4 Introducción a los tipos abstractos de datos

- Definiciones
- TAD String
- Concepto de contenedor
- Colecciones e Iteradores
- Relaciones entre elementos
- TAD's contenedores

Estructuras de datos, curso 2006/07
I.T.Informática de Gestión
Universidad de Valladolid

Concepto de tipo abstracto de datos (TAD)

- Tipo abstracto de datos: (TAD)
  - Un conjunto de valores y **operaciones** asociadas
  - especificados de manera **precisa**
  - e **independiente** de la implementación
- Objetivo:
  - Separar **interfaz** (definición operaciones) de **implementación** (representación de los datos + algoritmos de las operaciones).
- Notación:
  - El estado de un TAD viene dado por la secuencia de operaciones realizadas sobre él.
  - La definición de las operaciones suele darse mediante axiomas y reglas lógicas.

Ejemplo de definición de PILA

Definición axiomática (TAD)

ESPECIFICACIÓN PILA

TAD pila[elemento]

OPERACIONES

- crear : → pila
- esta_vacía : pila → booleano
- cima : pila → elemento
- apilar : pila, elemento → pila
- desapilar : pila → pila

PRECONDICIONES

- cima(*p*) ⇔ ¬ esta_vacía(*p*)
- desapilar(*p*) ⇔ ¬ esta_vacía(*p*)

ECUACIONES

- esta_vacía(crear) == **T**
- esta_vacía(apilar(*p*, *x*)) == **F**
- cima(apilar(*p*, *x*)) == *x*
- desapilar(apilar(*p*, *x*)) == *p*

FIN_ESPECIFICACIÓN

Definición por código (no TAD)

```
type
  PNodo = ^TNodo;
  TPila = PNodo;
  TNodo = record
            Dato : ...;
            Sig : TPila
          end;

function cima(P: TPila) : PNodo;
begin
  Result := P
end;

procedure apilar(var P: TPila;
                 X: PNodo);
begin
  X^.Sig := P; P := X
end;

procedure desapilar(var P: TPila);
begin
  P := P^.Sig
end;
```

3

Ejemplo de definición en Orientación a Objeto

Clase Abstracta (Eiffel)

```
deferred class CONJUNTO[E_]
feature {ANY}
  crear_vacio is deferred
  ensure esta_vacio
  end

  esta_vacio: BOOLEAN is deferred

  pertenece(e: E_): BOOLEAN is deferred
  ensure Result implies not esta_vacio
  end

  insertar(e: E_) is deferred
  ensure pertenece(e) and not esta_vacio
  end

  quitar(e: E_) is deferred
  ensure not pertenece(e)
  end
end -- CONJUNTO
```

4

TAD Cadena de caracteres (STRING)

"Secuencia ordenada de símbolos (caracteres) definidos por un determinado alfabeto"

- Aunque es posible considerar un STRING como un *contenedor de caracteres*, no es el enfoque habitual. En general se consideran como un tipo simple especial.
- El alfabeto suele definir las reglas de **comparación** entre cadenas.
- Las operaciones básicas de creación son la **concatenación** y la extracción, inserción y búsqueda de **subcadenas**. El acceso a caracteres individuales se consideran operaciones secundarias.
- Se pueden clasificar en **mutables** (es posible modificar el contenido de una cadena) e **inmutables** (una vez creadas no se pueden modificar). Es normal que existan clases para representar ambos tipos.
- Implementaciones:
  - Arrays de caracteres (estilo C ó Pascal, capacidad fija o variable)
  - Ropes (árboles de subcadenas)
  - Tries
  - Árboles o arrays de sufijos

5

Operaciones básicas del TAD STRING (I)

MUTABLES e INMUTABLES:

- **Creación** como copia de otro, de un vector de caracteres, literal, etc.
- **Comparación** con otro STRING
- **Búsqueda** de una subcadena: Es una operación de gran importancia práctica, y existen varios algoritmos para realizarla de manera eficiente. Suponiendo que las cadenas se representan mediante un **vector de caracteres**, que la cadena tiene longitud n, y la subcadena longitud m, se muestra la eficiencia de alguno de éstos algoritmos:

Algoritmo	Preproc.	Peor caso	Promedio	Características
"Normal"	0	$\Theta(n \cdot m)$	¿?	Mejor caso $\Theta(n)$
Rabin-Karp	$\Theta(m)$	$\Theta(n \cdot m)$	$\Theta(n+m)$	Func. dispersión
Knuth-Morris-Prat	$\Theta(m)$	$\Theta(n)$	$\Theta(n)$	
Boyer-Moore	$\Theta(m)$	$\Theta(n)$	$\Theta(n/m)$	

Otras operaciones relacionadas son:

- Búsqueda de múltiples subcadenas
- Búsqueda de un **patrón** de subcadena
- Búsqueda de zona con mejor ajuste

6

Operaciones básicas del TAD STRING (II)

MUTABLES e INMUTABLES (sigue):

- Es posible mejorar la eficiencia de la búsqueda de una subcadena usando representaciones más sofisticadas que el vector de caracteres. En particular, usando **árboles de sufijos** es posible obtener un tiempo $\Theta(m)$ (independiente del tamaño del texto donde se busca).
- **Concatenación**: En inmutables, devolver la cadena concatenada. En mutables, extender la cadena con la otra. Usando listas enlazadas circulares o **ropes** es posible realizar ésta operación en $O(1)$.
- Obtención de la **subcadena** que se encuentra en una determinada posición y longitud.
- Acceso al carácter i-ésimo.

