Introducción al Diseño

Algoritmos y Estructuras de Datos II

Introducción al diseño

Cualidades del software

Fundamentales:

- Correcto con respecto a una especificación.

Más o menos importantes, dependiendo del **contexto de uso**:

- Eficiente (tiempo, memoria, consumo de energía, ...).
- Reutilizable.
- Extensible / modificable.
- Usable.
- Legible.
- Predecible.
- ...

El **diseño** consiste en organizar el programa de tal manera que cumpla con las cualidades requeridas, en algún contexto de uso.

Introducción al diseño

Receta para el desastre

Entender un tipo a través de su **estructura**.

Ejemplo

"Un diccionario es una `secu<tupla<clave, valor>>`." ☹

Si queremos modificar el diccionario para saber a qué hora se insertó cada clave por última vez:

- Hay que cambiar la estructura a:
  `secu<tupla<clave, tupla<valor, hora>>>`.
- Hay que cambiar las inserciones para que agreguen la hora.
- Hay que cambiar todas las búsquedas para quedarse sólo con el valor.
- Estos cambios pueden estar desperdigados por el programa.

Introducción al diseño

Diseño por contratos

Entender un tipo a través de su **interfaz**.

Ejemplo

"Un diccionario provee operaciones: ☺

1. Crear un diccionario vacío.
2. Asociar una clave a un valor.
3. Buscar una clave."

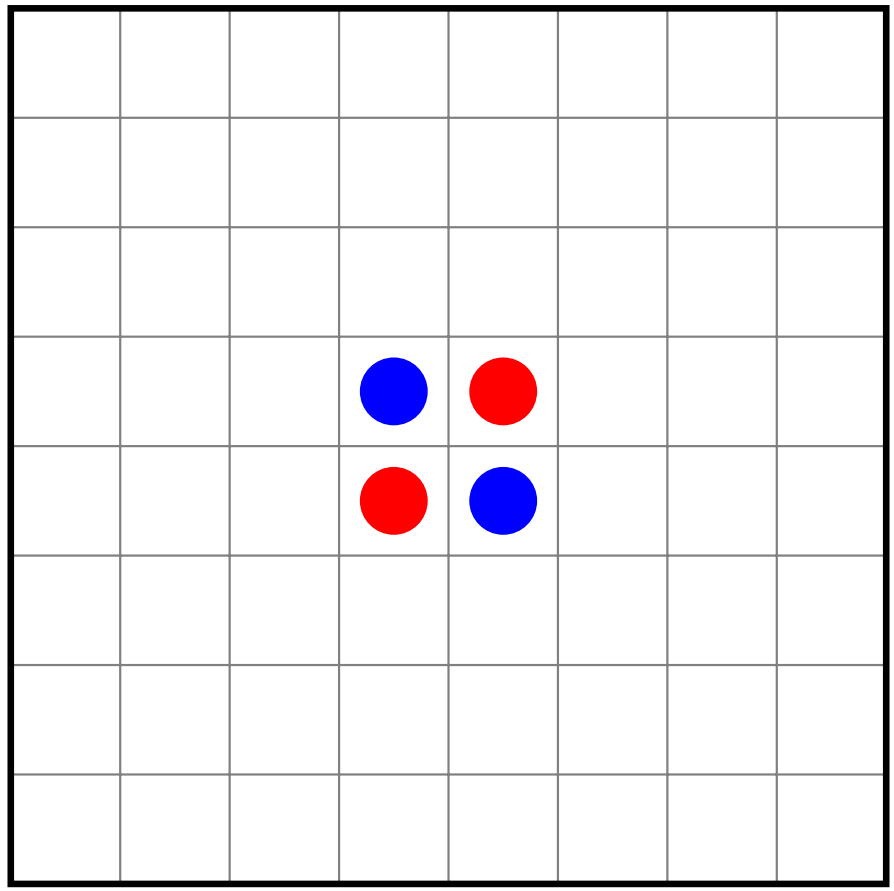
Si queremos modificar el diccionario para saber a qué hora se insertó cada clave por última vez:

- Basta con modificar la implementación (privada) del diccionario.
- La interfaz se extiende, pero las operaciones anteriores siguen funcionando.

