Universidad Diego Portales
escuela de informática & telecomunicaciones

ESTRUCTURAS DE DATOS & ANÁLISIS DE ALGORITMOS

Laboratorio 5: Conecta 4.

Autores:
Valentina Martínez
José Pablo Peña

Profesor:
Marcos Fantoval

24 de junio de 2025

Índice

1. Introducción

Este laboratorio número cinco, tiene como objetivo implementar una versión programada del juego "Conecta 4", para así aprender conceptos de programación orientada a objetos y estructuras de datos. Este laboratorio por un lado tiene la lógica del juego, la cual es que dos jugadores tienen turnos para insertar fichas en un tablero de 6x7, buscando formar una línea de cuatro símbolos iguales, y por el otro lado tiene que se combina con el uso de estructuras clave para el seguimiento y análisis de partidas.

El sistema nos pide utilizar un árbol binario de búsqueda (BST) para ordenar jugadores según su número de victorias, y una tabla de hash (HashST) para registrar y acceder a los datos de cada jugador. Esto permite gestionar el historial de partidas, calcular victorias y realizar consultas como rangos de ganadores o búsqueda de sucesores. Además de crear todo el juego, el laboratorio tiene como objetivo el diseño de un sistema que combina lógica de juego y estructura de datos, para así crear habilidades fundamentales para problemas similares en el desarrollo de estos programas.

2. Implementación

Este laboratorio se hizo con el lenguaje de programación Java, el cual al ser orientado a objeto, nos permite crear distintas clases. El código completo se encuentra en: https://github.com/valemiauz/Lab-5

Antes de construir las clases principales del sistema, se implementaron dos estructuras de datos fundamentales: en primer lugar un árbol binario de búsqueda (BST), usado para ordenar jugadores por número de victorias, y en segundo lugar una tabla de hash (HashST), utilizada para registrar y acceder rápidamente a la información de cada jugador, ya que ambas están incluidas en las clases solicitadas. Ambas estructuras tienen métodos para insertar y para búsqueda de elementos, también fueron desarrolladas respetando los convenios de Java en cuanto al uso de equals y hashCode, y por último la implementación de Comparable.

Respecto a la construcción de las clases, se hicieron modificaciones menores necesarias para el funcionamiento del sistema y se agregaron los métodos:

Métodos get en Player para acceso a datos privados.

Método printBoard() en ConnectFour para mostrar el juego.

Método getCurrentSymbol() en ConnectFour para determinar a cual jugador le toca.

Método isGameOver() para retornar String con el estado del juego.

Todas estas modificaciones son necesarias para que el código funcione correctamente.

Ya con las estructuras BST y HashST listas, se construyeron las siguientes clases:

2.1. Clase Player

La primera clase es llamada Player que representa a cada jugador con su historial de resultados.

Tiene como atributo lo siguiente:

- playerName (String): Nombre del jugador (id).
- wins (int): Número de partidas ganadas.
- draws (int): Número de partidas empatadas.
- losses (int): Número de partidas perdidas.

Tiene los siguientes métodos:

- 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. Si el jugador no ha participado en ninguna partida, debe retornar 0.

Resultando así la clase:

```java
class Player {
    private String playerName;
    private int wins;
    private int draws;
    private int losses;

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

    public void addWin() {
        wins++;
    }

    public void addDraw() {
        draws++;
    }

    public void addLoss() {
        losses++;
    }

    public double winRate() {
        int totalGames = wins + draws + losses;
        if (totalGames == 0) return 0;
        return (double) wins / totalGames;
    }

    public String getPlayerName() {
        return playerName;
    }

    public int getWins() {
        return wins;
    }

    public int getDraws() {
        return draws;
    }

    public int getLosses() {
        return losses;
    }
```

2.2. Clase Scoreboard

Esta clase representa Scoreboard que integra el BST y la HashST para registrar jugadores, partidas y permitir consultas.

