

Estructuras de Datos y Algoritmos

TAD fundamentales

LECCIÓN 5

© All wrongs reversed – bajo licencia [CC-BY-NC-SA 4.0](#)



Universidad
Zaragoza

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, España

Grado en Ingeniería Informática
UNIVERSIDAD DE ZARAGOZA



Adaptadas de diapositivas de Javier Campos

Índice

1 TAD contenedores

- TAD contenedores sin relación entre los elementos
- TAD contenedores con relación entre elementos

2 Iterador en contenedores

3 Biblioteca *Standard Template Library* de C++

Índice

1 TAD contenedores

- TAD contenedores sin relación entre los elementos
- TAD contenedores con relación entre elementos

2 Iterador en contenedores

3 Biblioteca *Standard Template Library* de C++

TAD contenedores

- TAD que actúan como *agregaciones de elementos de un mismo tipo*, junto con las operaciones para manipularlas
- Algunos no precisan almacenar relaciones¹ entre los elementos
 - Conjunto
 - Multiconjunto (también llamado bolsa o saco)
 - Diccionario (también llamado mapa o tabla)
 - Etcétera
- Algunos otros sí almacenan relaciones¹ entre los elementos
 - Pilas, colas, listas
 - Árboles, grafos, etcétera

¹Relaciones de orden, de secuencia, jerarquías, o de otro tipo.

TAD contenedores sin relación entre elementos

Conjunto

- **Colección de 0 o más elementos**
- **Sin elementos repetidos**
- Operaciones típicas:
 - crear: crea un conjunto vacío (sin elementos)
 - añadir: dado un elemento, si no pertenece al conjunto lo añade
 - pertenece?: dado un elemento, comprueba si se encuentra en el conjunto
 - quitar: dado un elemento, si pertenece al conjunto lo elimina
 - cardinal: devuelve el número de elementos en el conjunto
 - esVacio?: comprueba si el conjunto esta vacío (no contiene ningún elemento)
 - eliminarTodos: elimina todos los elementos del conjunto, dejándolo vacío
 - ...
- Otras operaciones frecuentes: relativas a conjuntos de matemáticas
 - Unión ($\cup$)
 - Diferencia ($\setminus$)
 - Intersección ($\cap$)

TAD contenedores sin relación entre elementos

Multiconjunto

- **Colección de 0 o más elementos**
- **Puede contener elementos repetidos** (es una generalización de conjunto)
- Operaciones típicas:
 - crear: crea un multiconjunto vacío (sin elementos)
 - añadir: dado un elemento, si no pertenece al multiconjunto lo añade
 - pertenece?: dado un elemento, comprueba si se encuentra en el multiconjunto
 - multiplicidad: devuelve el nº de ejemplares de un elemento dado en el multiconjunto
 - quitar: dado un elemento, si pertenece al multiconjunto, elimina una de sus apariciones
 - cardinal: devuelve el nº de elementos en el multiconjunto (considerando multiplicidad)
 - esVacio?: comprueba si el multiconjunto esta vacío (no contiene ningún elemento)
 - eliminarTodos: elimina todos los elementos del multiconjunto, dejándolo vacío
 - ...
- Otras operaciones frecuentes: relativas a multiconjuntos de matemáticas
 - Unión ($\cup$)
 - Diferencia ($\setminus$)
 - Intersección ($\cap$)

TAD contenedores sin relación entre elementos

Diccionario

- **Conjunto de 0 o más elementos formados por pares (clave, valor)**

donde **no** se permiten claves repetidas

- Formalmente, son una función que aplica el tipo clave en el tipo valor:
 $Conj : clave \mapsto valor$ (es decir, cada clave tiene asociada un valor)

- **Las claves no pueden cambiar, pero los valores sí**

- **Operaciones de acceso y manipulación por clave**

- Operaciones típicas:

- crear: crea un diccionario sin elementos
- añadir: dada una clave y un valor, añade el par (clave,valor) al diccionario
- pertenece?: dada una clave devuelve un booleano que indica si la clave está
- obtenerValor: dada una clave devuelve el valor asociado a ella
- quitar: dada una clave la borra del diccionario junto con su valor
- cardinal: devuelve el número de pares (clave,valor) en el diccionario
- esVacio?: comprueba si el diccionario esta vacío
- ...