Ejemplo: Reversi
(descripción informal)

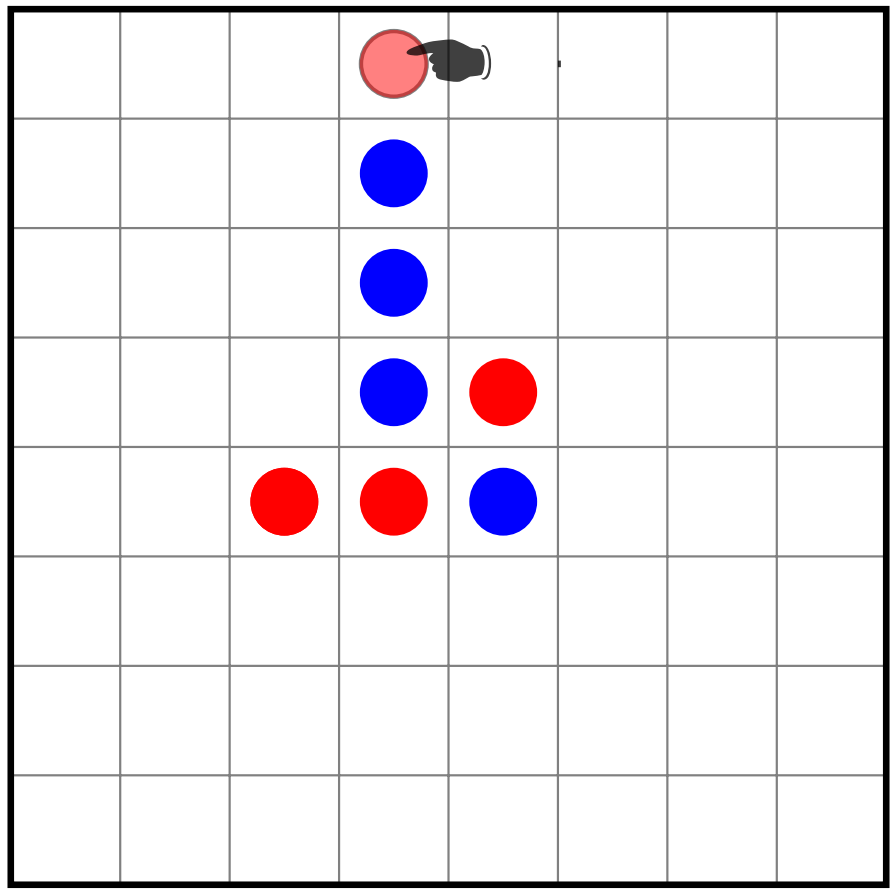
Reversi: descripción

Dos jugadores (rojo y azul) se alternan para jugar fichas sobre un tablero de $n \times n$ (con n par). El tablero empieza así:



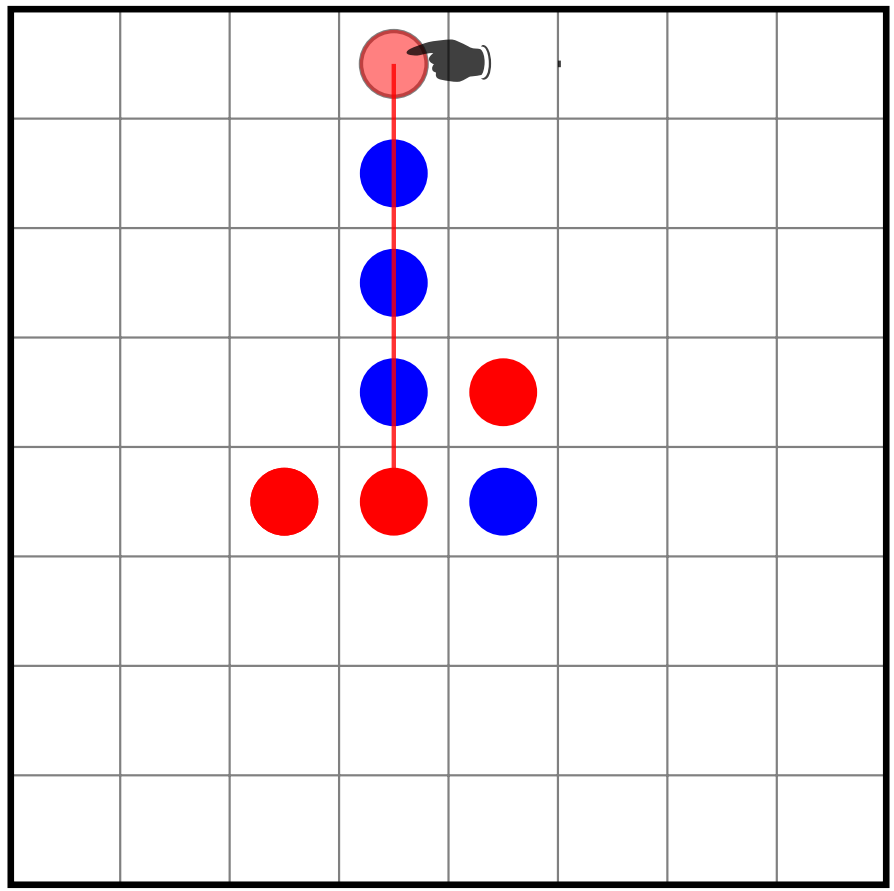
Reversi: descripción

Si un jugador juega una ficha que encierra una línea de fichas del oponente opuesto entre dos fichas propias, las fichas del oponente se vuelven del color propio.