Tiene como atributo lo siguiente:

- `winTree (BST<int, String>)`: Árbol binario de búsqueda que organiza a los jugadores según la cantidad de partidas ganadas. La clave es el número de victorias y el valor asociado es el nombre del jugador.
- `players (HashST<String, Player>)`: Tabla hash que relaciona cada nombre de jugador con su objeto correspondiente de la clase Player. Mantiene un registro de todos los jugadores registrados en el sistema.
- `playedGames (int)`: Contador que representa la cantidad total de partidas que han sido registradas en el sistema. Comienza con un valor inicial de cero.

Los métodos a implementar son:

- addGameResult(String winnerPlayerName, String looserPlayerName, boolean draw): Este método guarda el resultado de una partida (victoria, derrota o empate), actualizando las estadísticas de los jugadores involucrados y modificando el árbol winTree.
- registerPlayer(String playerName): Registra un nuevo jugador en el sistema, lo añade a la tabla hash players. Si el jugador ya se encuentra registrado, no se realiza ninguna acción.
- checkPlayer(String playerName): Revisa si un jugador con el nombre dado está registrado en el sistema, buscando su existencia en la tabla hash players.
- winRange(int lo, int hi): Devuelve un arreglo de objetos Player que corresponde a los jugadores cuyo número de victorias está dentro del rango definido entre los valores lo y hi.
- winSuccessor(int wins): Retorna un arreglo con los jugadores que tienen la cantidad de victorias mayor a la dada. Si hay varios jugadores con ese mismo número de victorias, todos se incluyen en el arreglo. Si no existe un número mayor de victorias, se devuelve un arreglo vacío.

Resultando así la clase:

```
class Scoreboard {
    private BST<Integer, String> winTree;
    private HashST<String, Player> players;
    private int playedGames;

    public Scoreboard() {
        winTree = new BST<>();
        players = new HashST<>();
        playedGames = 0;
    }

    public void addGameResult(String winnerPlayerName, String
        looserPlayerName, boolean draw) {
        if (draw) {
            Player winner = players.get(winnerPlayerName);
            Player looser = players.get(looserPlayerName);
            winner.addDraw();
            looser.addDraw();
        } else {
            Player winner = players.get(winnerPlayerName);
            Player looser = players.get(looserPlayerName);

            winner.addWin();
            looser.addLoss();

            winTree.put(winner.getWins(), winner.getPlayerName());
        }
        playedGames++;
    }

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

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

    public Player[] winRange(int lo, int hi) {
        List<Integer> keys = winTree.keysInRange(hi, lo);
        List<Player> result = new ArrayList<>();

        for (Integer wins : keys) {
            String playerName = winTree.get(wins);
            Player player = players.get(playerName);
            result.add(player);
        }

        return result.toArray(new Player[0]);
```

```
51      }
52
53      public Player[] winSuccessor(int wins) {
54          Integer successor = winTree.successor(wins);
55          if (successor == null) {
56              return new Player[0];
57          }
58
59          String playerName = winTree.get(successor);
60          Player player = players.get(playerName);
61          return new Player[]{player};
62      }
63  }
```

2.3. Clase ConnectFour

Esta clase es llamada ConnectFour que gestiona el tablero, los turnos y las jugadas válidas.

Tiene como atributo lo siguiente:

- grid (char[7][6]): Es una matriz de 7 filas por 6 columnas que representa el tablero de juego. Cada celda de la matriz puede estar vacía o contener el símbolo de un jugador.
- currentSymbol (char): Este carácter puede ser X o O, y se alterna automáticamente cada vez que se ejecuta un movimiento a través del método makeMove.

Los métodos a implementar son:

- ConnectFour(): Constructor de la clase. Su función es inicializar el tablero (grid) llenando todas sus posiciones con el carácter espacio (' '), y establecer el símbolo actual (currentSymbol) como "X" al comienzo del juego.
- makeMove(int z): Intenta colocar el símbolo del jugador actual en la columna especificada. Si la columna no es válida, el método retorna false. Si el movimiento es posible, se actualiza la matriz grid con el símbolo correspondiente, se alterna el valor de currentSymbol, y el método retorna true.
- isGameOver(): Revisa si el juego ha terminado, ya sea porque un jugador ganó o porque se produjo un empate. El método debe indicar cuál de estas condiciones ocurrió. El tipo de retorno de esta función es String, con valores "X", "O", "DRAW", "IN_PROGRESS".
- printBoard(): Necesario para mostrar el estado del tablero durante el juego interactivo.
- getCurrentSymbol(): Necesario para determinar que jugador debe realizar el siguiente movimiento.

