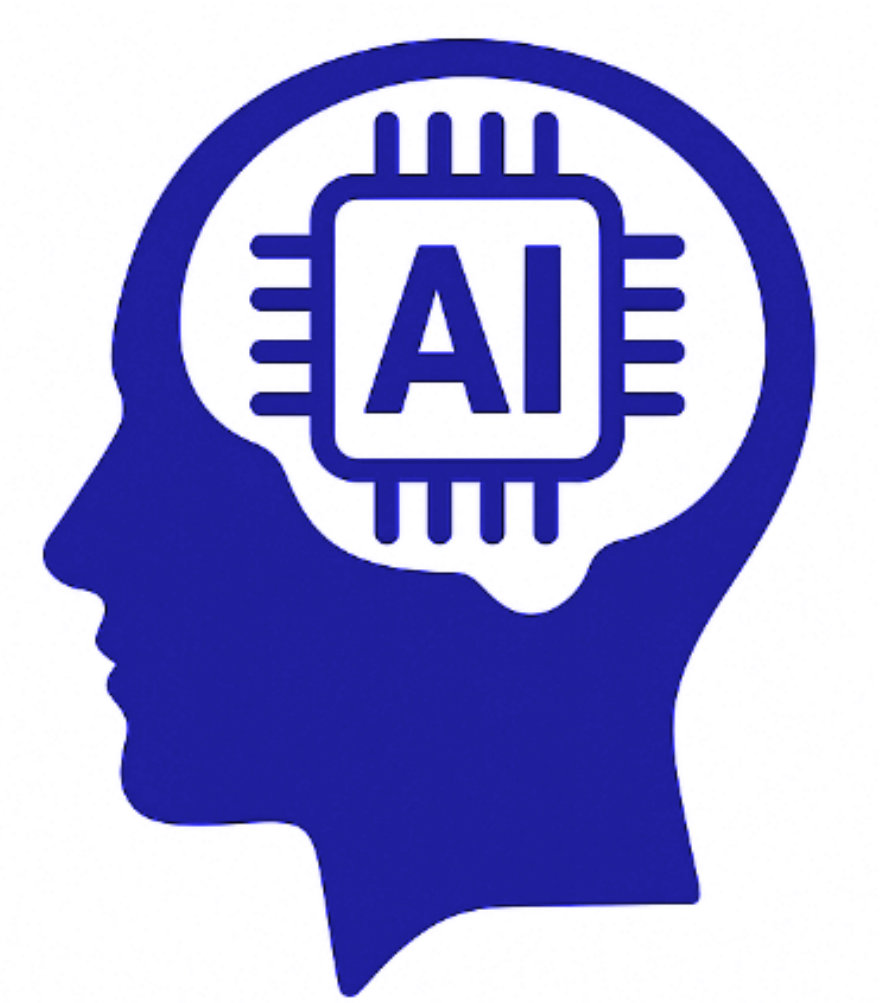
# Entrega Práctica 2 - Inteligencia Artificial

**Resolución de problemas y búsqueda. Búsqueda informada.**




**Autor:** Fernando Pastor Peralta  (897113)

**Fecha de entrega:** 21/10/2025

**Grupo:** Martes B, 16:00-19:00

# Índice

**Introducción:** ..... **3**
**Metodología:** ..... **3**
Implementación: ..... 3
Análisis de los resultados: ..... 4
**Conclusión:** ..... **5**
**Anexo:** ..... **6**

# Introducción:

El objetivo de esta práctica es familiarizarse con el código en Java para la resolución de problemas de búsqueda informada y diferenciar sus características. En concreto, nos seguiremos centrando en el problema del 8-puzzle.

Se compararon algoritmos de búsqueda ciega (BFS - Breadth-First Search e IDS - Iterative Deepening Search) con búsqueda informada A*, utilizando dos heurísticas:

- Heurística de fichas descolocadas (h1): Cuenta el número de fichas que no están en su posición objetivo.

- Heurística de Manhattan (h2): Suma las distancias Manhattan de cada ficha a su posición objetivo, se ignora el espacio.

La eficiencia se mide mediante el número de nodos generados y el factor de ramificación efectivo b*, calculado mediante la resolución de la ecuación N = b* (b*d - 1) / (b* - 1), mediante el método de bisección. Se realizaron 100 experimentos con distintas profundidades, más en concreto con profundidades “d” desde 2 hasta 24, generando 100 estados iniciales aleatorios por cada profundidad. También hay que destacar que se limita IDS para profundidades mayores a 10, debido a su gran coste espacio-temporal.

# Metodología:

## Implementación:

Nuestra implementación parte del código AIMA proporcionado en Java, en el que se han adaptado o creado clases para la resolución del problema a resolver.