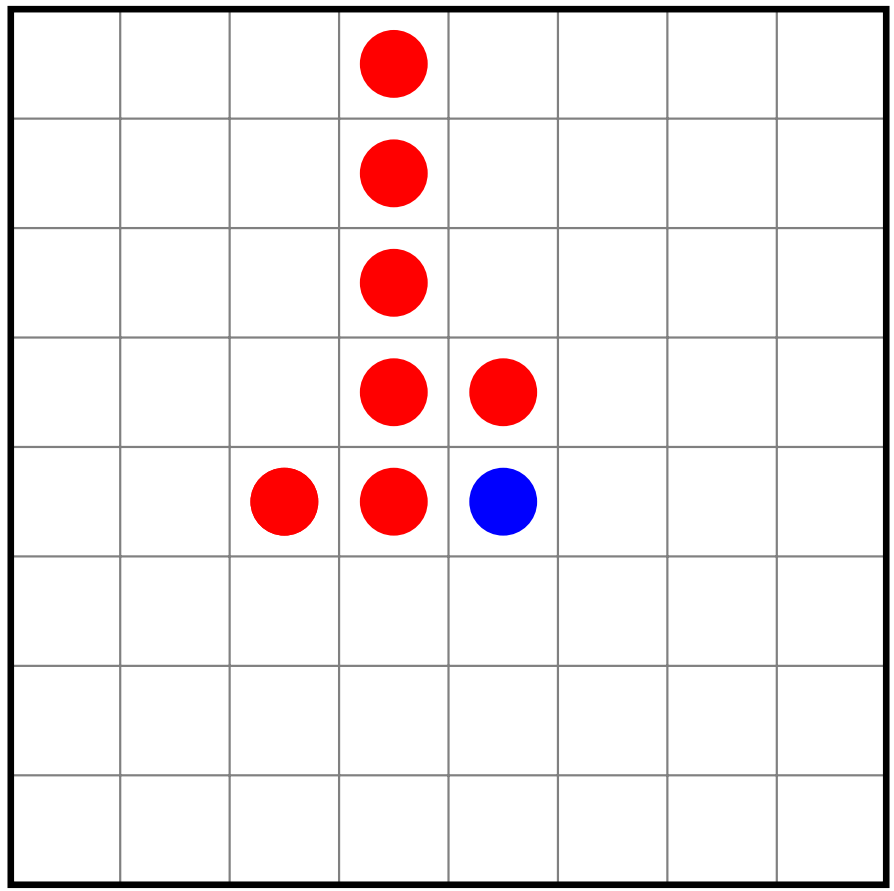
Reversi: descripción

Si un jugador juega una ficha que encierra una línea de fichas del oponente opuesto entre dos fichas propias, las fichas del oponente se vuelven del color propio.



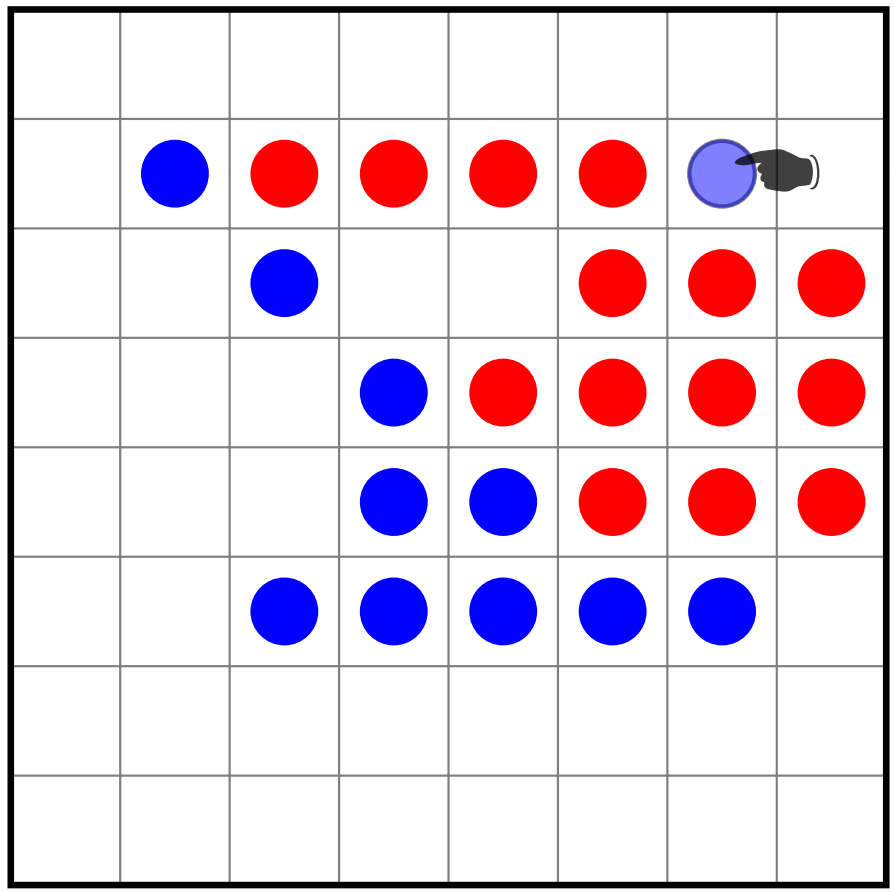
Reversi: descripción

Si un jugador juega una ficha que encierra una línea de fichas del oponente opuesto entre dos fichas propias, las fichas del oponente se vuelven del color propio.



