Universidad Diego Portales
Escuela de Informática &
Telecomunicaciones

Estructuras de Datos & Análisis de Algoritmos

Laboratorio n°5

Autores:
Allen Mora
Gabriel Varas

Profesor:
Marcos Fantóval

28/06/2025

Índice

1. Introducción

En este laboratorio se desarrollará e implementará un algoritmo capaz de gestionar y jugar partidas del juego ¨Conecta 4¨. El objetivo del laboratorio fue el uso de BST(Binary Search Tree) y tablas hash(HashST),

2. Metodología

Para la creación del algoritmo se emplearon 5 clases en java siendo estas:

- ConnectFour: En esta clase se representa el tablero y toda la lógica del juego, es en esta clase que se gestiona el estado del tablero, los turnos, las jugadas realizadas y la verificación de los ganadores.
  Como atributos esta clase contiene ¨grid¨ la cual es una matriz de 6x7 que representa el tablero.
  Su siguiente atributo ¨currentSymbol¨ indica de quien es el turno, este alterna entre ¨X¨ y ¨O¨ dependiendo del jugador del cual sea el turno.
  Los método de esta clase son:
  - public ConnectFour(): Este método es el constructor de la clase, se encarga de inicializar el tablero con espacios vacíos ¨ ¨, luego el primer turno siempre es del jugador ¨X¨.

```
public class ConnectFour {
    char[][] grid;
    char currentSymbol;

    public ConnectFour() {
        this.grid = new char[6][7];
        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 7; j++) {
                grid[i][j] = ' ';
            }
        }
        this.currentSymbol = 'X';
    }
```

Figura 1: Clase ConnectFour

  - public boolean makeMove(int col): Este método intenta colocar una ficha en la columna ¨col¨, recorre desde abajo hacia arriba para colocar la ficha en la primera posición libre, si la columna esta llena retorna ¨false¨ por ultimo luego de colocar la ficha, alterna el símbolo para el siguiente turno.

```java
public boolean makeMove(int col) {
    if (col < 0 || col >= 7) {
        return false;
    }
    for (int i = 5; i >= 0; i--) {
        if (grid[i][col] == ' ') {
            grid[i][col] = currentSymbol;
            currentSymbol = (currentSymbol == 'X') ? 'O' : 'X';
            return true;
        }
    }
    return false;
}
```

Figura 2: Método makeMove

- public boolean isGridFull(): Este método devuelve ¨true¨ si la fila superior de todas las columnas esta ocupada (esto significa que el tablero esta lleno), si hay por lo menos una celda vacía retorna ¨false¨ y aun se puede jugar.

```java
public boolean isGridFull() {
    for (int j = 0; j < 7; j++) {
        if (grid[0][j] == ' ') {
            return false;
        }
    }
    return true;
}
```

Figura 3: Método isGridFull

- public boolean isGameOver(): Este método retorna ¨true¨, si hay un ganador (checkWinner()), o si el tablero está lleno (isGridFull()).

```java
public boolean isGameOver() {
    return checkWinner() || isGridFull();
}
```

Figura 4: Método isGameOver

- public boolean checkWinner(): Este método recorre todas las celdas del tablero, si la celda no está vacía comprueba si hay 4 en línea, de la siguiente manera:
  - Horizontal (1,0)
  - Vertical (0,1)
  - Diagonal inferior derecha (1,1)
  - Diagonal inferior izquierda (1, -1)

  Para eso utiliza el método de apoyo llamado checkDirection()

```
public boolean checkWinner() {
    for (int i = 0; i < 6; i++) {
        for (int j = 0; j < 7; j++) {
            if (grid[i][j] != ' ' && (
                checkDirection(i, j, 1, 0) || // Horizontal
                checkDirection(i, j, 0, 1) || // Vertical
                checkDirection(i, j, 1, 1) || // Diagonal
                checkDirection(i, j, 1, -1))) { // Diagonal
                return true;
            }
        }
    }
    return false;
}
```

