

## Guía Práctica No. 5: Búsqueda

Esta guía práctica corresponde a las Técnicas de Búsqueda.

Esta práctica cuenta con un framework de búsqueda que puede accederse a través del classroom de github [disenho-de-algoritmos-unrc-2021](#).

Esta práctica no tiene entrega formal. Su resolución es opcional.

1. Utilizando el framework de búsqueda disponible en el repositorio git de la materia y siguiendo el ejemplo de la implementación de *breadth first search*, realice una implementación de *depth first search* y una implementación de *iterative deepening*, para problemas de búsqueda convencionales.<sup>1</sup>

Cada implementación debe permitir incorporar problemas de búsqueda concretos con facilidad, sobre los cuales el algoritmo definido pueda utilizarse sin realizar modificaciones. La implementación debe además incluir mecanismos para que, en los casos en los cuales se encuentre una solución, se pueda reconstruir el camino desde el estado inicial a la misma.

2. Utilizando mecanismos de reutilización propios de la programación orientada a objetos, realice una implementación de *best first search* para problemas de búsqueda genéricos.

La implementación debe permitir incorporar problemas de búsqueda informada con facilidad, sobre los cuales el algoritmo definido pueda utilizarse sin realizar modificaciones. La implementación debe además incluir mecanismos para que, en los casos en los cuales se encuentre una solución, se pueda reconstruir el camino desde el estado inicial a la misma.

3. Estudie el problema de las 8 reinas, y presente un modelo del mismo como un problema de búsqueda. Proponga además una función heurística para este problema que no haya sido mencionada en clases.

Implemente soluciones al problema usando las estrategias de visita *depth first search*, *best first search*, *iterative deepening* y *breadth first search*, tales que, si se encuentran soluciones, las reporten e impriman el camino desde el estado inicial a la solución encontrada.

4. Utilizando mecanismos de reutilización propios de la programación orientada a objetos, realice una implementación de min max para problemas de búsqueda con adversarios. La implementación debe permitir incorporar problemas de búsqueda con adversarios concretos con facilidad, sobre los cuales el algoritmo definido pueda utilizarse sin realizar modificaciones. La implementación debe tener en cuenta una profundidad máxima para la construcción del árbol de juego, de manera tal que si ésta es alcanzada y aún no se consiguió una hoja, permita valorar los estados del último nivel, y así completar la valoración de los nodos del árbol. La técnica de búsqueda, aplicada a un problema particular en un estado particular, debe indicar cuál es la mejor forma de “jugar” en la movida corriente.

5. Desarrolle una aplicación que permita jugar al conocido juego del *Ta-Te-Ti*, instancie la solución genérica de búsqueda Min Max, definida en el ejercicio previo, para conseguir estrategias óptimas de juego.

<sup>1</sup>Recuerde que el algoritmo de búsqueda *depth first search* puede no encontrar una solución, cuando ésta existe, debido a que se pierde por una rama infinita

6. El algoritmo de búsqueda adversaria MinMax puede mejorarse significativamente utilizando poda alfa-beta. Dé dos ejemplos de árboles de juego, en uno de los cuales la poda alfa-beta mejore significativamente a MinMax podando al menos dos tercios de los nodos del árbol, mientras que en el otro no produzca ninguna poda.
7. Utilizando mecanismos de reutilización propios de la programación orientada a objetos, realice una implementación de *min max con poda alfa-beta* para problemas de búsqueda con adversarios. La implementación debe permitir incorporar problemas de búsqueda con adversarios concretos con facilidad, sobre los cuales el algoritmo definido pueda utilizarse sin realizar modificaciones. La implementación debe tener en cuenta una profundidad máxima para la construcción del árbol de juego, de manera tal que si ésta es alcanzada y aún no se consiguió una hoja, permita valorar los estados del último nivel, y así completar la valoración de los nodos del árbol. La técnica de búsqueda, aplicada a un problema particular en un estado particular, debe indicar cuál es la mejor forma de “jugar” en la movida corriente.

8. El *2048* es un popular juego de único jugador, cuyo objetivo es deslizar piezas en una grilla de manera tal de combinar las mismas y conseguir formar una con el número 2048.

El juego se juega en una grilla de 4 x 4 casillas, en la cual se alojan diferentes piezas numeradas.

El estado inicial del juego es la grilla con exactamente dos piezas, ubicadas en posiciones aleatorias, y cuyos valores (también decididos aleatoriamente) pueden ser 2 o 4. El juego admite cuatro posibles movimientos, a izquierda, derecha, arriba y abajo. Al tomar una de estas direcciones, las piezas se deslizan en la dirección correspondiente, hasta chocar con otras piezas o contra el borde de la grilla. Si dos piezas con el mismo número se chocan en el movimiento, las mismas se combinan, formando una única pieza, cuyo valor es la suma de las dos piezas que la originaron. Además, luego de cada movimiento, se genera una nueva pieza, en una de las posiciones libres de la grilla, con valor 2 o 4; tanto el valor de la nueva pieza como su ubicación se deciden aleatoriamente. El jugador gana el juego cuando consigue formar una pieza con el valor 2048. Por el contrario, el jugador pierde si el tablero queda cubierto de piezas sin posibilidad de efectuar movimientos que liberen al menos una casilla.

Se desea desarrollar una aplicación que permita, a un jugador controlado por la computadora, jugar al *2048*. Para decidir cómo jugar, debe utilizarse la técnica de búsqueda en problemas con adversarios conocida como *MinMax con poda alfa-beta*. Nótese que, si bien el *2048* es un juego “solitario” (de un único jugador), la generación de una nueva pieza para cubrir una celda libre del tablero es un movimiento no controlado por el jugador a programar, y que por lo tanto debe ser tratada como una operación de “adversario”.

La función de valoración de estados no terminales, necesaria para la implementación de la técnica *MinMax con poda alfa-beta*, quedará a criterio de los desarrolladores (una bastante elemental es simplemente tomar la pieza de mayor valor en el tablero).

9. El *Duidoku* es una versión de Sudoku de dos jugadores. En este juego, los jugadores alternan movimientos, que consisten en posicionar valores en el tablero de manera tal de no generar conflictos en el mismo. Para un tablero clásico de 9 x 9, los conflictos son los usuales de Sudoku: no puede haber valores repetidos en una misma fila, columna, o región de 3 x 3; y los valores permitidos en el tablero son enteros entre 1 y 9. Gana el juego aquel jugador que es capaz de realizar el último movimiento en el mismo, es decir, aquel que fuerza que su adversario no pueda colocar ningún valor en el tablero.

Como referencia, puede considerar la versión online del juego en <http://www.duidoku.com>.

Se requiere implementar un programa que juegue automáticamente a *Duidoku* contra un jugador humano, en un tablero clásico de 9 x 9. La solución debe implementarse utilizando la técnica *Minmax con poda alfa-beta*. No es necesario soportar una interfaz gráfica, es suficiente con una interfaz de consola.