Sistemas Inteligentes
Práctica 1: 15-Puzzle

Descripción de la práctica

Se trata de implementar los siguientes algoritmos de búsqueda para resolver el 15–puzzle:

- Búsqueda primero en anchura
- Búsqueda primero en profundidad
- Búsqueda en profundidad iterativa
- Búsqueda con heurística de fichas descolocadas
- Búsqueda con heurística de distancias Manhattan

Se considerará como objetivo la siguiente configuración:

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Hay que tener en cuenta que hay configuraciones del 15–puzzle que no se pueden resolver con ninguna combinación de movimientos del hueco. Por ejemplo:

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

Por tanto, los casos de prueba se deben generar mediante varios movimientos del hueco partiendo del estado objetivo. En total pueden haber sólo $16! / 2 \simeq 10^{13}$.

Código fuente

puzzle15 / Main.java

Contiene la función main, el “esqueleto” del programa y 10 puzzles para probar los algoritmos.

No se debe alterar la función **main** para facilitar la corrección de las prácticas.

En esta clase hay que implementar las siguientes funciones:

- busquedaAnchura
- busquedaProfundidad
- busquedaProfundidadIterativa
- busquedaHeuristicaDescolocadas
- busquedaHeuristicaManhattan

puzzle15 / Estado.java

Representa un estado del 15–puzzle.

Es necesario implementar las funciones que contengan "UnsupportedOperation–Exception", y además, comprobar que los objetos se crean correctamente, es decir, con todas las fichas y sin repetir fichas.

puzzle15 / Heuristica.java

Es el interfaz que hay que implementar definir las dos heurísticas:

- Heurística de fichas descolocadas
- Heurística de distancias Manhattan

En este fichero no hay que modificar nada.

puzzle15 / Operador.java

Este interfaz se puede implementar para definir los cambios de estado.

En este fichero no hay que modificar nada.

Paquete lib

lib / XPriorityQueue.java

Implementa una cola de prioridad. Esta clase es necesaria para las búsquedas heurísticas.

lib / XHashSet.java

Implementa el tipo abstracto de datos conjunto. Este conjunto permite comprobar si hay un estado idéntico a otro, aunque se haya llegado a ellos a distintas profundidades.

Detalles a tener en cuenta

Búsqueda en anchura

El control de estados repetidos se puede realizar mediante un objeto **HashSet<Estado>**.

Búsqueda en profundidad

El control de estados repetidos se puede realizar con un objeto **XHashSet<Estado>**.

En este algoritmo se considerará un estado repetido si ya se encontraba en el conjunto de estados repetidos a una profundidad menor o igual. Si un estado ya estaba a una profundidad mayor, deberá sustituir al que se encontraba en el conjunto y no se le considerará repetido.

Heurística de fichas descolocadas

Consiste en contar el número de fichas que no están en la posición del estado objetivo. No hay que tener en cuenta el hueco para asegurarse de que la heurística es optimista.

En el siguiente ejemplo la distancia estimada hasta el objetivo sería 5.

1	2		7
4	5	3	6
8	9	10	11
12	13	14	15

Heurística de distancias Manhattan

En esta heurística se suman las distancias que tienen que recorrer las fichas para estar en las posiciones del estado objetivo. No hay que tener en cuenta el hueco para que la heurística sea optimista.

En el ejemplo anterior la distancia estimada hasta el objetivo sería 6.

Normas de entrega

La práctica se realizará de forma individual o por parejas.

Hay que entregar un fichero zip con todo el código Java necesario para compilar la práctica, y una pequeña memoria en PDF donde se indiquen los resultados obtenidos con los diferentes casos de prueba:

- Configuración inicial del puzzle.
- Profundidad de la solución.
- Cuántos nodos se han generado.
- Tiempo de ejecución.
- Explicación de los resultados obtenidos comparando con los demás algoritmos.

La fecha límite para entregar la práctica es el 8 de Noviembre, y se debe realizar mediante una tarea de PoliformaT.