MUTABLES sólo:

- **Inserción** de subcadena en una determinada posición.
- **Borrado** de un bloque contiguo de la cadena.
- Modificación del carácter i-ésimo.

7

Contenedores

- TADs contenedores:
  - Almacenan **información** sobre varios elementos.
  - No tienen porqué almacenar "físicamente" a los elementos: pueden almacenar referencias a ellos y/o información sobre relaciones entre ellos. Además, es común que los elementos sean "compartidos" por varias estructuras.
- Criterios de clasificación:
  - Los elementos se pueden considerar **atómicos** o divididos en **partes** relevantes.
  - Puede imponerse requisitos sobre los elementos o alguna parte de ellos:
    - **Equivalencia**: Poder saber si dos elementos son iguales.
    - **Comparación**: Poder saber si un elemento es mayor o menor que otro.
  - Puede permitirse elementos **repetidos** o no.
  - Puede existir una **relación** (o no) entre los elementos.
  - Si existe relación de orden, ésta puede ser **externa** (decidida por quien usa la estructura) o **interna** (determinada por los valores de los elementos)

8

Implementación de contenedores

La inmensa mayoria de las representaciones de contenedores tan sólo hacen uso de los siguientes elementos básicos de construcción (presentes o definibles en prácticamente todos los lenguajes imperativos):

- Arrays
- Referencias a elementos:
  - Punteros a variables dinámicas
  - Referencias a objetos
  - Índices a la posición donde se encuentra en un array
  - Nombre del fichero y posición dentro de éste en que se encuentra
- Funciones resumen ó dispersión (hash functions):
  - Una función que traduce el valor de un elemento a un número, tipicamente en el rango de los enteros de la máquina.
  - No es necesario (ni posible en la mayoría de los casos) que a cada valor de un elemento se le asigne un número distinto.
  - Para ser útiles deben cumplir una serie de requisitos.

9

Contenedores principales - **Conjunto**

"Colección de valores sin un orden definido en la que no se permiten elementos repetidos"

- Los conjuntos se utilizan para *clasificar* valores en dos categorías: Los que pertenecen al conjunto y los que no pertenecen a él.
- Las operaciones básicas son:
  - **Pertenencia**: Comprobar si un valor pertenece al conjunto.
  - **Inclusión**: Hacer que un valor pertenezca al conjunto.
  - **Exclusión**: Hacer que un valor no pertenezca al conjunto.
  - También pueden ser importantes las operaciones de combinación de conjuntos: **unión**, **intersección**, **diferencia**.
  - El **recorrido** de los elementos de un conjunto no garantiza ningún orden en particular.
- Existen muchas posibilidades respecto a la implementación:
  - Arrays de bits indexados por valor.
  - Tablas de dispersión (Hash tables)
  - Contigua lineal (arrays de valores), ya sea desordenados u ordenados.
  - Árboles binarios equilibrados (Árboles AVL, Red-Black Trees, etc.)
  - Opciones exóticas: Tries, Bloom filter, Van Emde Boas Tree.

10

Contenedores principales - **Lista**

"Colección ordenada de entidades"

- A diferencia de los conjuntos, en las listas existe una relación de orden impuesta a los elementos, y se permiten elementos repetidos. Si el orden es interno hablamos de **Listas ordenadas**.
- El orden permite asociar los elementos con un conjunto de **índices**.
- Existen muchas variantes que difieren en las restricciones que impongamos a las operaciones de acceso, inserción y borrado:
  - En principio basta con que dispongamos de cuatro operaciones para a partir de ellas poder realizar cualquier otra: comprobación de si la lista está vacía, y acceso, inserción y borrado del primer elemento.
  - Las **listas secuenciales** permiten señalar un elemento ó posición en la lista (cursor), en base al cual se realizan todas las operaciones. Tambien se permite cambiar el cursor (pasar a siguiente y/o anterior).
  - Las **listas indexadas** permiten realizar las operaciones indicando el índice del elemento.
  - Las **pilas**, **colas** y **bicolas** sólo permiten realizar operaciones sobre los elementos extremos.
  - En las **listas ordenadas** el elemento con índice *i* es el *i*-ésimo menor.
  - Las **colas de prioridad** son un caso especial de lista ordenada en que las operaciones se restringen a un extremo (el elem. mínimo o máximo)