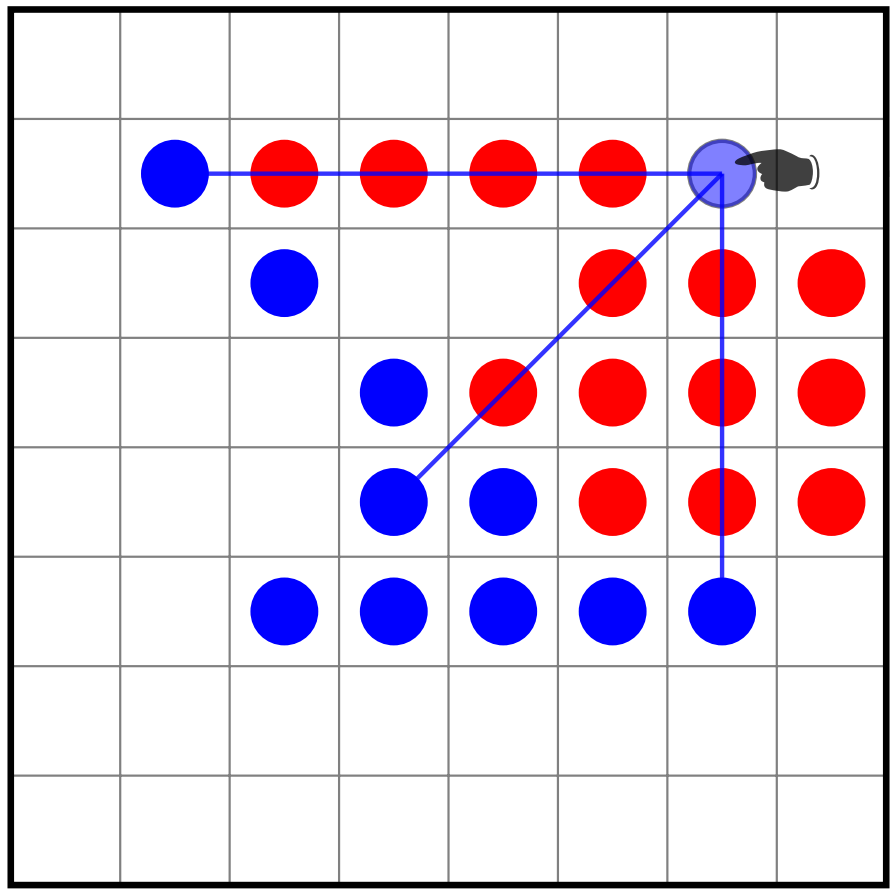
Reversi: descripción

Una ficha puede encerrar varias líneas simultáneamente:



