Grupo AmaIA
Lander San Milán Langa
Amaia Acha Aristegui

ASSEMBLER ROBOT

Para la resolución del problema hemos propuesto el siguiente enfoque, estado inicial, acciones, efectos, coste y estado final.

En un principio, nuestro objetivo fue aplicar en cada mapa una vez cada algoritmo, buscando todos los números en una sola ejecución. Para los primeros 2 mapas esto funcionó, pero tras intentar resolver los mapas 3 y 4 nos dimos cuenta de que la complejidad computacional era demasiado elevada como para resolverlo en una cantidad de tiempo razonable.

Para intentar solucionar esas instancias optamos por cambiar el **enfoque** de solucionar el problema, en vez de buscar todos los puntos de ensamblaje en orden en una sola ejecución de cada algoritmo subdividimos el problema en varios subproblemas, buscando una sola pieza en cada ejecución de cada algoritmo. Para realizar ese proceso ejecutamos los algoritmos dentro de un bucle que recorre la cantidad de puntos de ensamblaje (*problem$assembly_cuantity*).

El **estado inicial** se compone de una matriz en la que se incorporan los datos leídos del archivo *assembler-robot-X.txt*. Hemos definido con números cada casilla de la matriz simbolizando lo siguiente:

- 0 → Casilla vacía
- 1...n → Punto de ensamblaje con su orden
- -2 → Cabeza del robot
- -3 → Extensión del robot
- -4 → Casilla prohibida

Las **acciones** se considerarán como mover la cabeza del robot 1 casilla a cualquiera de las siguientes direcciones: Arriba, Abajo, Izquierda, Derecha. Estas acciones serán aplicables solo si la casilla está vacía, se respetan los límites de la matriz, y en caso de pasar por un punto de ensamblaje, que sea el adecuado. Los resultados finales pueden variar dependiendo del **orden** en el que se pasen las diferentes acciones al programa.

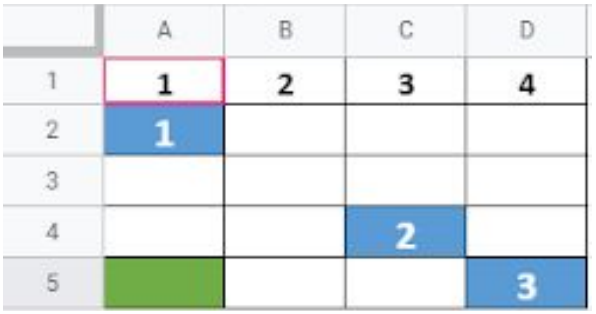
El **efecto** de cada acción será sustituir la casilla actual por un -3, la casilla a la que se accede será sustituida por un -2.

El **coste** de cada acción será de 1, ya que en cada movimiento solo avanzamos 1 casilla.

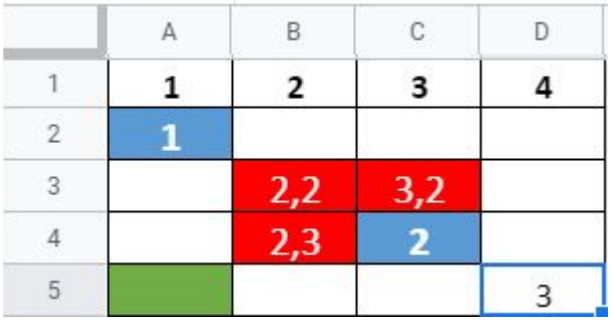
Se considerará **estado final** al estado que no contenga el punto de ensamblaje del que se está haciendo la búsqueda, es decir, la cabeza del robot haya llegado al punto de ensamblaje correspondiente.

Mapas 1 y 2

	name	solution	runtime	actions	cost
1	Breadth First Search	TRUE	0.25 secs	9	9
2	Breadth First Search + GS	TRUE	0.06 secs	9	9
3	Depth First Search	TRUE	0.04 secs	15	15
4	Depth First Search + GS	TRUE	0.04 secs	15	15
5	Depth First Search - Limit=6	TRUE	0.03 secs	15	15
6	Depth First Search - Limit=6 + GS	TRUE	0.03 secs	15	15
7	Depth First Search - Limit=10	TRUE	0.06 secs	15	15
8	Depth First Search - Limit=10 + GS	TRUE	0.04 secs	15	15
9	Iterative Deeping Search - It:3	TRUE	0.10 secs	9	9
10	Iterative Deeping Search - It:3 + GS	TRUE	0.10 secs	9	9
11	Greedy Best-First Search	TRUE	0.02 secs	9	9
12	Greedy Best-First Search + GS	TRUE	0.04 secs	9	9
13	Uniform Cost Search	TRUE	0.07 secs	9	9
14	Uniform Cost Search + GS	TRUE	0.06 secs	9	9