Figura 5: Método checkWinner

- prive boolean checkDirection(int row, int col, int deltaRow, int deltaCol): Este método verifica si existen 4 fichas iguales a partir de una celda (row,col) y en una dirección(deltaRow,deltaCol), en caso de encontrar una secuencia de 4 fichas iguales, el método retorna ¨true¨.

```
private boolean checkDirection(int row, int col, int deltaRow, int deltaCol) {
    char symbol = grid[row][col];
    int count = 0;
    for (int i = 0; i < 4; i++) {
        int newRow = row + i * deltaRow;
        int newCol = col + i * deltaCol;
        if (newRow < 0 || newRow >= 6 || newCol < 0 || newCol >= 7 || grid[newRow][newCol] != symbol) {
            return false;
        }
        count++;
    }
    return count == 4;
}
```

Figura 6: Método checkDirection

- public void printGrid(): Este método imprime el tablero en consola para poder visualizarlo.

```
    public void printGrid() {
        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 7; j++) {
                System.out.print("|" + grid[i][j]);
            }
            System.out.println("|");
        }
        System.out.println("---------------");
        System.out.println(" 0 1 2 3 4 5 6 ");
    }
}
```

Figura 7: Método printGrid

- Game: Esta clase cuenta con 5 atributos:
  - String status: Indica el estado del juego, ¨IN_PROGRESS¨ si aun se esta jugando,¨VICTORY¨ si el jugador gano o ¨DRAW¨ si el juego terminó en empate.
  - String winnerPlayerName: Almacena el nombre del jugador que ganó en caso de existir.
  - String PlayerNameA: Almacena el nombre del primer jugador.
  - String PlayerNameB: Almacena el nombre del segundo jugador.
  - ConnectFour connectFour: Es el tablero de juego.

  Los métodos que se encuentran en esta clase son los siguientes:

  - public Game(String playerNameA, String playerNameB): Este método es el constructor de la clase, recibe los nombres de ambos jugadores, crea un nuevo tablero (ConnectFour) y establece el estado como ¨IN_PROGRESS¨.

```
import java.util.Scanner;

public class Game {
    String status; // "IN_PROGRESS", "VICTORY", "DRAW"
    String winnerPlayerName;
    String playerNameA;
    String playerNameB;
    ConnectFour connectFour;

    public Game(String playerNameA, String playerNameB) {
        this.status = "IN_PROGRESS";
        this.winnerPlayerName = "";
        this.playerNameA = playerNameA;
        this.playerNameB = playerNameB;
        this.connectFour = new ConnectFour();
    }
```

Figura 8: Clase Game

- public String play(): Este método controla todo el ciclo de una partida, hace que el juego continúe mientras el estado no sea igual a ¨VICTORY¨ o ¨DRAW¨, también controla el turno de cada jugador, mostrando el tablero en la consola y pidiendo al jugador que ingrese la columna en la cual quiere colar su ficha en este turno, en caso de que la jugada sea valida se verifica el termino del juego, si hay 4 en linea se declara la victoria y se guarda el nombre del jugador que gano, si se llena el tablero sin un ganador se declara el empate, en caso de que no se termine el juego se cambia al otro jugador, este método además se asegura que las jugadas sean validas, en caso de no serlo muestra en consola que es un movimiento invalido y se solicita al jugador que pruebe con otro, luego imprime el tablero final y felicita al ganador o declara el empate, finalmente retorna el nombre del jugador que sale victorioso.

```
    public String play() {
        Scanner scanner = new Scanner(System.in);
        String currentPlayer = playerNameA;
        while (!status.equals("VICTORY") && !status.equals("DRAW")) {
            connectFour.printGrid();
            System.out.println("Es turno de " + currentPlayer + ". Ingresa la columna (0-6):");

            int col = scanner.nextInt();

            if (connectFour.makeMove(col)) {
                if (connectFour.isGameOver()) {
                    if (connectFour.checkWinner()) {
                        status = "VICTORY";
                        winnerPlayerName = currentPlayer;
                    } else {
                        status = "DRAW";
                    }
                }
                currentPlayer = (currentPlayer.equals(playerNameA)) ? playerNameB : playerNameA;
            } else {
                System.out.println("Movimiento inválido. Intenta de nuevo.");
            }
        }

        connectFour.printGrid();
        if (status.equals("VICTORY")) {
            System.out.println("Felicidades " + winnerPlayerName + ", ganaste");
        } else {
            System.out.println("Empate. No hay más movimientos posibles.");
        }
        return winnerPlayerName;
    }
}
```