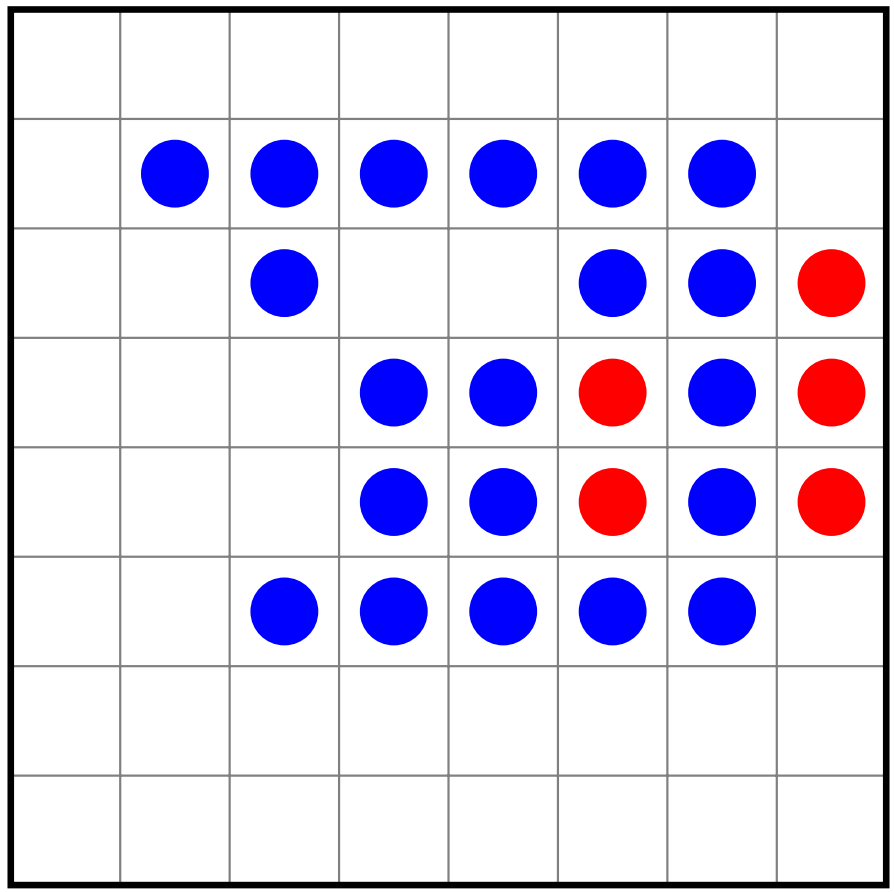
Reversi: descripción

Una ficha puede encerrar varias líneas simultáneamente:



Reversi: descripción

Una ficha puede encerrar varias líneas simultáneamente:



Reversi: descripción

- Un jugador sólo puede jugar una ficha en una posición si esa jugada da vuelta al menos una ficha del oponente.
- En esta variante del reversi, si un jugador no tiene movidas válidas, termina el juego.
- Cuando termina el juego, gana el jugador que tiene más fichas de su color sobre el tablero.

A donde queremos llegar...

```
struct Coord { int x; int y; };
enum Jugador { Rojo, Azul };
enum Celda { FichaRoja, FichaAzul, Nada };
typedef std::vector<std::vector<Celda>> Tablero;

class Reversi {
public:
  Reversi(int n);
  Jugador turno() const;
  void jugar(Coord c);
  int puntaje(Jugador j) const;
  void mostrar(Tablero& t) const;
private:
  ...
};
```

Entreacto

¿Porqué TADs?

Tipo de dato abstracto

Este artículo o sección necesita **referencias** que aparezcan en una publicación acreditada.
Este aviso fue puesto el 14 de febrero de 2015.

Un **tipo de dato abstracto** (**TDA**) o **tipo abstracto de datos** (**TAD**) es un modelo matemático compuesto por una colección de operaciones definidas sobre un conjunto de datos para el modelo.

Índice [ocultar]

Introducción [editar]

En el mundo de la programación existen diversos lenguajes de programación que se han ido creando con el paso del tiempo y que se han perfeccionado debido a las necesidades de los programadores de la época a la que pertenecen. Los primeros lenguajes de programación eran de tipo lineal, ya que un programa se recorría desde un punto marcado como Inicio hasta llegar a un punto Fin. Con el tiempo se fueron creando nuevos lenguajes y en nuestros días los más utilizados son los llamados "orientados a objetos".

Los lenguajes orientados a objetos (LOO) tienen la característica de que no son lenguajes lineales, sino que se forman de diversas funciones, las cuales son llamadas en el orden en que el programa mismo las pide o el usuario determina. Para entender mejor cómo funcionan los lenguajes orientados a objetos, vamos a introducir un concepto fundamental en las

Abstract data type

From Wikipedia, the free encyclopedia

Not to be confused with algebraic data type.

This article has multiple issues. Please help improve it or discuss these issues on the talk page. *(Learn how and when to remove these template messages)* [hide]

- This article **possibly contains original research**. *(March 2015)*
- This article **needs additional citations for verification**. *(May 2009)*

In computer science, an **abstract data type** (**ADT**) is a mathematical model for data types, where a data type is defined by its behavior (semantics) from the point of view of a *user* of the data, specifically in terms of possible values, possible operations on data of this type, and the behavior of these operations. This contrasts with data structures, which are concrete representations of data, and are the point of view of an implementer, not a user.

Formally, an ADT may be defined as a "class of objects whose logical behavior is defined by a set of values and a set of operations";[1] this is analogous to an algebraic structure in mathematics. What is meant by "behavior" varies by author, with the two main types of formal specifications for behavior being *axiomatic (algebraic) specification* and an *abstract model*;[2] these correspond to axiomatic semantics and operational semantics of an abstract machine, respectively. Some authors also include the computational complexity ("cost"), both in terms of time (for computing operations) and space (for representing values). In practice many common data types are not ADTs, as the abstraction is not perfect, and users must be aware of issues like arithmetic overflow that are due to the representation. For example, integers are often stored as fixed width values (32-bit or 64-bit binary numbers), and thus experience integer overflow if the maximum value is exceeded.

ADTs are a theoretical concept in computer science, used in the design and analysis of algorithms, data structures, and

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

In other projects

En general

Un TAD está dado por:

- Un conjunto de valores.
- Operaciones para manipularlos.
- Relaciones entre dichas operaciones, por ejemplo: `pop(push(x, pila)) = x`.

En esta materia

Distinguimos varias clases de operaciones:

- Generadores.
- Observadores básicos.
- Otras operaciones.