Resultando así la clase:

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

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

public boolean makeMove(int z) {
    if (z < 0 || z >= 7) return false;

    for (int row = 5; row >= 0; row--) {
        if (grid[z][row] == ' ') {
            grid[z][row] = currentSymbol;
            currentSymbol = (currentSymbol == 'X') ? 'O' : 'X';
            return true;
        }
    }
    return false;
}

public String isGameOver() {

    for (int row = 0; row < 6; row++) {
        for (int col = 0; col < 4; col++) {
            char symbol = grid[col][row];
            if (symbol != ' ' &&
                    grid[col][row] == symbol &&
                    grid[col+1][row] == symbol &&
                    grid[col+2][row] == symbol &&
                    grid[col+3][row] == symbol) {
                return String.valueOf(symbol);
            }
        }
    }


    for (int col = 0; col < 7; col++) {
        for (int row = 0; row < 3; row++) {
            char symbol = grid[col][row];
            if (symbol != ' ' &&
                    grid[col][row] == symbol &&
                    grid[col][row+1] == symbol &&
                    grid[col][row+2] == symbol &&
                    grid[col][row+3] == symbol) {
```

```
52                      return String.valueOf(symbol);
53                  }
54              }
55          }
56
57
58          for (int col = 0; col < 4; col++) {
59              for (int row = 3; row < 6; row++) {
60                  char symbol = grid[col][row];
61                  if (symbol != ' ' &&
62                          grid[col][row] == symbol &&
63                          grid[col+1][row-1] == symbol &&
64                          grid[col+2][row-2] == symbol &&
65                          grid[col+3][row-3] == symbol) {
66                      return String.valueOf(symbol);
67                  }
68              }
69          }
70
71
72          for (int col = 0; col < 4; col++) {
73              for (int row = 0; row < 3; row++) {
74                  char symbol = grid[col][row];
75                  if (symbol != ' ' &&
76                          grid[col][row] == symbol &&
77                          grid[col+1][row+1] == symbol &&
78                          grid[col+2][row+2] == symbol &&
79                          grid[col+3][row+3] == symbol) {
80                      return String.valueOf(symbol);
81                  }
82              }
83          }
84
85
86          boolean full = true;
87          for (int col = 0; col < 7; col++) {
88              if (grid[col][0] == ' ') {
89                  full = false;
90                  break;
91              }
92          }
93
94          if (full) return "DRAW";
95          return "IN_PROGRESS";
96      }
97
98      public void printBoard() {
99          System.out.println("  0 1 2 3 4 5 6");
100         for (int row = 0; row < 6; row++) {
101             System.out.print(row + " ");
102             for (int col = 0; col < 7; col++) {
103                 System.out.print(grid[col][row] + " ");
```

```
104             }
105             System.out.println();
106         }
107     }
108
109     public char getCurrentSymbol() {
110         return currentSymbol;
111     }
112     }
```

2.4. Clase Game

La última clase es llamada Game que ejecuta una partida interactiva entre dos jugadores y determina el resultado final.

- status (String): representa el estado actual del juego. Puede ser IN_PROGRESS, VICTORY o DRAW.
- winnerPlayerName (String): Guarda el nombre del jugador que ganó la partida.
- playerNameA (String): nombre del jugador A.
- playerNameB (String): nombre del jugador B.
- ConnectFour (ConnectFour): Instancia de la clase ConnectFour que se encarga de gestionar el tablero y la lógica del juego.

Los métodos a implementar son:

- Game(String playerNameA, String playerNameB): Inicializa una nueva partida entre los jugadores que se entregan. El estado del juego queda como IN_PROGRESS.
- play(): Este método ejecuta la partida. Se hizo uso de un ciclo que se repite hasta que uno de los jugadores gane o haya empate. En cada turno, el método makeMove de la clase ConnectFour se usará para hacer un movimiento, y después con isGameOver se revisa si ya terminó la partida. Si termina, el estado del juego pasa a VICTORY o DRAW y guarda el nombre del ganador o vacío si fue empate. Para que los jugadores puedan hacer sus jugadas, se pide que ingresen por consola (usando System.in en Java) el número de la columna donde quieren jugar. También se imprime el estado del tablero y quién debe jugar en cada turno, todo esto fue hecho con libertad creativa tal como se indicó.

Resultando así la clase:

```
1  class Game {
2      private String status;
3      private String winnerPlayerName;
4      private String playerNameA;
```

```
5       private String playerNameB;
6       private ConnectFour connectFour;
7
8       public Game(String playerNameA, String playerNameB) {
9           this.playerNameA = playerNameA;
10          this.playerNameB = playerNameB;
11          this.status = "IN_PROGRESS";
12          this.winnerPlayerName = "";
13          this.connectFour = new ConnectFour();
14      }
15
16      public String play() {
17          Scanner scanner = new Scanner(System.in);
18
19          while (status.equals("IN_PROGRESS")) {
20              connectFour.printBoard();
21
22              String currentPlayer = (connectFour.getCurrentSymbol()
                    == 'X') ? playerNameA : playerNameB;
23              System.out.println("Turno de " + currentPlayer + " ("
                    + connectFour.getCurrentSymbol() + ")");
24              System.out.print("Ingrese columna (0-6): ");
25
26              int column = scanner.nextInt();
27
28              if (connectFour.makeMove(column)) {
29                  String gameResult = connectFour.isGameOver();
30
31                  if (gameResult.equals("X")) {
32                      status = "VICTORY";
33                      winnerPlayerName = playerNameA;
34                  } else if (gameResult.equals("O")) {
35                      status = "VICTORY";
36                      winnerPlayerName = playerNameB;
37                  } else if (gameResult.equals("DRAW")) {
38                      status = "DRAW";
39                      winnerPlayerName = "";
40                  }
41              } else {
42                  System.out.println("Movimiento invalido. Intente
                        de nuevo.");
43              }
44          }
45
46          connectFour.printBoard();
47
48          if (status.equals("VICTORY")) {
49              System.out.println("Gano" + winnerPlayerName);
50              return winnerPlayerName;
51          } else {
52              System.out.println("Empate");
53              return "";          }    } }
```

3. Ánalisis

3.1. Análisis asintótico de métodos críticos

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

- addGameResult:
  - Complejidad: O(log n) donde n son los nodos en el BST.
  - La inserción en BST es O(log n) en el caso promedio.
- registerPlayer
  - Complejidad: O(1) en promedio.
  - Ambas operaciones en HashST son O(1) en promedio.
  - En el peor caso podría ser O(n) por colisiones en la tabla hash.
- checkPlayer:
  - Complejidad: O(1) en promedio.
  - La operación de HashST es O(1) en promedio.
  - En el peor caso podría ser O(n) debido a colisiones.
- winRange:
  - Complejidad: O(k log n) k siendo el número de elementos en el rango.
  - keysInRange: O(log n + k) para encontrar y recorrer los elementos en rango.
  - El bucle ejecuta k iteraciones, cada una con O(log n) para get del BST.
  - La complejidad total es O(log n + k + k·log n) = O(k log n).
- winSuccessor:
  - Complejidad: O(log n).
  - successor: O(log n) para encontrar el sucesor en el BST.
  - Complejidad total: O(log n).

Las complejidades del peor caso ocurren cuando el BST está degenerado (actúa como lista enlazada) o cuando hay muchas colisiones en la tabla hash.

3.2. Ventajas y desventajas de la implementación

- Ventajas: Una de las ventajas es que las clases del sistema están bien separadas según su funcionalidad, lo que hace que el código sea fácil de mantener y comprender.
- Otra ventaja fue el uso de árbol binario de busqueda y la tabla de hash, lo cual permite mantener orden en el código y ejecutarlo de forma rápida y precisa.
- Desventajas: se tiene la desventaja de que al no usar las estructuras normales de Java, hay posibilidades de cometer errores o que falten validaciones, cosa que no pasaría en clases como HashMap o TreeMap.