Figura 9: Método play

- Player: En esta clase se guarda la información y estadísticas de un jugador de conecta 4, esta clase cuenta con 4 atributos siendo estos:
  - String playerName: Almacena el nombre del jugador.
  - int wins: Almacena el numero de partidas ganadas.
  - int draws: Almacena el numero de partidas empatadas.
  - int losses: Almacena el numero de partidas perdidas.

Esta clase también cuenta con sus métodos propios los cuales son:

- public Player(String playerName): Este método es el constructor de la clase inicializa al jugador con su nombre y con las estadisticas en 0.

```
public class Player {
    String playerName;
    int wins;
    int draws;
    int losses;

    public Player(String playerName) {
        this.playerName = playerName;
        this.wins = 0;
        this.draws = 0;
        this.losses = 0;
    }
```

Figura 10: Clase Player

- public void addWin(): Este método suma una victoria.

```
public void addWin() {
    this.wins++;
}
```

Figura 11: Método addWin

- public void addDraw(): Este método suma un empate.

```
public void addDraw() {
    this.draws++;
}
```

Figura 12: Método addDraw

- public void addLoss(): Este método suma una derrota.

```
public void addLoss() {
    this.losses++;
}
```

Figura 13: Método addLoss

- public int winRate(): Este método devuelve el porcentaje de partidas ganas (win rate) como un numero entero, en caso de que el jugador aun no haya jugado retorna 0.

```
public int winRate() {
    int totalGames = wins + draws + losses;
    return totalGames == 0 ? 0 : (wins * 100) / totalGames;
}
```

Figura 14: Método addRate

- Scoreboard: Esta clase cuenta con 3 atributos:
  - TreeMap¡Integer, String¿winTree: Ordena los jugadores por la cantidad de victorias, la clave es el numero de victorias y el valor es el nombre del jugador.
  - HashMap¡String, Player¿players: guarda a cada jugador registrado con su nombre como clave.
  - int playedGames: contador total de partidas jugadas.

  Esta clase también cuenta con 7 métodos siendo estos:
  - public Scoreboard(): Este método es el constructor de la clase, inicializa las estructuras vacías.

```
import java.util.HashMap;
import java.util.TreeMap;

public class Scoreboard {
    TreeMap<Integer, String> winTree;
    HashMap<String, Player> players;
    int playedGames;

    public Scoreboard() {
        this.winTree = new TreeMap<>();
        this.players = new HashMap<>();
        this.playedGames = 0;
    }
```

Figura 15: Clase Scoreboard

- public void addGameResult(String winnerPlayerName, String loserPlayerName, boolean draw): Este método funciona de 2 maneras, si la partida jugada no es un empate, el método suma 1 victoria al ganador y una derrota al perdedor, luego actualiza el ¨winTree¨ para que el TreeMap tenga al ganador con su nuevo numero de vitorias. Por otro lado si la partida termina en empate se suma un empate a ambos jugadores y se incrementa el contador de partidas jugadas.

```
public void addGameResult(String winnerPlayerName, String loserPlayerName, boolean draw) {
    Player winner = players.get(winnerPlayerName);
    Player loser = players.get(loserPlayerName);

    if (!draw) {
        winner.addWin();
        loser.addLoss();

        winTree.put(winner.wins, winnerPlayerName);
    } else {
        winner.addDraw();
        loser.addDraw();
    }
    playedGames++;
}
```

Figura 16: Método addGameResult

- public void registerPlayer(String playerName): Este método registra un nuevo jugador en caso de no existir uno aun.

```
public void registerPlayer(String playerName) {
    if (!players.containsKey(playerName)) {
        players.put(playerName, new Player(playerName));
    }
}
```

Figura 17: Método registerPlayer

- public boolean checkPlayer(String playerName): Este método entrega el valor ¨true¨ en caso de que el jugador ya esté registrado.

```
public boolean checkPlayer(String playerName) {
    return players.containsKey(playerName);
}
```