Usamos un formalismo específico.

Ejemplo: Reversi
(especificación formal)

Reversi: especificación

TAD Coord es Tupla$\langle x$: int, y: int$\rangle$
TAD Jugador es Enumerado$\langle$ROJO, AZUL$\rangle$

TAD Reversi

observadores básicos

...

Reversi: especificación

TAD Coord es Tupla⟨x: int, y: int⟩
TAD Jugador es Enumerado⟨ROJO, AZUL⟩

TAD Reversi

observadores básicos

dim	:	reversi	$\longrightarrow$	nat
rojas	:	reversi	$\longrightarrow$	conj(coord)
azules	:	reversi	$\longrightarrow$	conj(coord)
turno	:	reversi	$\longrightarrow$	jug

generadores

...

Reversi: especificación

TAD Coord es Tupla⟨x: int, y: int⟩
TAD Jugador es Enumerado⟨ROJO, AZUL⟩

TAD Reversi

observadores básicos

dim	:	reversi	$\longrightarrow$	nat
rojas	:	reversi	$\longrightarrow$	conj(coord)
azules	:	reversi	$\longrightarrow$	conj(coord)
¿turno?	:	reversi	$\longrightarrow$	jug

generadores

...

Reversi: especificación

TAD Coord es Tupla$\langle x$: int, y: int$\rangle$
TAD Jugador es Enumerado$\langle$ROJO, AZUL$\rangle$

TAD Reversi

observadores básicos

dim	:	reversi	$\longrightarrow$	nat
rojas	:	reversi	$\longrightarrow$	conj(coord)
azules	:	reversi	$\longrightarrow$	conj(coord)
~~turno~~	~~:~~	~~reversi~~	~~$\longrightarrow$~~	~~jug~~

generadores

...

Reversi: especificación

TAD COORD es TUPLA⟨x: int, y: int⟩
TAD JUGADOR es ENUMERADO⟨ROJO, AZUL⟩

TAD REVERSI

observadores básicos

dim : reversi $\longrightarrow$ nat
rojas : reversi $\longrightarrow$ conj(coord)
azules : reversi $\longrightarrow$ conj(coord)

generadores

nuevo : nat n $\longrightarrow$ reversi
$\{n > 0 \wedge (\exists k : \text{nat})(n =_{\text{obs}} 2 * k)\}$
jugar : reversi $r \times$ coord c $\longrightarrow$ reversi $\{\text{puedeJugar?}(r, c)\}$

otras operaciones

puedeJugar? : reversi $r \times$ coord c $\longrightarrow$ bool
turno : reversi r $\longrightarrow$ jug

axiomas

$$\begin{aligned}
&\text{dim}(\text{nuevo}(n)) && \equiv \ ...\\
&\text{dim}(\text{jugar}(r,\ c)) && \equiv \ ...\\
&\text{rojas}(\text{nuevo}(n)) && \equiv \ ...\\
&\text{rojas}(\text{jugar}(r,\ c)) && \equiv \ ...\\
&\text{azules}(\text{nuevo}(n)) && \equiv \ ...\\
&\text{azules}(\text{jugar}(r,\ c)) && \equiv \ ...
\end{aligned}$$

axiomas

$$\begin{array}{lcl}
\text{dim}(\text{nuevo}(n)) & \equiv & n \\
\text{dim}(\text{jugar}(r,\, c)) & \equiv & \text{dim}(r) \\
 & & \\
\text{rojas}(\text{nuevo}(n)) & \equiv & \{\langle \frac{n}{2}-1, \frac{n}{2}-1\rangle, \langle \frac{n}{2}, \frac{n}{2}\rangle\} \\
\text{rojas}(\text{jugar}(r,\, c)) & \equiv & \textbf{if } \text{turno}(r) = \texttt{ROJO} \\
 & & \quad \textbf{then } \text{rojas}(r) \cup \{c\} \cup \text{invertidas}(r,\, c) \\
 & & \quad \textbf{else } \text{rojas}(r) \setminus \text{invertidas}(r,\, c) \\
 & & \textbf{fi} \\
 & & \\
\text{azules}(\text{nuevo}(n)) & \equiv & \{\langle \frac{n}{2}-1, \frac{n}{2}\rangle, \langle \frac{n}{2}, \frac{n}{2}-1\rangle\} \\
\text{azules}(\text{jugar}(r,\, c)) & \equiv & \textbf{if } \text{turno}(r) = \texttt{AZUL} \\
 & & \quad \textbf{then } \text{azules}(r) \cup \{c\} \cup \text{invertidas}(r,\, c) \\
 & & \quad \textbf{else } \text{azules}(r) \setminus \text{invertidas}(r,\, c) \\
 & & \textbf{fi}
\end{array}$$

