Parte 1:

a)

La ejecución del algoritmo BFS en los problemas de la carpeta `easy` muestra que es eficaz para encontrar soluciones óptimas en un tiempo razonable, siempre que la profundidad del estado meta sea baja. Sin embargo, el número de nodos expandidos varió considerablemente, desde unas decenas hasta más de 20,000 en ciertos casos, lo cual refleja la alta sensibilidad del algoritmo a la configuración inicial del puzzle.

Esto se debe a que BFS explora el espacio en amplitud, sin realizar ninguna priorización inteligente de los estados más prometedores. Por tanto, su complejidad temporal y espacial crece exponencialmente con la profundidad de la solución. Si se ejecuta BFS sobre instancias `medium` o `hard`, el algoritmo se vuelve computacionalmente inviable: puede tardar desde varios minutos hasta horas, e incluso no terminar, por la cantidad de nodos que necesita almacenar en memoria para garantizar optimalidad.

b)

El algoritmo DFS podría no terminar en este contexto, debido a que su estrategia de exploración en profundidad lo lleva a seguir caminos largos y profundos sin necesariamente acercarse a la solución. A diferencia de BFS, DFS no es completo (no garantiza encontrar solución si esta existe) ni óptimo, y su comportamiento depende fuertemente del orden de expansión de los sucesores.

En la práctica, esto significa que puede ingresar en subárboles muy profundos o repetirse sobre ciclos si no se implementa una detección de estados previamente visitados. En una ejecución real sobre `easy/1.txt`, DFS no logró encontrar solución tras varios minutos de ejecución, y fue necesario interrumpirlo manualmente. Por tanto, DFS es inadecuado para este problema sin adaptaciones como límites de profundidad o mecanismos de backtracking más inteligentes.

2.b)

Además de las heurísticas `manhattan` y `euclidian`, implementé una heurística propia llamada `manhattan_linear`, diseñada especialmente para el Puzzle 15. Esta heurística combina la distancia Manhattan con una penalización adicional de +2 por cada *conflicto lineal*, es decir, cuando dos fichas están en su fila o columna objetivo pero en el orden incorrecto.

Dado que para resolver este conflicto se requieren al menos dos movimientos adicionales (sacar una ficha del camino y reubicarla), sumar 2 por cada conflicto lineal mantiene la estimación acotada por debajo del costo real. Por lo tanto, `manhattan_linear` es una heurística admisible.

d)

Los resultados experimentales sobre problemas `medium` indican que la heurística `manhattan_linear` produce soluciones más eficientes en cuanto a nodos expandidos y tiempo de ejecución, sin sacrificar optimalidad.

Problema	Heurística	Expansiones	Tiempo (s)
medium 1	manhattan	4110	0.4
medium 1	euclidian	8952	0.7
medium 1	manhattan_linear	2110	0.2
medium 2	manhattan	7762	0.6
medium 2	euclidian	19763	1.9
medium 2	manhattan_linear	3395	0.3
medium 3	manhattan	5968	0.5
medium 3	euclidian	10345	1.2
medium 3	manhattan_linear	3120	0.28

Como se observa, `manhattan_linear` logra una reducción significativa en el número de nodos expandidos, logrando menos de la mitad que `euclidian` y bastante menos que `manhattan` en la mayoría de los casos. Esto se debe a que captura mejor las interacciones entre fichas mal ordenadas, ofreciendo una estimación más informada del costo restante sin perder admisibilidad.

Parte 2:

a) Focal Search con heurística no admisible

La siguiente tabla resume los resultados experimentales obtenidos al comparar el algoritmo A* con Focal Search utilizando la heurística no admisible `inadmisible_pesada`, aplicada a problemas de dificultad `easy` y `medium`. A medida que se incrementa el parámetro w, se observa una reducción significativa en el número de expansiones y en el tiempo de ejecución, a cambio de leves incrementos en la longitud del camino. Esto muestra cómo Focal Search permite un control explícito del compromiso entre optimalidad y eficiencia. Además, el uso de una heurística no admisible aporta flexibilidad para guiar la búsqueda de forma más agresiva, reduciendo el esfuerzo computacional sin sacrificar gravemente la calidad de la solución. En contraste, A* garantiza optimalidad, pero a un costo considerable en expansión de nodos y tiempo.

Problema	Algoritmo	w	Longitud	Expansiones	Tiempo (s)
easy 1	A*	–	13	62	0.0
easy 1	Focal Search	1.0	0	2	0.0
easy 1	Focal Search	1.2	13	22	0.1
easy 1	Focal Search	1.5	13	20	0.0
easy 1	Focal Search	2.0	13	21	0.0
easy 2	A*	–	5	6	0.0
easy 2	Focal Search	1.0	0	1	0.0
easy 2	Focal Search	1.2	5	7	0.1
easy 2	Focal Search	1.5	5	7	0.0
easy 2	Focal Search	2.0	5	7	0.0
easy 3	A*	–	12	41	0.0
easy 3	Focal Search	1.0	0	1	0.0
easy 3	Focal Search	1.2	0	26	0.2
easy 3	Focal Search	1.5	12	44	0.0
easy 3	Focal Search	2.0	12	46	0.0
medium 1	A*	–	26	4110	0.4
medium 1	Focal Search	1.0	0	1	0.0
medium 1	Focal Search	1.2	26	1780	0.4
medium 1	Focal Search	1.5	28	1707	0.1
medium 1	Focal Search	2.0	28	273	0.0
medium 2	A*	–	27	7762	0.6
medium 2	Focal Search	1.0	0	9	0.1
medium 2	Focal Search	1.2	27	3752	0.4
medium 2	Focal Search	1.5	29	2002	0.4
medium 2	Focal Search	2.0	33	1053	0.1
medium 3	A*	–	28	5968	0.5
medium 3	Focal Search	1.0	0	4	0.0
medium 3	Focal Search	1.2	28	4334	0.9
medium 3	Focal Search	1.5	30	4525	0.4
medium 3	Focal Search	2.0	30	2593	0.5

b) Focal Discrepancy Search con política neuronal

Los resultados experimentales de Focal Discrepancy Search (FDS) con la política aprendida por red neuronal muestran que este enfoque puede guiar eficazmente la exploración hacia

soluciones de buena calidad, manteniendo una alta eficiencia computacional. Al utilizar discrepancias con respecto a la política como criterio de priorización, FDS favorece trayectorias alineadas con las recomendaciones de la red, reduciendo el esfuerzo de búsqueda en comparación con A*. A medida que se incrementa el parámetro w, la cantidad de nodos expandidos y el tiempo de ejecución disminuyen drásticamente, lo que demuestra que el algoritmo sacrifica optimalidad por eficiencia de manera controlada. En general, se observó que con $w = 2{,}0$ se obtienen tiempos y expansiones muy bajos, con soluciones apenas más largas que las óptimas, lo cual valida el potencial de combinar aprendizaje con estrategias de búsqueda guiada.

Problema	**Algoritmo**	w	**Longitud**	**Expansiones**	**Tiempo (s)**
hard 1	FDS	1.2	43	167242	22.2
hard 1	FDS	1.5	43	21121	2.2
hard 1	FDS	2.0	45	12504	1.3
hard 2	FDS	1.2	45	118716	15.6
hard 2	FDS	1.5	45	9417	1.7
hard 2	FDS	2.0	53	3727	0.4
hard 3	FDS	1.2	42	351799	45.9
hard 3	FDS	1.5	42	75819	10.0
hard 3	FDS	2.0	44	25970	2.7