11

Contenedores principales - **Lista**

IMPLEMENTACIONES

- **Listas (orden externo)**: Listas indexadas, secuenciales, pilas, colas, bicolas. Dependiendo de la variante concreta, y de las operaciones más relevantes para el problema, se usarán alguna de las siguientes implementaciones:
  - Contigua lineal, contigua circular (arrays)
  - Listas enlazadas (lineales|circulares, simples|dobles)
- **Listas ordenadas (orden interno)**
  - Contigua lineal ordenada (arrays)
  - Árboles binarios de búsqueda (ABB)
  - ABB equilibrados (Árboles AVL, Árboles B, Red-Black Trees, ..)
  - Otras opciones: Skip Lists, Splay Trees
- **Colas de prioridad**
  - Montículos binarios (binary heap)
  - Otros montículos: Binomiales, Fibonacci
  - Treaps

12

Contenedores principales - **Mapa**

"TAD compuesto por un conjunto de claves y una colección de valores donde cada clave tiene asociado un valor"

- Un mapa (associative array, map, lookup table) es una generalización del concepto de conjunto, en la cual cada elemento que pertenece al conjunto (denominado clave) tiene asociado un valor.
- Los elementos del mapa son pares (clave, valor)
- No pueden existir claves repetidas. Sin embargo si que pueden existir valores repetidos.
- Los elementos se localizan mediante su clave.
- Las implementaciones principales suelen ser las mismas que las de los conjuntos, adaptadas al hecho de que ahora deben almacenar el valor asociado a la clave.
- Si se necesitan operaciones de **recorrido en orden** de las claves, entonces nos referimos al **TAD Tabla**.
- Si en lugar de un valor se debe asociar a cada clave una **lista de valores**, entonces nos referimos al **TAD Diccionario**.

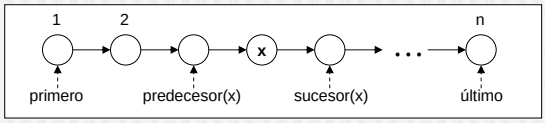
13

Colecciones e Iteradores

- Actualmente existe una tendencia a organizar las librerias de clases bajo los conceptos de **Colección** (Collection) e **Iterador** (Iterator). Ejemplo: Java, C#, Eiffel, .NET. Existen lenguajes que añaden un tipo adicional de bucle, el bucle controlado por iterador.
- Una **colección** es una fuente de datos (estructura de datos, fichero, conexión de red, etc.) de la cual se puede obtener un iterador para acceder a sus datos.
- Un **iterador** es un mecanismo que permite acceder a un elemento (el elemento actual) y pasar al siguiente (dependiendo del tipo de iterador es posible que se permitan más operaciones).
- Tipicamente el concepto de colección se representa mediante una clase abstracta o interfaz de la que heredan/implementan aquellas clases que pueden servir de fuente de datos.
- El iterador se representa por una clase/interfaz con las operaciones de acceso y paso al siguiente. Suelen existir clases iteradoras con más operaciones disponibles, que heredan del iterador general.
- El objetivo es proporcionar un mecanismo **uniforme** de acceso a cualquier fuente de datos, considerándolas a todas como un TAD Lista secuencial con operaciones restringidas.

14

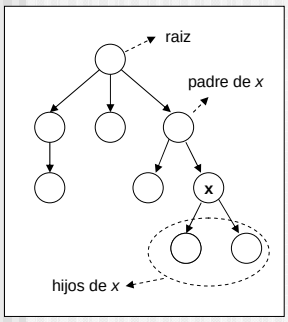
Relación de precedencia u orden



- Existe una relación de **orden total** sobre los elementos.
- Existen dos elementos **extremos**: El **primero** y el **último**.
- Todo elemento (salvo el primero) tiene un único elemento **predecesor**.
- Todo elemento (salvo el último) tiene un único elemento **sucesor**.
- Se pueden realizar operaciones basadas en la **posición** de los elementos:
  - Sobre algún extremo.
  - Sobre el predecesor o sucesor del elemento actual.
  - Asignando un índice a cada elemento.
- El orden puede ser **interno** (basado en el valor del elemento o de una parte suya) u **externo** (impuesto por las operaciones de inserción y borrado)

15

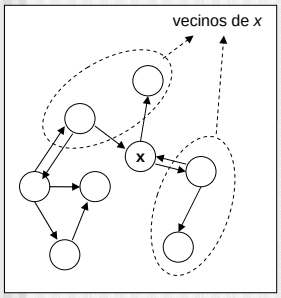
Relación de parentesco o jerarquía (árbol)