Reversi: especificación

$$\text{puedeJugar}(r,\ c) \equiv \text{enRango}(r,\ c) \land c \notin \text{fichas}(r) \land_{\text{L}} \neg\, \emptyset?(\text{invertidas}(r,\ c))$$

$$\text{turno}(r) \equiv \textbf{if } \text{esPar?}(\#\text{fichas}(r)) \textbf{ then } \texttt{ROJO} \textbf{ else } \texttt{AZUL} \textbf{ fi}$$

Reversi: especificación

$$\text{puedeJugar}(r,\ c) \equiv \text{enRango}(r,\ c) \land c \notin \text{fichas}(r) \land_{\text{L}} \neg\, \emptyset?(\text{invertidas}(r,\ c))$$

$$\text{turno}(r) \equiv \textbf{if } \text{esPar?}(\#\text{fichas}(r)) \textbf{ then } \texttt{ROJO} \textbf{ else } \texttt{AZUL} \textbf{ fi}$$

Agregamos algunas operaciones extra en **otras operaciones**:

$$\text{enRango} : \text{reversi} \times \text{coord} \longrightarrow \text{bool}$$

$$\text{enRango}(r,\ c) \equiv 0 \leq c.x \land c.x < \text{dim}(r) \land 0 \leq c.y \land c.y < \text{dim}(r)$$

$$\text{fichas} : \text{reversi} \longrightarrow \text{conj(coord)}$$

$$\text{fichas}(r) \equiv \text{rojas}(r) \cup \text{azules}(r)$$

Reversi: especificación

(Más operaciones:)

$$\text{invertidas} : \text{reversi} \times \text{coord } c \longrightarrow \text{conj(coord)} \ \{\text{enRango}(r, c)\}$$

$$\begin{aligned}
\text{invertidas}(r, c) \equiv\ & \text{invertidasHacia}(r, \text{turno}(r), c, \emptyset, \langle 1, 0\rangle) && \Rightarrow \\
\cup\ & \text{invertidasHacia}(r, \text{turno}(r), c, \emptyset, \langle 1, 1\rangle) && \nearrow \\
\cup\ & \text{invertidasHacia}(r, \text{turno}(r), c, \emptyset, \langle 0, 1\rangle) && \Uparrow \\
\cup\ & \text{invertidasHacia}(r, \text{turno}(r), c, \emptyset, \langle -1, 1\rangle) && \nwarrow \\
\cup\ & \text{invertidasHacia}(r, \text{turno}(r), c, \emptyset, \langle -1, 0\rangle) && \Leftarrow \\
\cup\ & \text{invertidasHacia}(r, \text{turno}(r), c, \emptyset, \langle -1, -1\rangle) && \swarrow \\
\cup\ & \text{invertidasHacia}(r, \text{turno}(r), c, \emptyset, \langle 0, -1\rangle) && \Downarrow \\
\cup\ & \text{invertidasHacia}(r, \text{turno}(r), c, \emptyset, \langle 1, -1\rangle) && \searrow
\end{aligned}$$

Reversi: especificación

invertidasHacia : reversi r $\times$ jug $\times$ coord c
$\times$ conj(coord) $\times$ coord Δ
$\longrightarrow$ conj(coord) $\{$ enRango(r, c) $\}$

invertidasHacia(r, j, c, cs, Δ) $\equiv$
if enRango(r, c + Δ) $\wedge$ $c + \Delta \in$ fichas(r)
then **if** $c + \Delta \in$ fichasDe(r, j)
then cs
else invertidasHacia(r, j, $c + \Delta$, Ag($c + \Delta$, cs), Δ)
fi
else $\emptyset$
fi

fichasDe : reversi $\times$ jug $\longrightarrow$ conj(coord)
fichasDe(r, j) $\equiv$ **if** $j =$ ROJO **then** rojas(r) **else** azules(r) **fi**

$\bullet + \bullet$: coord $\times$ coord $\longrightarrow$ coord
c + d $\equiv \langle c.x + d.x, c.y + d.y \rangle$

Ejemplo: Reversi
(diseño de la interfaz)

Reversi: diseño de la interfaz

En general

- Un *módulo* es un componente de software que provee operaciones que implementan un *contrato*.
- La *interfaz* es el conjunto de dichas operaciones.
- Las operaciones se *implementan* con algoritmos y estructuras de datos.

En esta materia

- Cada *módulo* se corresponde con una clase en C++. (Típicamente).
- La *interfaz* está dada por los métodos públicos. (Pero ojo, en C++ no se explicita el contrato).
- La *implementación* está dada por los miembros privados e implementación de los métodos.

Reversi: diseño de la interfaz

El módulo Reversi debe permitir:

- Crear un nuevo juego.
- Contar con una operación para mostrar el estado del tablero.
- Jugar una ficha en una coordenada.

Reversi: diseño de la interfaz

El módulo REVERSI debe permitir:

- Crear un nuevo juego.
- Contar con una operación para mostrar el estado del tablero.
- Jugar una ficha en una coordenada.
  Nos piden que esta operación sea total.
  (Si no se puede jugar, la operación no tiene efecto).

