Universidad Diego Portales

Laboratorio 05:
Conecta 4

Estructuras de Datos y Algoritmos
2025-1

Profesores:
Valentina Aravena
Cristián Llull
Marcos Fantoval

1. Introducción

Conecta 4, también conocido como *4 en Línea*, es un juego de dos jugadores con reglas simples. Para este laboratorio se considerarán las siguientes:

1. En un tablero de 6×7 se enfrentan el jugador A con el jugador B.
2. Los jugadores alternan sus turnos. El jugador A juega primero (inicia la partida).
3. En su turno, el jugador A inserta una ficha de color rojo (representado con una "X") en alguna columna con espacio (hay filas sin llenar). Dentro del tablero, esta ficha se desliza hasta llegar a la menor fila sin llenar, lo más al fondo posible.
4. Luego, el jugador B inserta una ficha de color amarillo (representado con una "O").
5. El juego termina cuando exista un ganador o empate.
    - Ganador: jugador que logró conectar 4 fichas de su color en forma horizontal, vertical o diagonal.
    - Empate: cuando ya no existen más movimientos disponibles (todas las columnas se encuentran llenas).

2. Implementación

Antes de empezar la implementación del tablero de *Conecta 4*, usted debe hacer implementación y/o tener a mano alguna implementación realizada en clases o ayudantía de BST (mapa basado en un árbol binario de búsqueda) y HashST (mapa basado en una tabla de hash). También puede usar las implementaciones ofrecidas por el texto guía. Los métodos de cada implementación quedan a criterio de usted; no obstante, como mínimo, ambas implementaciones requieren de un método para buscar y otro para insertar un elemento.

Su trabajo consiste en la integración de dichas estructuras de datos a la implementación que se especificará a continuación.

Cabe recalcar que en las implementaciones debe respetar los convenios de Java de usar el método equals para verificar igualdad de objetos y el método hashCode para calcular el código de hash; ambos presentes ya en la clase Object. Además, debe usar la interfaz Comparable o Comparator para comparar objetos.

Despuóes de haber conseguido las implementaciones de BST y HashST, debe implementar las siguientes clases con sus respectivos atributos y métodos:

2.1. Clase Player

La clase Player representa a un jugador de *Conecta 4* mediante los siguientes atributos:

- playerName (String): nombre del jugador, usado como identificador único.
- wins (int): número de partidas ganadas por el jugador.
- draws (int): número de partidas empatadas por el jugador.
- losses (int): número de partidas perdidas por el jugador.

Los métodos a implementar son los siguientes:

- addWin(): incrementa en 1 el número de partidas ganadas por el jugador.
- addDraw(): incrementa en 1 el número de partidas empatadas por el jugador.
- addLoss(): incrementa en 1 el número de partidas perdidas por el jugador.
- winRate(): retorna el porcentaje de victorias del jugador entre todas las partidas jugadas como un valor decimal entre 0 y 1. Si el jugador no ha participado en ninguna partida, debe retornar 0.

2.2. Clase Scoreboard

La clase `Scoreboard` gestiona el registro de jugadores y los resultados de las partidas mediante los siguientes atributos:

- winTree (BST<int, String>): árbol binario de búsqueda que ordena a los jugadores por su número de victorias. La llave es el número de victorias, y el valor es el nombre del jugador (playerName).
- players (HashST<String, Player>): Tabla de hash que asocia el nombre de un jugador (`playerName`) con su instancia de la clase Player. Esta tabla de hash es la que lleva registro de los jugadores.
- playedGames (int): Número total de partidas jugadas registradas en el sistema. Es un contador que inicializa en cero.

Los métodos a implementar son:

- addGameResult(String winnerPlayerName, String looserPlayerName, boolean draw): registra el resultado de una partida, actualizando las estadísticas de los jugadores involucrados y el árbol binario de búsqueda (`winTree`).
- registerPlayer(String playerName): registra un nuevo jugador en el sistema, agregándolo a la tabla de hash players. Si el jugador ya existe, no lo registra nuevamente.
- checkPlayer(String playerName): verifica si un jugador con el nombre dado (`playerName`) está registrado en el sistema (lo busca en la tabla de hash players).
- winRange(int lo, int hi): retorna un arreglo (Player[]) con los jugadores que tienen un número de victorias mayor o igual que lo y menor o igual que hi.
- winSuccessor(int wins): Retorna un arreglo de jugadores (Player[]) con el número de victorias mayor más cercano al valor especificado (wins). Se retorna un arreglo porque podrían ser múltiples jugadores con la misma cantidad de victorias.[1] Cuando no exista sucesor, el método debe retornar un arreglo vacío.

2.3. Clase ConnectFour

La clase ConnectFour representa el tablero y la lógica del juego mediante los siguientes atributos:

- grid (char[7][6]): matriz 7×6 que representa el estado del tablero. Las celdas vacías están representadas por el carácter espacio (`' '`).
- currentSymbol (char): currentSymbol es un caracter que puede tener valores X o O. Los valores van alternando tras cada ejecución de `makeMove`.