- Los elementos se relacionan con una estructura jerárquica (tipo **arbol**)
- Hay un elemento distinguido, el **raiz**, y los demas se particionan en subárboles, y así recursivamente.
- Cada elemento (excepto el raiz) tiene un **padre** (superior) y cero o más **hijos** (subordinados).
- Permite representar elementos con una estructura jerárquica (más compleja y flexible que la relación de orden).
- A diferencia de la relación de orden, no existe una única forma distinguida de recorrer los elementos, sino varias: preorden, inorden, postorden y por niveles.
- Si se impone un **orden interno** la relación puede servir como una representación (montículos, arboles binarios de búsqueda, etc.) útil para varios TADs

16

Relación de vecindad (grafo)



- No existen restricciones respecto a la forma de relacionar unos elementos con otros.
- Cada elemento puede estar relacionado con 0 o más elementos, sus vecinos.
- No existe nigún elemento distinguido.
- No hay recorridos distinguidos, tan sólo la posibilidad de elegir un nodo y recorrer en profundidad o en anchura.
- Las relaciones entre elementos pueden ser (o no) direccionales y suelen llevar información asociada.

17

Clasificación operaciones básicas

Acceso ó Borrado

- por **Valor**
  - **Elemento**
    - • Pertenencia
    - • Recuento
  - **Clave**
    - • Dato asociado
    - • Lista elementos asociados a clave

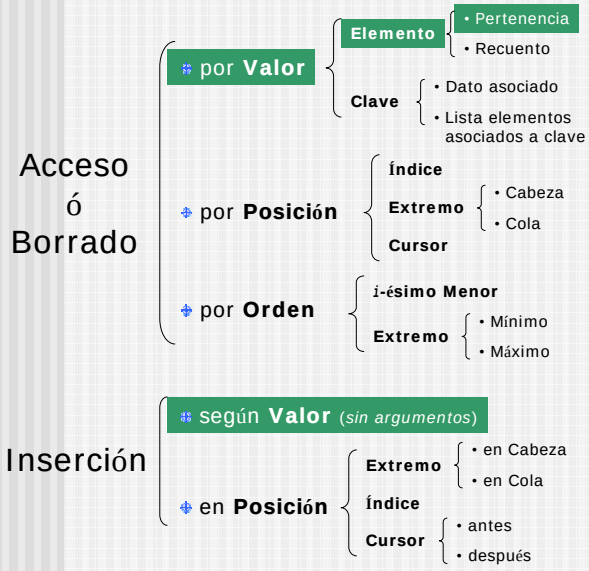
  → **Resultado del acceso**
- por **Posición**
  - **Índice**
  - **Extremo**
    - • Cabeza
    - • Cola
  - **Cursor**
- por **Orden**
  - ***i*-ésimo Menor**
  - **Extremo**
    - • Mínimo
    - • Máximo

Inserción

- según **Valor** (*sin argumentos*)
- en **Posición**
  - **Extremo**
    - • en Cabeza
    - • en Cola
  - **Índice**
  - **Cursor**
    - • antes
    - • después

18

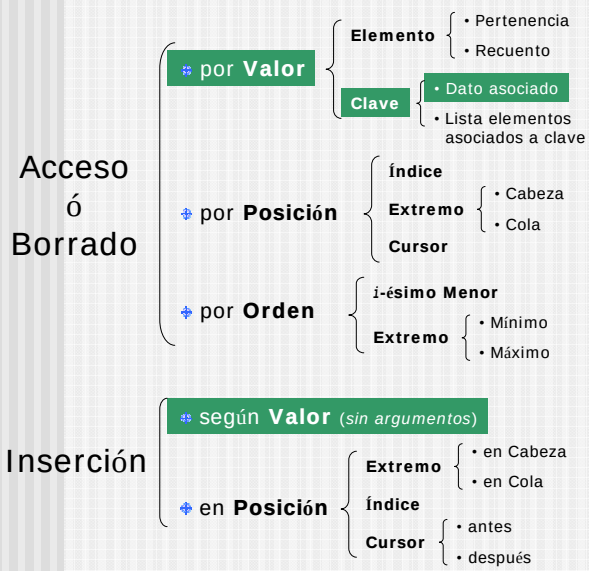
Conjunto (Set)



- ✓ **Sin relación**
- ✓ **Elementos atómicos**
- ✓ **Req: Equivalencia**
- ✓ **Sin elementos repetidos**
- ✓ **Otras operaciones:**
  - ➢ **Unión**
  - ➢ **Recorrido**