Reversi: diseño de la interfaz

El módulo REVERSI debe permitir:

- Crear un nuevo juego.
- Contar con una operación para mostrar el estado del tablero.
- Jugar una ficha en una coordenada.
  Nos piden que esta operación sea total.
  (Si no se puede jugar, la operación no tiene efecto).
- Saber a quién le toca.
- Saber cuántas fichas hay de cada color.
  (Por ejemplo para saber quién ganó).

reversi: diseño de la interfaz

Módulo Reversi

Este módulo implementa las reglas de una partida de Reversi. Permite iniciar una partida nueva sobre una grilla cuadrada de lado par. El juego siempre tiene dos jugadores, Rojo y Azul, donde el Rojo es el primer jugador. El módulo permite a los jugadores jugar uno a uno y mantiene la información del estado del tablero para calcular las conversiones de fichas y saber si un movimiento es válido o no.

se explica con: TAD Reversi

Nuevo(**in** n : nat) $\longrightarrow$ *res* : Reversi

Pre $\equiv \{n > 0 \wedge (\exists k : \text{nat})(n =_{\text{obs}} 2 * k)\}$

Post $\equiv \{res =_{\text{obs}} \text{nuevo}(n)\}$

Crea un nuevo juego.

reversi: diseño de la interfaz

$\textsc{Turno}(\textbf{in}\ r : \textsc{Reversi}) \longrightarrow res : \textsc{Jug}$

Pre $\equiv \{\text{True}\}$

Post $\equiv \{res =_{\text{obs}} \text{turno}(r)\}$

Devuelve el jugador al que le toca jugar actualmente.

Reversi: diseño de la interfaz

$\textsc{Jugar}(\textbf{in/out}\ r : \textsc{Reversi}, \textbf{in}\ c : \textsc{Coord})$

$\text{Pre} \equiv \{r_0 =_{\text{obs}} r\}$

$\text{Post} \equiv$

$$\left\{ \begin{array}{lll} & (\text{puedeJugar?}(r, c) & \Rightarrow_{\text{L}} \quad r =_{\text{obs}} \text{jugar}(r_0, c)) \\ \wedge & (\neg\ \text{puedeJugar?}(r, c) & \Rightarrow \quad r =_{\text{obs}} r_0) \end{array} \right\}$$

Juega una ficha en la coordenada c.
Si no es posible, esta operación no tiene efecto.

Reversi: diseño de la interfaz

$\textsc{Puntaje}(\textbf{in}\ r : \textsc{Reversi}, \textbf{in}\ j : \textsc{Jug}) \longrightarrow res : \text{nat}$

$\text{Pre} \equiv \{\text{True}\}$

$\text{Post} \equiv \{res =_{\text{obs}} \#(\text{fichasDe}(r, j))\}$

Devuelve el puntaje del jugador indicado, es decir el número total de fichas de su color en el tablero.

$\textsc{Mostrar}(\textbf{in}\ r : \textsc{Reversi}, \textbf{out}\ t : \textsc{Tablero})$

Pre $\equiv \{\text{True}\}$

Post $\equiv$

$$\left\{ \begin{array}{lll} & \text{tam}(m) = \text{dim}(r) & \\ \wedge & (\forall i : \text{nat})(i < \text{tam}(t) \Rightarrow_{\text{L}} \text{tam}(t[i]) < \text{dim}(r)\ ) & \\ \wedge_{\text{L}} & (\forall i,j : \text{nat})(\text{enRango}(r, \langle i,j \rangle) \Rightarrow_{\text{L}} & \\ & \quad (\langle i,j \rangle \in \text{rojas}(r) & \Rightarrow \quad t[i][j] = \texttt{FichaRoja}) \\ & \wedge \quad (\langle i,j \rangle \in \text{azules}(r) & \Rightarrow \quad t[i][j] = \texttt{FichaAzul}) \\ & \wedge \quad (\langle i,j \rangle \notin \text{fichas}(r) & \Rightarrow \quad t[i][j] = \texttt{Nada})\ ) \end{array} \right\}$$

Arma una matriz que contiene una representación del tablero.

Donde:

- $\textsc{Tablero}$ es un renombre de $\textsc{Vector}(\textsc{Vector}(\textsc{Celda}))$
- $\textsc{Celda}$ es un tipo enumerado que puede tomar tres posibles valores: `FichaRoja`, `FichaAzul`, `Nada`.

Ejemplo: Reversi
(implementación en C++)

Reversi: implementación en C++

```
struct Coord { int x; int y; };
enum Jugador { Rojo, Azul };
enum Celda { FichaRoja, FichaAzul, Nada };
typedef std::vector<std::vector<Celda>> Tablero;

class Reversi {
public:
  Reversi(int n);
  Jugador turno() const;
  void jugar(Coord c);
  int puntaje(Jugador j) const;
  void mostrar(Tablero& t) const;
private:
  ...
};
```