- Otra desventaja es que el hash puede generar colisiones con una cantidad muy grande de jugadores.

3.3. Desafíos encontrados

Uno de los desafíos más grandes fue el método isGameOver en la clase ConnectFour, porque había que revisar si un jugador ganaba en todas las direcciones posibles. Al principio fue difícil que no se saliera de los límites del tablero y que detectara bien todas las combinaciones. Para resolverlo, arreglamos los ciclos de los métodos para que no se salieran del array.

3.4. Extensión del sistema

Para extender el sistema y permitir la realización de torneos, se necesita agregar una funcionalidad que permita crear y administrar varios torneos de forma independiente.

En el diseño, se incorpora una clase encargada de gestionar los torneos, organizar los partidos y controlar el avance por rondas. También se deben hacer ajustes en la información de los jugadores para registrar su ranking, historial y estadísticas dentro de cada torneo.

Esta clase podría usar una matriz para organizar los emparejamientos entre jugadores y llevar una tabla de puntajes que se actualice automáticamente según los resultados. Además, habría que modificar Scoreboard para que pueda registrar múltiples partidas seguidas y calcular el total de puntos de cada jugador. También se deberán incluir reglas de eliminación o descalificación, dependiendo del tipo de torneo.

3.5. Propuesta estructura de datos

Para reemplazar el BST implementado en el código se puede utilizar TreeMap, el cual es un mapa basado en un árbol que mantiene keys ordenadas y distintas operaciones en complejidad logarítmica. Para poder reemplazar el uso de HashST se puede

utilizar un HashMap, el cual implementa una tabla Hash para manejo de colisiones uso dinámico.

Dentro de las ventajas de poder usar estas estructuras, se encuentra tener un código más confiable debido a su extendido uso, por lo tanto habrá mayor control sobre el código.

Dentro de las desventajas está la dependencia de la librería estándar de Java, por ende tiene muchas limitaciones en cuanto a personalización para el uso.

3.6. Prueba de implementación

Se probó el sistema jugando partidas entre compañeros. El juego se turnaba correctamente, el tablero se actualizaba en cada movimiento y los resultados se guardaban de la manera adecuada.

Varias de las partidas terminaron en victorias y también empates, lo cual demostró la correcta funcionalidad del código. El menú también ayudó a ver información relevante sobre los jugadores registrados.

4. Conclusión

En este laboratorio se implementaron estructuras de datos en Java, específicamente árbol binario de búsqueda y tabla de hash, y se integraron en un sistema que simula el juego Conecta 4. Se experimentó para entender mejor cómo funcionan internamente estas estructuras, cómo se recorren, cómo se insertan datos y qué problemas pueden aparecer cuando se combinan entre sí.

También se puso en práctica la programación orientada a objetos propia de Java, dividiendo el programa en clases separadas que interactúan entre ellas, como Player, Scoreboard, ConnectFour y Game. Nos enfrentamos a desafíos reales como validar entradas, mantener la consistencia entre estructuras y detectar las victorias en el tablero.

En la vida real estas estructuras se pueden aplicar en sistemas donde se necesita control total sobre el diseño del trabajo, por ejemplo en servidores de videojuegos con rankings online. Otro ejemplo sería una aplicación de ranking académico que ordena estudiantes según el rendimiento y permite hacer búsquedas por rango o puntaje, algo que se hizo de forma parecida con winRange y winSuccessor.

La librería estándar de Java puede resultar un poco extensa en comparación a otros lenguajes de programación, además de que su rendimiento no es necesariamente el más óptimo en muchos casos. Sin embargo, esto permitió entender cada una de las estructuras utilizadas y verlas funcionar de manera directa y concreta.