19

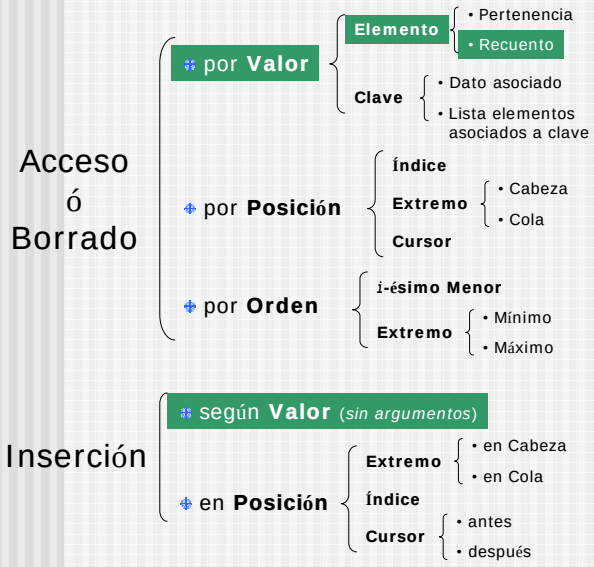
Mapa (Associative array, Lookup table, Map, Index)



- ✓ **Sin relación**
- ✓ **Elementos son pares clave - dato**
- ✓ **Req: Equivalencia claves**
- ✓ **Sin claves repetidas**
- ✓ **Otras operaciones:**
  - ➢ **Modificar dato**
- ✓ **Usos:**
  - ➢ **Lenguajes Awk, Perl, JavaScript, PHP**

20

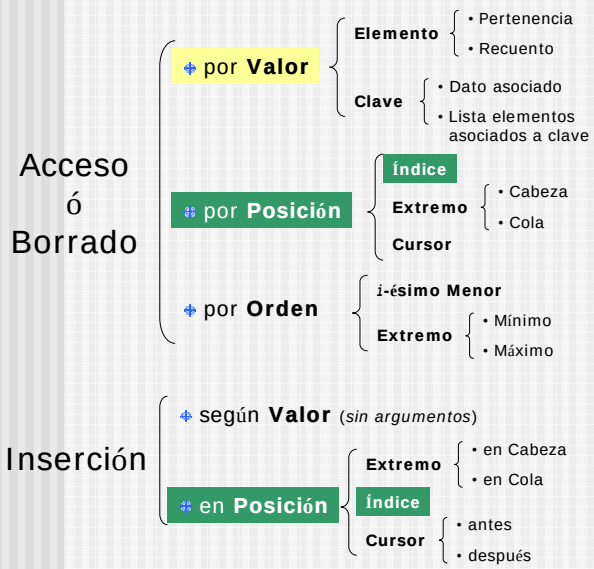
Colección (Bag, MultiSet)



- ✓ **Sin relación**
- ✓ **Elementos atómicos**
- ✓ **Req: Equivalencia**
- ✓ **Elementos repetidos**
- ✓ **Otras operaciones:**
  - ➢ **Unión**
  - ➢ **Recorrido**
  - ➢ **Borrar todos**

21

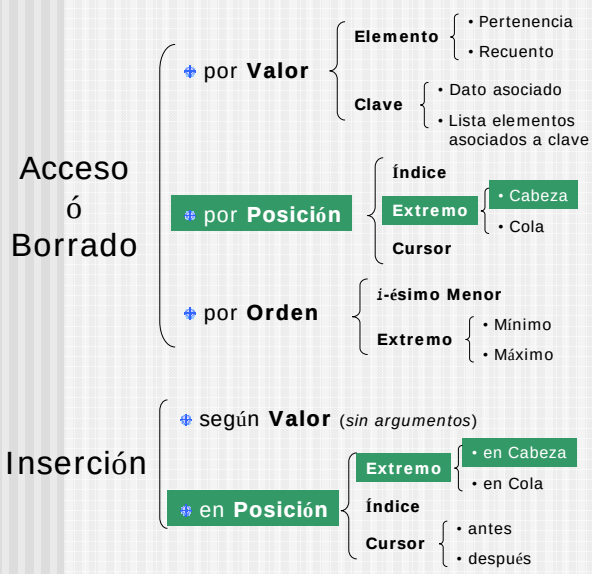
Lista indexada (List)



- ✓ **Relación Orden Externo**
- ✓ **Elementos atómicos**
- ✓ **Req: Ninguno**
- ✓ **Elementos repetidos**
- ✓ **Otras operaciones:**
  - ➢ **Concatenación**
  - ➢ **Recorrido**
  - ➢ **Búsqueda**
  - ➢ **Ordenación**