Definimos nuestro programa de test para plasmar la resolución de la práctica en el método main de la clase EightPuzzlePract2. Esta clase tiene una estructura similar a la de EightPuzzlePract1 de la primera práctica, sobre todo en el uso de los métodos eightPuzzleSearch() y tipoSearch() para la ejecución de los distintos algoritmos partiendo de un estado inicial dado, y también es similar en la presentación de resultados por pantalla, siguiendo la misma idea de presentar cabecera y después ir rellenando fila a fila mediante una función. Dentro del método main hacemos distintas cosas para la resolución del problema:

1. Iniciamos el estado objetivo, definido en una nueva clase EightPuzzleGoalTest2, que obviamente es una pequeña modificación del EightPuzzleGoalTest inicial, para poder modificar tanto el estado objetivo (método setGoalState()), como para que las heurísticas tenga en cuenta el estado objetivo definido (método getGoalState()).

2. Configuramos las heurísticas para que trabajen con un determinado estado objetivo, para ello se han modificado las definiciones originales de ambas heurísticas, y se han creado ManhattanHeuristicFunction2 y MisplacedTilleHeuristicFunction2.

3. Generamos un total de 100 estados iniciales aleatorios por cada profundidad:

    - Generamos estados iniciales mediante la clase GenerateInitialEightPuzzleBoard, que fue proporcionada por los profesores de la asignatura, específicamente contiene el método EightPuzzleBoard random(int depth), que genera un estado inicial que se puede alcanzar en por lo menos d movimientos. Se ha modificado dicho método, ya que dentro del mismo solo se aceptan movimientos si podemos mover el hueco a esa posición o si el estado resultante no fué visitado, esto es bueno, sin embargo, si el tablero actual llega a un estado donde todos los posibles movimientos son inválidos, lo que provoca es que el código se quede en un bucle sin hacer nada. Para profundidades pequeñas (2-10), es raro atascarse, pero para profundidades más altas, es más probable que pase, por ello la modificación.

    - Con estos estados iniciales, llamamos a eightPuzzleSearch() con cada algoritmo, el resultado de la llamada representa los nodos generados por cada uno de ellos, en cada iteración, guardaremos los resultados de cada algoritmo mediante un array que funciona como sumatorio.

4. Calculamos las medias de nodos generados.

5. Calculamos b* en cada caso, mediante el método calcularBestrella(), que funciona gracias a la clase proporcionada por el profesorado llamada Biseccion, la cuál proporciona el metodoDeBiseccion() en la que se resuelve la ecuación $N=b*(b*d-1)/(b*-1)$ (en nuestro caso mediante el intervalo [1.0001, 10] con tolerancia 0.001).

6. Por último se imprime cada fila con su estilo correspondiente, y se pasa a la siguiente iteración (si hay más profundidades que explorar).

# Análisis de los resultados:

Tras probar el algoritmo varias veces, podemos sacar en claro algunas conclusiones:

- Los nodos generados crecen de forma exponencial según d, tal y como se espera en los árboles de búsqueda O(b^d), donde b es el factor de ramificación y d la profundidad.

- BFS e IDS escalan peor que A*, confirmando la eficiencia de las heurísticas informadas, sobre los algoritmos no informados.

- BFS decrementa su b* de 2,37 a 1,54, mostrando la efectiva ramificación mientras profundizamos, gracias a los descartes de estados inválidos y duplicados (búsqueda en grafo).

- Es muy notable la explosión de nodos generados por IDS según aumentamos la profundidad (por ello es buena idea parar el algoritmo de forma temprana para no tener tiempos de espera especialmente altos). Además, b* se mantiene constante, es consistente.

- A* es muy eficiente frente a BFS e IDS. Con la heurística de fichas descolocadas (h1), obtenemos mejores resultados, tanto en nodos generados, como en b*. Por otro lado, la heurística de Manhattan (h2), es aún mejor, casi generando aproximadamente 10 veces menos nodos que h1 y con un b* aún menor.

- El aumento de tiempo para que terminen los algoritmos crece considerablemente según aumenta la profundidad.

- Nuestros nodos generados son menores porque el método random(d) genera d pasos, sin asegurarse de que el estado objetivo esté en profundidad d. Sin embargo en el enunciado del PDF, se asegura de que el estado objetivo esté a una profundidad d.

En el anexo se muestra la foto de los resultados tras la ejecución del programa.

# Conclusión:

Hemos comprendido la eficiencia de los algoritmos informados frente a los algoritmos ciegos en el problema del 8-puzzle, confirmando que las heurísticas de fichas descolocadas (h1) y Manhattan (h2) reducen drásticamente la optimalidad.

Los resultados han mostrado un crecimiento exponencial en los nodos generados según la profundidad en BFS e IDS (este último destaca por su explosión exponencial y su b* constante). Mientras que A* resulta más eficiente, y en concreto h2 genera aproximadamente 10 veces menos nodos que h1, además de que b* se reduce más.