Figura 18: Método checkPlayer

- public Player[] winRange(int lo, int hi): Este método retorna un arreglo con los jugadores que tengan entre ¨lo¨ y ¨hi¨ victorias, de mayor a menor.

```
public Player[] winRange(int lo, int hi) {
    Player[] result = new Player[hi - lo + 1];
    int index = 0;

    for (int i = hi; i >= lo; i--) {
        if (winTree.containsKey(i)) {
            String playerName = winTree.get(i);
            result[index++] = players.get(playerName);
        }
    }

    return result;
}
```

Figura 19: Método winRange

- public Player[] winSuccessor(int wins): Este método recorre el ¨winTree¨ desde ¨wins + 1¨ hasta ¨players.size()¨, además redimensiona el arreglo para eliminar posiciones vacías.

```
public Player[] winSuccessor(int wins) {
    Player[] result = new Player[players.size()];
    int index = 0;

    for (int i = wins + 1; i <= players.size(); i++) {
        if (winTree.containsKey(i)) {
            String playerName = winTree.get(i);
            result[index++] = players.get(playerName);
        }
    }

    // Resize the array to the actual number of players found
    Player[] trimmedResult = new Player[index];
    System.arraycopy(result, 0, trimmedResult, 0, index);
    return trimmedResult;
}
```

Figura 20: Método winSuccessor

- public void printScoreboard(): Este método muestra una tabla con:
  - Nombre.
  - Partidas ganadas.
  - Partidas empatadas.
  - Partidas perdidas.
  - Porcentaje de victorias

  Además muestra el total de partidas jugadas.

```
public void printScoreboard() {
    System.out.println("Scoreboard:");
    System.out.println("Player\tWins\tDraws\tLosses\tWin Rate");
    for (Player player : players.values()) {
        System.out.printf("%s\t%d\t%d\t%d\t%d%%\n",
            player.playerName,
            player.wins,
            player.draws,
            player.losses,
            player.winRate());
    }
    System.out.println("Total games played: " + playedGames);
}
```

Figura 21: Método printScoreboard

- Main: Es la clase que inicia el algoritmo, controla las partidas y muestra los resultados usando ¨Scoreboard¨. Primero se solicita ingresar los nombres de los dos jugadores, los registra en el ¨Scoreboard¨, luego inicia una partida creando una instancia de ¨Game¨. Finalmente, al terminar la partida verifica si existe un ganador; en caso de no existir, se considera empate y la partida se registra como tal. Si hay un ganador, se actualiza la victoria y la derrota de los jugadores, luego imprime una tabla actualizada con las estadísticas de los jugadores y pregunta al usuario si desea volver a jugar.

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Scoreboard scoreboard = new Scoreboard();

        System.out.println("Bienvenido al juego Conecta 4");
        System.out.print("Ingresa el nombre del jugador A: ");
        String playerNameA = scanner.nextLine();
        System.out.print("Ingresa el nombre del jugador B: ");
        String playerNameB = scanner.nextLine();

        scoreboard.registerPlayer(playerNameA);
        scoreboard.registerPlayer(playerNameB);

        boolean playing = true;
        while (playing) {
            Game game = new Game(playerNameA, playerNameB);
            String winner = game.play();

            if (winner.isEmpty()) {
                System.out.println("No hubo ganador esta vez.");
                scoreboard.addGameResult(playerNameA, playerNameB, true);
            } else if (winner.equals(playerNameA)) {
                scoreboard.addGameResult(playerNameA, playerNameB, false);
            } else {
                scoreboard.addGameResult(playerNameB, playerNameA, false);
            }

            scoreboard.printScoreboard();

            System.out.print("¿Quieres jugar otra vez? (s/n): ");
            String response = scanner.nextLine();
            playing = response.equalsIgnoreCase("s");
        }