- Situaciones de error: especificarlas e implementar de forma robusta

- Por ejemplo, si se intenta obtener el valor de una clave no existente en el diccionario

TAD contenedores con relación entre elementos

Tipos

TAD lineales

- Dominio de valores: **secuencia** de elementos de un mismo tipo (una dimensión), e.g., $e_1 e_2 \dots e_n$ (los datos de una secuencia **están en orden**)
- Mucha variedad de TAD lineales:
 - Con o sin elementos repetidos
 - Distintos criterios de orden en la secuencia (e.g., orden alfabético, por inserción, ...)
 - Operaciones típicas: crear, añadir, esVacía?, quitar, pertenece?, tamaño, ...
 - Operaciones de acceso y recorrido (iteradores): según el orden en la secuencia
 - primero, último, iniciarIterador, existeSiguiente?, siguiente, avanza
 - Pueden ser operaciones:
 - Limitadas a determinados extremos de la secuencia (TAD pila, cola, ...)
 - Relativas a la posición en la secuencia (listas con acceso por posición)
 - Relativas al último elemento accedido o insertado (listas con punto de interés)

TAD no lineales

- Árboles (jerarquía), grafos, ... *(los veremos a partir de la lección 11)*

TAD contenedores

¿Qué contenedor elijo? La tabla de decisión

- Elegir mal la estructura es una de las familias de error más frecuentes en el examen (F3 elección de estructura)
- **Rutina del curso:** mantendremos una **tabla de decisión** estructura $\leftrightarrow$ requisito
 - hoy arranca con los contenedores fundamentales
 - cada nueva estructura (árboles, tries, montículos, hash) añadirá su fila
 - se cierra en la lección 19: será vuestro “mapa” para el examen
- La pregunta correcta nunca es “¿qué estructura me gusta?”, sino “¿qué **requisito** (de acceso y de coste) me piden?”

Tabla de decisión: estructura ↔ requisito

Estructura	Coste de las operaciones clave	Elegirla cuando...	Antipatrón
Pila	apilar, desapilar, cima: $O(1)$	solo importa el último en llegar (LIFO): deshacer, llamadas, delimitadores	buscar o recorrer «dentro» de una pila
Cola	encolar, desencolar, primero: $O(1)$	importa el orden de llegada (FIFO): turnos, esperas	usarla cuando se necesita acceso por clave
Lista	insertar/borrar en el punto de interés: $O(1)$; buscar: $O(n)$	colección recorrible con orden posicional	lista donde el requisito pide búsqueda eficiente
Diccionario	según implementación ($\rightarrow$ ABB, AVL, trie, hash)	acceso por clave	fixar la implementación sin leer el requisito de coste

La tabla se irá completando a lo largo del curso: cópiala y ve rellenándola tú también.

TAD contenedores

Antipatrón: elegir el contenedor sin el requisito de coste delante

- Primer ejemplar de la **galería de antipatronos** (uno por estructura, a lo largo del curso)

× Así no

“Guardo los clientes en una lista, que es lo que conozco”:

- se elige la estructura **antes** de leer qué operaciones se piden y con qué coste
- luego el coste no cumple el requisito y hay que rehacer el diseño (o se entrega mal)

✓ Así sí

Leer primero el requisito y **después** elegir:

- *“atender por orden de llegada”* → cola
- *“deshacer la última acción”* → pila
- *“consultar por identificador, eficientemente”* → diccionario (y su implementación, por el coste exigido)

Regla: la estructura la elige el **requisito**, no la costumbre

Índice

1 TAD contenedores

2 Iterador en contenedores

3 Biblioteca *Standard Template Library* de C++

Iterador en contenedores

- Operaciones típicas en los contenedores: crear, añadir, esVacio?, quitar, pertenece?, cardinal

Iterador en contenedores

- Operaciones típicas en los contenedores: crear, añadir, esVacio?, quitar, pertenece?, cardinal

¿Cómo podemos ...

- ... *mostrar todos los elementos por pantalla?*
- ... *conocer todos los elementos que cumplan cierta condición?*

Iterador en contenedores

- Operaciones típicas en los contenedores: crear, añadir, esVacio?, quitar, pertenece?, cardinal

¿Cómo podemos ...

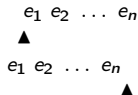
- *... mostrar todos los elementos por pantalla?*
- *... conocer todos los elementos que cumplan cierta condición?*

Iterador

- **Acceso a todos y cada uno de los elementos de forma eficiente**
 - Sin acceder a un elemento más de una vez
 - Sin dejarse ningún elemento sin acceder

Iterador en contenedores

- Un iterador definido sobre un contenedor permite **visitar todos sus elementos, uno tras otro**
 - En orden indeterminado, o no (dependiendo del TAD, ya veremos...)
 - Permiten implementar algoritmos de recorrido o búsqueda
- **Puede interpretarse como cursor o índice** para recorrer la colección
 - El cursor apunta al siguiente (el siguiente elemento a visitar), dentro del recorrido
 - Inicialmente, el siguiente es siempre el primer elemento a visitar
 - Cuando ya se ha visitado el último elemento, no hay siguiente



Iterador en contenedores

Operaciones básicas

- `iniciarIterador`

- Prepara el iterador y su cursor para que el siguiente elemento a visitar sea el primero (situación de no haber visitado ningún elemento)

- `existeSiguiete?`

- Devuelve falso si ya se ha visitado el último elemento
- Devuelve cierto en caso contrario

- `siguiete`

- Devuelve el siguiete elemento
- Operación parcial: sólo definida si `existeSiguiete?` es verdad

- `avanza`

- Avanza el cursor
- Operación parcial: sólo definida si `existeSiguiete?` es verdad

Todas estas operaciones habrá que especificarlas...