# Anexo:

Un ejemplo de la ejecución:

| | Nodos Generados | | | | b* | | | |
|---|---|---|---|---|---|---|---|---|
| d | BFS | IDS | A*h(1) | A*h(2) | BFS | IDS | A*h(1) | A*h(2) |
| 2 | 8 | 11 | 6 | 6 | 2,19 | 2,70 | 1,79 | 1,79 |
| 3 | 18 | 32 | 9 | 9 | 2,16 | 2,75 | 1,58 | 1,58 |
| 4 | 36 | 95 | 12 | 12 | 2,10 | 2,80 | 1,45 | 1,45 |
| 5 | 77 | 333 | 16 | 15 | 2,11 | 2,94 | 1,42 | 1,37 |
| 6 | 132 | 889 | 23 | 20 | 2,02 | 2,89 | 1,40 | 1,34 |
| 7 | 228 | 2403 | 30 | 24 | 1,96 | 2,86 | 1,37 | 1,30 |
| 8 | 372 | 6477 | 42 | 28 | 1,91 | 2,84 | 1,36 | 1,27 |
| 9 | 629 | 18211 | 69 | 37 | 1,88 | 2,83 | 1,40 | 1,28 |
| 10 | 895 | 39718 | 96 | 46 | 1,82 | 2,76 | 1,40 | 1,26 |
| 11 | 1592 | --- | 148 | 61 | 1,82 | --- | 1,41 | 1,27 |
| 12 | 2415 | --- | 225 | 81 | 1,79 | --- | 1,42 | 1,27 |
| 13 | 4140 | --- | 333 | 103 | 1,78 | --- | 1,43 | 1,27 |
| 14 | 5706 | --- | 488 | 143 | 1,75 | --- | 1,43 | 1,28 |
| 15 | 9270 | --- | 737 | 174 | 1,74 | --- | 1,43 | 1,28 |
| 16 | 12363 | --- | 987 | 222 | 1,71 | --- | 1,43 | 1,28 |
| 17 | 19985 | --- | 1690 | 330 | 1,70 | --- | 1,44 | 1,29 |
| 18 | 24763 | --- | 1972 | 371 | 1,67 | --- | 1,43 | 1,28 |
| 19 | 36995 | --- | 3083 | 509 | 1,66 | --- | 1,43 | 1,28 |
| 20 | 53258 | --- | 4743 | 653 | 1,64 | --- | 1,44 | 1,28 |
| 21 | 66450 | --- | 5944 | 826 | 1,62 | --- | 1,43 | 1,28 |
| 22 | 71720 | --- | 7607 | 895 | 1,59 | --- | 1,42 | 1,27 |
| 23 | 84625 | --- | 8785 | 1079 | 1,57 | --- | 1,41 | 1,27 |
| 24 | 92575 | --- | 11127 | 1053 | 1,54 | --- | 1,40 | 1,25 |

Otro ejemplo:

| | Nodos Generados | | | | b* | | | |
|---|---|---|---|---|---|---|---|---|
| d | BFS | IDS | A*h(1) | A*h(2) | BFS | IDS | A*h(1) | A*h(2) |
| 2 | 8 | 11 | 6 | 6 | 2,37 | 2,85 | 1,79 | 1,79 |
| 3 | 19 | 34 | 9 | 9 | 2,26 | 2,81 | 1,66 | 1,66 |
| 4 | 35 | 90 | 12 | 12 | 2,08 | 2,76 | 1,45 | 1,45 |
| 5 | 74 | 311 | 16 | 15 | 2,08 | 2,90 | 1,42 | 1,37 |
| 6 | 127 | 841 | 23 | 20 | 2,00 | 2,86 | 1,39 | 1,34 |
| 7 | 231 | 2459 | 32 | 24 | 1,97 | 2,87 | 1,38 | 1,31 |
| 8 | 343 | 5620 | 44 | 29 | 1,89 | 2,78 | 1,37 | 1,28 |
| 9 | 605 | 17181 | 61 | 35 | 1,87 | 2,81 | 1,38 | 1,26 |
| 10 | 951 | 45799 | 99 | 46 | 1,84 | 2,80 | 1,40 | 1,27 |
| 11 | 1472 | --- | 142 | 62 | 1,80 | --- | 1,40 | 1,27 |
| 12 | 2392 | --- | 217 | 77 | 1,79 | --- | 1,42 | 1,27 |
| 13 | 4039 | --- | 343 | 117 | 1,78 | --- | 1,43 | 1,29 |
| 14 | 5985 | --- | 509 | 137 | 1,75 | --- | 1,43 | 1,28 |
| 15 | 9318 | --- | 805 | 209 | 1,74 | --- | 1,44 | 1,30 |
| 16 | 12950 | --- | 1029 | 247 | 1,71 | --- | 1,43 | 1,29 |
| 17 | 19642 | --- | 1626 | 298 | 1,70 | --- | 1,44 | 1,28 |
| 18 | 29246 | --- | 2393 | 450 | 1,68 | --- | 1,44 | 1,29 |
| 19 | 34561 | --- | 2994 | 499 | 1,65 | --- | 1,43 | 1,28 |
| 20 | 45601 | --- | 4221 | 622 | 1,63 | --- | 1,43 | 1,28 |
| 21 | 67533 | --- | 6300 | 868 | 1,62 | --- | 1,43 | 1,28 |
| 22 | 67116 | --- | 7062 | 832 | 1,58 | --- | 1,42 | 1,26 |
| 23 | 79590 | --- | 8167 | 1024 | 1,56 | --- | 1,40 | 1,26 |
| 24 | 95778 | --- | 11738 | 1212 | 1,54 | --- | 1,40 | 1,26 |