        System.out.println("Gracias por jugar Conecta 4.");
        scanner.close();
    }
}
```

Figura 22: Clase Main

3. Análisis

3.1. Análisis asintótico

Análisis asintótico para los métodos de la clase Scoreboard donde ¨w¨ es el numero máximo de victorias registradas en el winTree y ¨n¨ es el numero total de jugadores.

- addGameResult: Este método tiene complejidad O(log w).
- registerPlayer: Este método tiene complejidad O(1).
- checkPlayer: Este método tiene complejidad O(1).
- winRange: Este método tiene complejidad de O((hi - lo + 1) * log w).
- winSuccessor: Este método tiene complejidad O(n * log w).

3.2. Ventajas y desventajas

Ventajas :

- La implementación divide el sistema en clases específicas (Player, Game, ConnectFour, Scoreboard), lo que facilita la separación de responsabilidades. Esto permite que cada componente se enfoque en una tarea específica.
- La estructura permite reutilizar componentes como Player y Scoreboard en otros juegos o sistemas similares.

Desventajas:

- La estructura está diseñada para un juego sencillo con dos jugadores y un tablero fijo. Si se quisiera expandir el sistema para manejar múltiples partidas simultáneas, torneos o estadísticas más avanzadas, sería necesario realizar cambios significativos en el código.
- No hay una función que permita almacenar datos, como los resultados o jugadores.

3.3. Desafíos

La parte más desafiante durante la creación del código fue el algoritmo con el cual se determina un ganador (checkWinner) puesto que se necesita verificar cada posible combinación de fichas para detectar si existen 4 fichas en raya.

3.4. Extensión

Para poder extender el sistema a uno de torneos, se podría crear una nueva clase ‘Tournament‘, que controle el flujo del torneo, es decir, el tipo de torneo (mediante llaves de eliminación o tipo liga). En caso de ser formato liga, el jugador con mayor

cantidad de puntos sería el campeón(3 puntos por victoria, 1 punto por empate y 0 puntos por derrota). Si fuera mediante llaves, el último jugador en ganar sería el campeón. Las modificaciones necesarias serian:

- En la clase Main: Agregar la opción de crear y efectuar torneos.
- En la clase Player: Agregar atributos tales como los puntos obtenidos en el torneo o la cantidad de torneos ganados.
- en la clase Scoreboard: Poder dividir entre torneos y partidas casuales.

