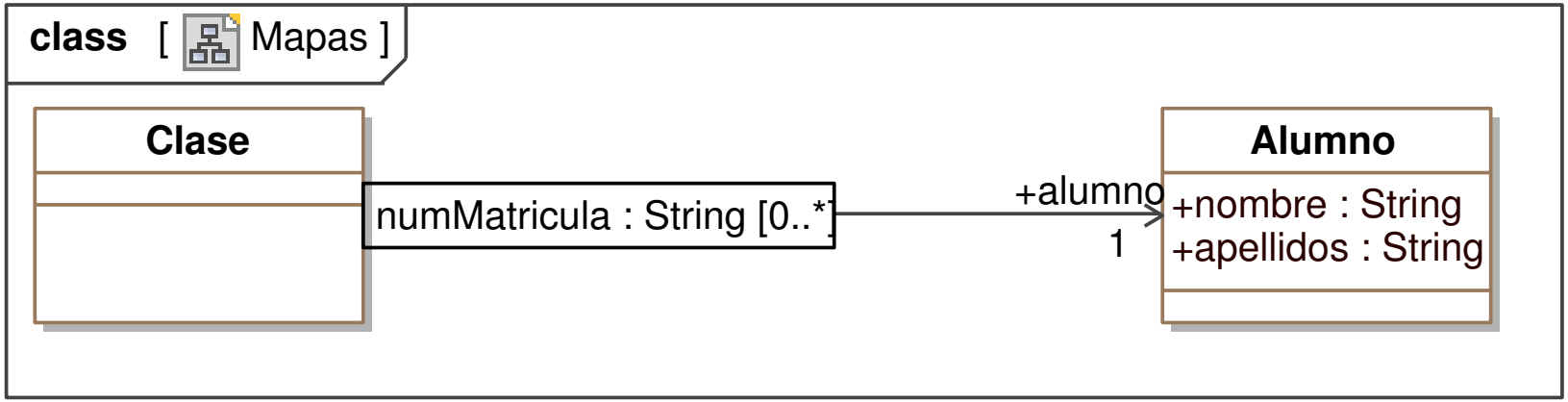
Tablas (Aplicaciones, Diccionarios, Mapas)

Colecciones de elementos identificables por una clave, que debe ser única para cada elemento de la colección, y donde el acceso, tanto para lectura como para escritura, se hace a través de dicha clave.

Operaciones

1. `emptyTable : -> Table(K,E)`
2. `set : Table(K,E) Key E -> Table(K,E)`
3. `get : Table(K,E) Key -> E`
4. `remove : Table(K,E) Key -> Table(K,E)`
5. `contains : Table(K,E) Key -> Boolean`
6. `size : Table(K,E) -> Natural`
7. `isEmpty : Table(K,E) -> Boolean`

Uso de Tablas



1. Algoritmos de compresión (ej. LZW).
2. Tablas de Bases de Datos.

Implementación de Tablas

1. Vector (array) con acceso directo al último.
2. Lista enlazada con inserción en la cabeza (o al final).
3. Tablas de dispersión usando como valor de dispersión la clave.
4. Árboles binarios de búsqueda balanceados, ordenado por clave.
5. Puedo usar un contador para mantener el tamaño calculado.

Comparación de Implementaciones

	Dispersión	Vector	Lista Enlazada	Árbol Balanceado
emptyTable	$\mathcal{O}(M)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
set	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
get	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
remove	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
contains	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log(n))$
size	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
isEmpty	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

`M es el tamaño de la tabla de dispersión`

`n es el tamaño de la bolsa`

Índice

¿Qué tengo que saber de todo esto?

1. Saber y entender como funcionan los diferentes TADs lineales.
2. Se capaz de escoger la implementación más adecuada para un TAD.
3. Ser capaz de implementar los diferentes TADs lineales.
4. Ser capaz de usar TADs lineales para diseñar y construir aplicaciones.
5. Ser capaz de estimar la complejidad de las operaciones de un TAD lineal, conocida su implementación.