Los métodos a implementar son:

- ConnectFour(): constructor que inicializa grid con el tablero llenado con caracteres de espacio (`' '`) y `currentSymbol` con el valor X.
- makeMove(int z): intenta realizar un movimiento en la columna indicada con el simbolo actual (currentSymbol). Retorna false si la columna se encuentra llena (no se puede realizar el movimiento) o si la coordenada es inválida, y true si el movimiento pudo realizarse (se puede apilar el símbolo actual dentro de la columna). En caso de poder realizar el movimiento solicitado, se actualiza el contenido de grid y el valor en currentSymbol. De lo contrario, grid y currentSymbol quedan sin modificaciones.
- isGameOver(): verifica si el juego ha terminado, ya sea porque un jugador ganó o porque se produjo un empate. Este método debe determinar qué jugador ganó, o si el juego quedó en empate. Usted debe definir los parámetros del método isGameOver, como asimismo el tipo del valor devuelto.

[1] Por ejemplo, si el árbol contiene las llaves 4, 6, 8, 9, 12, el sucesor de 9 es 12 puesto que es el número mayor más cercano a 9.

2.4. Clase Game

La clase Game representa una partida de *Conecta 4* entre dos jugadores mediante los siguientes atributos:

- status (String): estado de la partida, que puede ser: `IN_PROGRESS`, `VICTORY` o `DRAW`.
- winnerPlayerName (String): nombre del jugador ganador, o un String vacío si el resultado fue un empate.
- playerNameA (String): nombre del jugador A.
- playerNameB (String): nombre del jugador B.
- ConnectFour (ConnectFour): instancia de la clase ConnectFour que representa el tablero y la lógica del juego.

Los métodos a implementar son:

- Game(String playerNameA, String playerNameB): constructor que inicializa una nueva partida con los jugadores especificados. El estado (status) del juego una vez inicializado es `IN_PROGRESS`.
- play(): ejecuta una partida de Conecta 4 en un ciclo **hasta que exista un ganador o empate**.
  En cada iteración se usa el método `makeMove` de la clase ConnectFour para realizar un movimiento y el método `isGameOver` para verificar el estado del juego después de realizar el movimiento. Este método actualiza el estado del juego (status) a `VICTORY` o `DRAW`, según corresponda, registra el resultado (`winnerPlayerName`) y retorna el nombre del ganador o un String vacío en caso de empate.
  Para interactuar con la instancia de ConnectFour, dentro de este método los jugadores realizan ingreso de la columna en la que desean insertar sus símbolos mediante `stdin` (System.in en Java). Se recomienda que los estudiantes impriman cada estado de Conecta 4: qué jugador debe ingresar su símbolo y el estado actual del tablero para interactuar de la mejor forma posible (hay libertad para ser creativos en este punto).

3. Análisis

A continuación se detallan distintos análisis esperados del laboratorio.

3.1. Análisis asintótico

Realizar un análisis asintótico del tiempo de ejecución para los siguientes métodos de la clase Scoreboard:

a. addGameResult

b. registerPlayer

c. checkPlayer

d. winRange

e. winSuccessor

3.2. Ventajas y desventajas

Describir dos ventajas y dos desventajas de la implementación propuesta del sistema. Por ejemplo, podría considerar la claridad del diseño, o las limitaciones de escalabilidad para un gran número de jugadores.

3.3. Desafíos

Describir los mayores desafíos que enfrentó durante la experiencia del laboratorio. Documentar los criterios que utilizó para resolver dichos desafíos. Con "desafío" se refiere a la dificultad (en otras palabras, lo que le resultó más difícil). Por ejemplo:

a. Diseñar la lógica del método isGameOver.

b. Integración de la clase BST y HashST a la implementación.

3.4. Extensión

Proponer cómo se podría extender el sistema para incluir soporte a torneos de *Conecta 4* entre los jugadores. Esto incluiría múltiples rondas, un sistema de clasificación y reglas para determinar al campeón. No se requiere una implementación, pero se solicita que describa los requerimientos y las modificaciones necesarias al diseño del sistema.

3.5. Revisión de estructuras de datos

Proponga una estructura de datos disponible en Java (cualquiera de las estudiadas en clases) como `LinkedList`, `HashSet`, etc. Con la que pueda reemplazar la implementación realizada de BST y otra para HashST. Comente sobre las ventajas y desventajas de usar las estructuras de la librería estándar de Java.

3.6. Pruebas

Para probar su implementación, realícelo de forma interactiva. Juegue unas cuantas partidas con alguno de sus compañeros y documente sus observaciones.

4. Informe

El informe debe tener las siguientes secciones:

1. **Introducción:** En esta sección, debe explicar a grandes rasgos de qué trata la experiencia.
2. **Implementación:** En esta sección, debe explicar de manera resumida cómo realizó la implementación. En caso de haber realizado modificaciones a las clases especificadas, se debe relatar en esta sección dichas modificaciones y las razones de cada decisión.
3. **Análisis:** En esta sección debe responder a las preguntas de la sección de análisis.
4. **Conclusión:** En esta sección, relate brevemente lo aprendido en la experiencia. También reflexione brevemente sobre uno o dos casos de la vida real (proyectos aplicados) en que sea necesario implementar una estructuras de datos como se realizó en el laboratorio, en lugar de utilizar las que existen en la librería estándar del lenguaje utilizado.