Iterador en contenedores

Implementación (en pseudocódigo) de operaciones básicas

... e implementarlas

procedimiento iniciarIterador(*e/s* c: tipo_del_contenedor)
{Prepara el cursor del contenedor c para que el elemento señalado sea el 1º del contenedor (situación de no haber visitado antes ningún elemento)}

función existeSiguiente(*c*: tipo_del_contenedor) **devuelve** booleano
{Devuelve falso si ya se ha visitado el último elemento del contenedor (el cursor no señala a ningún elemento), devuelve cierto en caso contrario}

procedimiento siguiente(*e/s* c: tipo_del_contenedor;
 sal e: elemento_base_del_contenedor;
 sal error: booleano)
{Si no existeSiguiente(c) entonces devuelve el valor verdad en el parámetro error. En caso contrario (es decir, existeSiguiente(c)):
1. *devuelve falso en el parámetro error;*
2. *devuelve en e el valor del elemento señalado por el cursor;*
3. *avanza el cursor para señalar a otro elemento del contenedor que todavía no se haya visitado}*

Iterador en contenedores

Uso

```
iniciarIterador(c);  
mientrasQue existeSiguiente(c) hacer  
    siguiente(c, e, error);  
    ...  
    {utilizar el valor de e para algo}  
fmq
```

Atención

- **No se debe modificar el valor del TAD contenedor** (con las operaciones de añadir, borrar o modificar de que disponga) **mientras se están recorriendo sus elementos** con las operaciones de un iterador

Índice

1 TAD contenedores

2 Iterador en contenedores

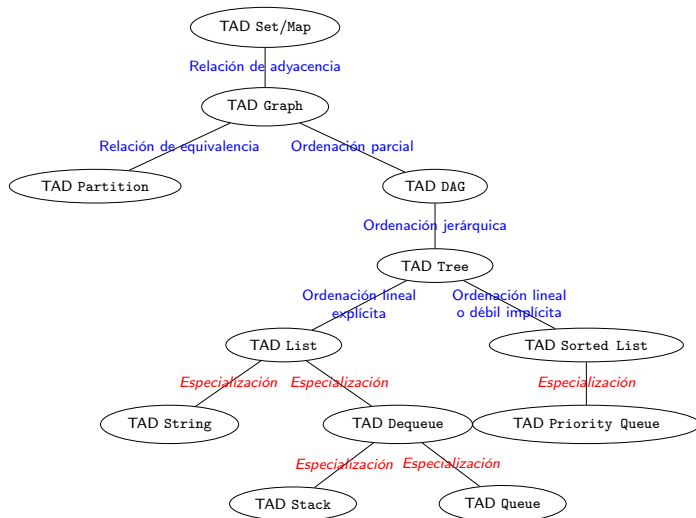
3 Biblioteca *Standard Template Library* de C++

Biblioteca *Standard Template Library* de C++

- Biblioteca de C++ que **proporciona TAD, estructuras de datos y algoritmos de uso frecuente**
- Ejemplo de definición/implementación de TAD (como los contenedores mencionados antes) reutilizables
- **No siempre encontraremos lo que necesitamos: eficiencia**
 - Para aumentar versatilidad y reutilizabilidad, puede sacrificar eficiencia y usar más recursos de los necesarios
- **No existe una biblioteca equivalente disponible en todos los lenguajes de programación** (aunque sí en muchos)

Biblioteca *Standard Template Library* de C++

Fragmento ilustrativo



Biblioteca *Standard Template Library* de C++

STL y los objetivos de la asignatura

“El alumno aprenderá a diseñar e implementar TAD para que sean reutilizables, eficientes y robustos, y a implementarlos garantizando dichas propiedades”

- El objetivo **NO ES** conocer y utilizar la STL de C++ ni bibliotecas similares
- Salvo que se indique explícitamente lo contrario, para la implementación de TAD
no se permitirá utilizar los TAD disponibles en la STL o similares

Estructuras de Datos y Algoritmos

TAD fundamentales

LECCIÓN 5

© All wrongs reversed – bajo licencia [CC-BY-NC-SA 4.0](#)



Universidad
Zaragoza

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, España

Grado en Ingeniería Informática
UNIVERSIDAD DE ZARAGOZA