3.5. Revisión de estructuras de datos

El BST(Binary Search Tree) puede ser reemplazado por un TreeMap o por un TreeSet, las ventajas de reemplazarlo serian:

- Ambas estructuras implementan arboles balanceados por lo que la busqueda, inserción y eliminación de elementos tendrían complejidad O(log n),
- Ordenan los elementos mediante un comparador.
- Evitan errores en las implementación manuales.
- probeen métodos avanzados como ¨subMap¨, ¨headMap¨ y ¨tailMap¨ además de una navegación por rangos.

Sin embargo, esto también traería sus desventajas, tales como:

- Menor control sobre la estructura del árbol(no se puede definir el tipo de balanceo deseado o como se recorrerá el árbol,¨IN_ORDER¨ , ¨PRE_ORDER¨ o ¨POST_ORDER¨).

Como reemplazo de HashST se puede utilizar un ¨HashMap¨, algunas ventajas de utilizar HashMap sobre HashSt son:

- Tiene un mejor rendimiento incluso con grandes cantidades de datos.
- El HashMap se re expande de manera automática cuando se excede su capacidad, por lo que no es necesario de gestionarlo manualmente.
- Tiene un código más comprensible y manteniente.

Por otro lado las desventajas de utilizar HashMap serían:

- No se puede definir exactamente como se calcula el indice o como se manejan las colisiones de datos.
- El HashMap no tiene un orden garantizado por lo que si se necesita un orden de inserción o por clave se debe utilizar ¨LinkedHashMap¨ o ¨TreeMap¨.

3.6. Pruebas

Se presentarán algunas imágenes de partidas realizadas con el algoritmo desarrollado.

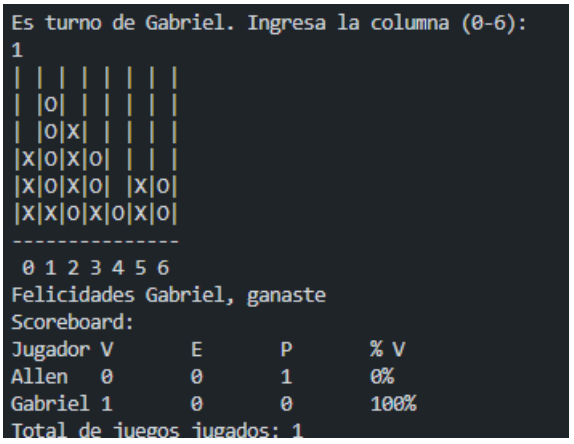

Figura 23: Partida n°1

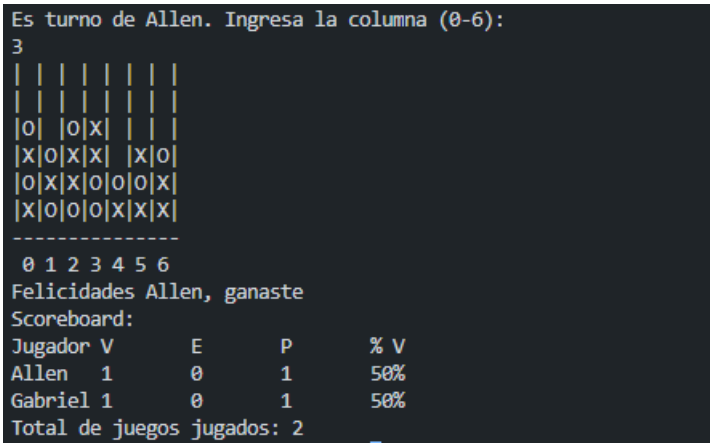

Figura 24: Partida n°2

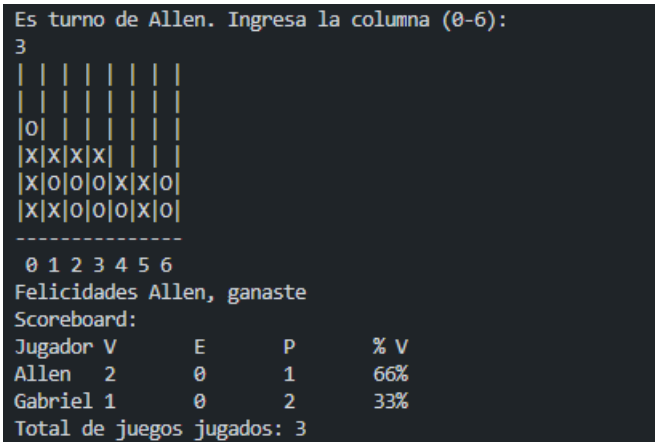

Figura 25: Partida n°3

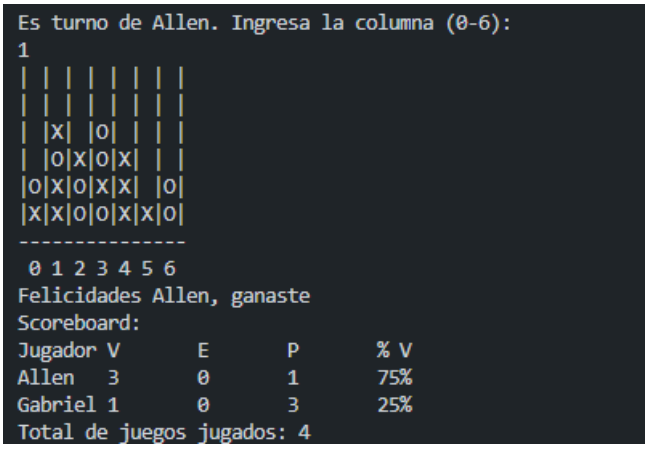

Figura 26: Partida n°4

4. Conclusiones

El desarrollo de este laboratorio, permitió más que crear un sistema funcional del juego conecta 4, permitió profundizar en el diseño modular utilizando estructuras de datos avanzadas tales como los BST y tablas hash. Por otro lado este laboratorio ayudó a comprender la importancia de definir de manera correcta las clases y el rol que cumplirán estas dentro del sistema, con el propósito de favorecer la mantenibilidad y el posible crecimiento a futuro del sistema.

https://github.com/gabovrs/lab5eda