22

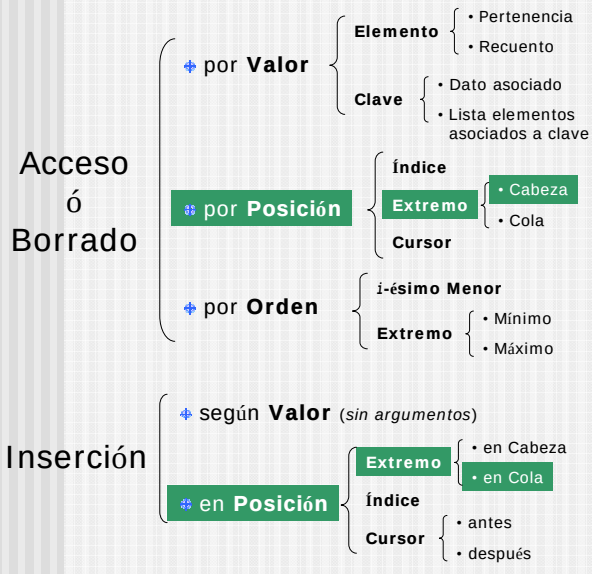
Pila (Stack)



- ✓ **Relación Orden Externo**
- ✓ **Elementos atómicos**
- ✓ **Req: Ninguno**
- ✓ **Elementos repetidos**
- ✓ **Estructura LIFO (Last In, First Out)**
- ✓ **Usos:**
  - ➢ **Gestión memoria**
  - ➢ **Recursivo → Iterativo**
  - ➢ **Traducción expr. matemáticas**
  - ➢ **Lenguages FORTH, Postscript, Java VM**

23

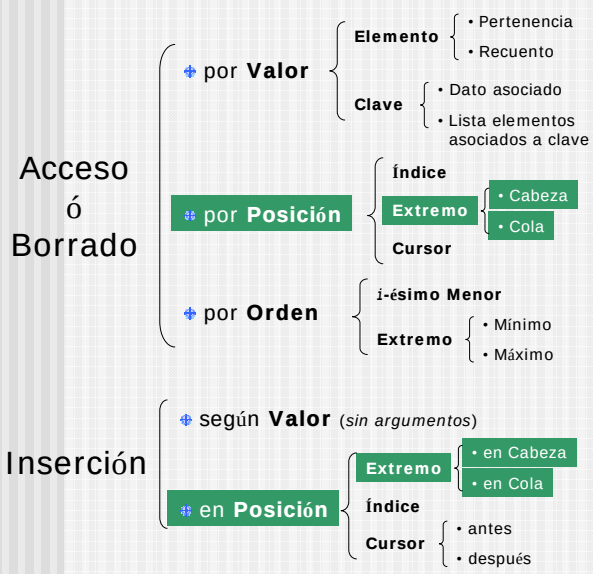
Cola (Queue)



- ✓ **Relación Orden Externo**
- ✓ **Elementos atómicos**
- ✓ **Req: Ninguno**
- ✓ **Elementos repetidos**
- ✓ **Estructura FIFO (First In, First Out)**
- ✓ **Usos:**
  - ➢ **Buffers**
  - ➢ **Algoritmos recorrido en anchura en árboles y grafos.**

24

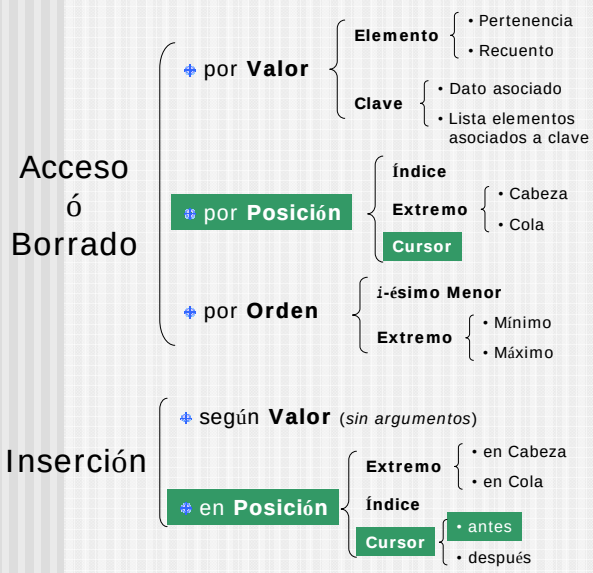
Bicola (Deque)



- ✓ **Relación Orden Externo**
- ✓ **Elementos atómicos**
- ✓ **Req: Ninguno**
- ✓ **Elementos repetidos**
- ✓ **Usos:**
  - ➢ **Buffers**
  - ➢ **Algoritmos recorrido en anchura en árboles y grafos.**

25

Lista secuencial (I) (Iterator)



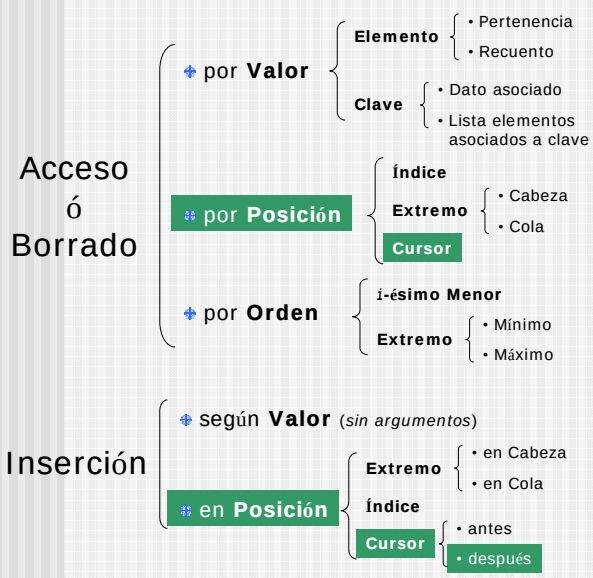
- ✓ **Relación Orden Externo**
- ✓ **Elementos atómicos**
- ✓ **Req: Ninguno**
- ✓ **Elementos repetidos**
- ✓ **Otras operaciones:**
  - ➢**Mover cursor**
  - ➢**Cursor fuera lista**
  - ➢**Concatenación**

Variante 1 (*notepad*)

- ✓ **Fuera de lista = despues del último elemento.**
- ✓ **Tras borrado cursor pasa a elemento siguiente.**

26

Lista secuencial (II) (Iterator)



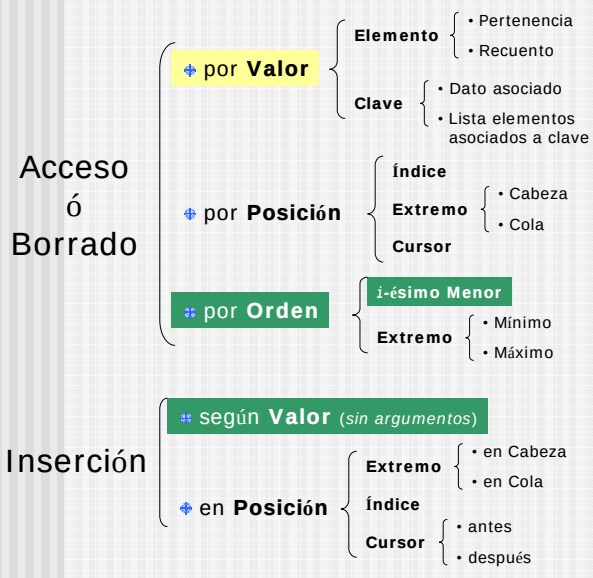
- ✓ **Relación Orden Externo**
- ✓ **Elementos atómicos**
- ✓ **Req: Ninguno**
- ✓ **Elementos repetidos**
- ✓ **Otras operaciones:**
  - ➢ **Mover cursor**
  - ➢ **Cursor fuera lista**
  - ➢ **Concatenación**

Variante 2

- ✓ **Fuera de lista = antes del primer elemento.**
- ✓ **Tras borrado cursor pasa a elemento anterior.**

27

Lista ordenada (Sorted List)



- ✓ **Relación Orden Interno**
- ✓ **Elementos atómicos**
- ✓ **Req: Comparación**
- ✓ **Elementos repetidos**
- ✓ **Otras operaciones:**
  - ➢ **Fusión**
  - ➢ **Recorrido**
  - ➢ **Búsqueda**

28

Cola de prioridad (Priority queue)

Acceso ó Borrado

- por **Valor**
  - **Elemento**
    - Pertenencia
    - Recuento
  - **Clave**
    - Dato asociado
    - Lista elementos asociados a clave
- por **Posición**
  - **Índice**
  - **Extremo**
    - Cabeza
    - Cola
  - **Cursor**
- por **Orden**
  - ***i*-ésimo Menor**
  - **Extremo**
    - Mínimo
    - Máximo

Inserción

- según **Valor** (*sin argumentos*)
- en **Posición**
  - **Extremo**
    - en Cabeza
    - en Cola
  - **Índice**
  - **Cursor**
    - antes
    - después

- ✓ **Relación Orden Interno**
- ✓ **Elementos son pares prioridad - dato**
- ✓ **Req: Comparación**
- ✓ **Variante: Cola de prioridad de máximos**
- ✓ **Otras operaciones:**
  - ➢ **Modificar prioridad**
  - ➢ **Fusión**
- ➢ **Usos:**
  - ➢ **Gestión procesos**
  - ➢ **Ord. Montículos**
  - ➢ **Alg. Dijkstra**

29

Tabla (Map, Dictionary)

Acceso ó Borrado

- por **Valor**
  - **Elemento**
    - Pertenencia
    - Recuento
  - **Clave**
    - Dato asociado
    - Lista elementos asociados a clave
- por **Posición**
  - **Índice**
  - **Extremo**
    - Cabeza
    - Cola
  - **Cursor**
- por **Orden**
  - ***i*-ésimo Menor**
  - **Extremo**
    - Mínimo
    - Máximo

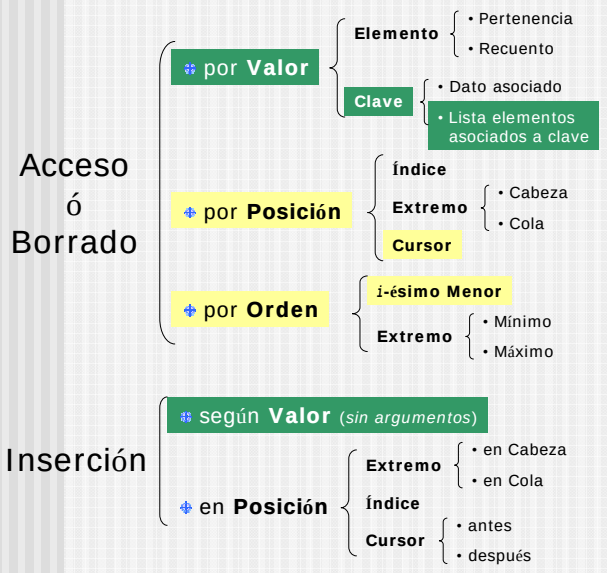
Inserción

- según **Valor** (*sin argumentos*)
- en **Posición**
  - **Extremo**
    - en Cabeza
    - en Cola
  - **Índice**
  - **Cursor**
    - antes
    - después

- ✓ **Mapa + Orden Interno en claves**
- ✓ **Elementos son pares clave - dato**
- ✓ **Req: Comparación claves**
- ✓ **Otras operaciones:**
  - ➢ **Recorrido ordenado basado en cursor**
  - ➢ **Modificación dato**
- ➢ **Usos: BD Relacionales**

30

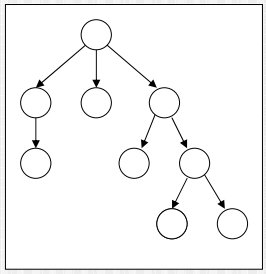
Diccionario (MultiMap)



- ✓ **Orden Interno en claves**
- ✓ **Elementos son pares clave - dato**
- ✓ **Req: Comparación claves**
- ✓ **Otras operaciones:**
  - ➢ **Recorrido ordenado basado en cursor**
  - ➢ **Reemplazamiento par clave - dato**

31

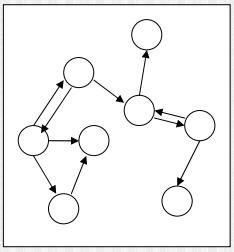
Directorio (General Tree, Directory)



- Un **directorio** representa una colección de elementos con una **relación de jerarquía** entre ellos. Ejemplo típico: Sistema de ficheros (SF).
- Es habitual que los elementos hojas se interpreten o contengan información distinta a la del resto de elementos (En un SF las hojas serían ficheros y los nodos internos directorios).
- Las operaciones de acceso suelen **basarse en cursor**: Hay un elemento distinguido y se puede acceder a su padre y sus hijos. (En un SF el directorio actual).
- Las implementaciones más habituales son las **enlazadas**, en particular la implementación padre-hijo-hermano.
- Atención: La palabra "arbol" puede usarse con dos significados distintos:
  - Un TAD con relación de jerarquía: El TAD Directorio
  - Una representación (montículo, ABB, árbol AVL, etc.) útil para algunos TADs (como por ejemplo el TAD Conjunto, TAD Tabla, etc.)

32

Grafo (Graph)



- Un grafo representa una colección de elementos (vértices) sobre los que existe una **relación de vecindad** definida por un conjunto de aristas.
- Ejemplo típico: World Wide Web → Colección de páginas web relacionadas entre sí por los hipervínculos de una página a otra.
- Las operaciones habituales son la inserción y borrado de aristas, inserción y borrado de nodos, los recorridos y operaciones más especializadas (camino mínimo, etc.)
- Las implementaciones mas comunes son:
  - Listas de adyacencia
  - Matriz de adyacencia

33

Otros TADs

- **TAD Array** ó TAD Vector: Operaciones de acceso por indice y reemplazar elemento (también por indice).
- **TAD Disjoint-Set**: Representa elementos clasificados en categorías (una serie de conjuntos disjuntos, donde cada elemento sólo pertenece a un solo conjunto), de manera que se establece una **partición** sobre ellos.

  A priori este tipo de información se podría representar mediante un TAD diccionario (las claves serían las categorías y los valores los elementos, en este caso sin repetición), pero aquí la intención es simplificarlo de manera que:
  - Cada categoría (conjunto) se represente no de manera explícita sino mediante un elemento cualquiera que pertenezca a ella (representante)
  - Las únicas operaciones fundamentales sean:
  - **Creación** a partir de una colección de elementos (cada elemento define una categoría)
  - **Búsqueda** (Find): Determinar a que categoría pertenece un elemento (se devuelve un representante de esa categoría)
  - **Union**: Unir dos categorías en una sola.

34