	name	solution	runtime	actions	cost
1	Breadth First Search	TRUE	0.30 secs	11	11
2	Breadth First Search + GS	TRUE	0.03 secs	11	11
3	Depth First Search	TRUE	0.03 secs	11	11
4	Depth First Search + GS	TRUE	0.04 secs	11	11
5	Depth First Search - Limit=6	TRUE	0.04 secs	11	11
6	Depth First Search - Limit=6 + GS	TRUE	0.03 secs	11	11
7	Depth First Search - Limit=10	TRUE	0.03 secs	11	11
8	Depth First Search - Limit=10 + GS	TRUE	0.02 secs	11	11
9	Iterative Deeping Search - It:3	TRUE	0.14 secs	11	11
10	Iterative Deeping Search - It:3 + GS	TRUE	0.13 secs	11	11
11	Greedy Best-First Search	TRUE	0.02 secs	11	11
12	Greedy Best-First Search + GS	TRUE	0.03 secs	11	11
13	Uniform Cost Search	TRUE	0.04 secs	11	11
14	Uniform Cost Search + GS	TRUE	0.04 secs	11	11



En este primer mapa todos los algoritmos encuentran solución al problema.

Datos / **Algoritmos**	**Completo**	**Óptimo**	**Complejidad temporal**	**Complejidad espacial**	**Tiempo**	**Movimientos**
Breadth First Search	Es completo, ya que el factor de ramificación es finito (4).	Es óptimo porque el coste de las acciones es siempre el mismo.	Complejidad exponencial. Al ser pocas casillas no se generan muchos estados. O(b^d+1)	Complejidad exponencial. Se guardan todos los nodos en memoria. O(b^d+1)	Es el que más tiempo tarda en encontrar solución debido a la cantidad de nodos que se generan.	Mapa 1 → 9 Mapa 2 → 11
Depth First Search	En este problema se podría decir que es completo, ya que no existe la posibilidad de entrar en un bucle.	En este problema no es óptimo, ya que el número de acciones que acaba encontrando se aproxima al número de celdas.	Complejidad exponencial en el peor caso. O(b^m)	Complejidad lineal, solo mantiene los nodos de la ruta actual. O(bm)	Tarda más que los algoritmos DFS con límite porque en este caso el límite es menor o igual a las celdas recorribles del mapa.	Mapa 1 → 15 Mapa 2 → 11

Depth First Search, límite = 6	En este problema sí que es completo, ya que la distancia del objetivo es menor al límite.	En este problema es óptimo, ya que la distancia del objetivo es menor al límite.	Complejidad exponencial en el peor caso. O(b^L)	Complejidad lineal. O(bL)	Este algoritmo tarda menos que los DFS sin límite porque tiene menos casillas.	Mapa 1 → 15 Mapa 2 → 11.
Depth First Search, límite = 10	En este problema sí que es completo, ya que la distancia es menor al límite.	Es óptimo, ya que la distancia del objetivo es menor al límite.	Complejidad exponencial en el peor caso. O(b^L)	Complejidad lineal. O(bL)	Este algoritmo tarda menos que los DFS sin límite porque tiene menos casillas.	Mapa 1 → 15 Mapa 2 → 11
Iterative Deeping Search	Es completo, ya que la profundidad de búsqueda es un número finito.	Es óptimo, ya que todas las acciones tienen el mismo coste.	Complejidad exponencial superior al DFS, ya que implica iterar sobre varias profundidades. O(b^d)	Complejidad lineal, solo mantiene los nodos de la ruta actual. O(bd)	Este algoritmo tarda más que los diferentes DFS debido al tiempo que implica iterar las diferentes profundidades.	Mapa 1 → 9 Mapa 2 → 11
Greedy BFS	No es completo, ya que solo se expande el nodo que produce un menor valor heurístico.	No es óptimo, aunque se enfoca en una búsqueda óptima de la solución.	Complejidad exponencial en el peor de los casos. O(b^m)	Complejidad exponencial en el peor caso, mantiene todos los nodos en memoria. O(b^m)	Es el algoritmo más rápido de todos en su versión TS.	Mapa 1 → 9 Mapa 2 → 11
Uniform CS	Es completo ya que los costes negativos en este problema no existen.	Es óptimo porque no existen en este problema costes negativos.	Complejidad exponencial. O($b^{\frac{C*}{\varepsilon}}$)	Complejidad exponencial, mantiene todos los nodos generados en memoria. O($b^{\frac{C*}{\varepsilon}}$)	Este algoritmo ha tenido un tiempo de resolución similar o superior a los algoritmos DFS.	Mapa 1 → 9 Mapa 2 → 11

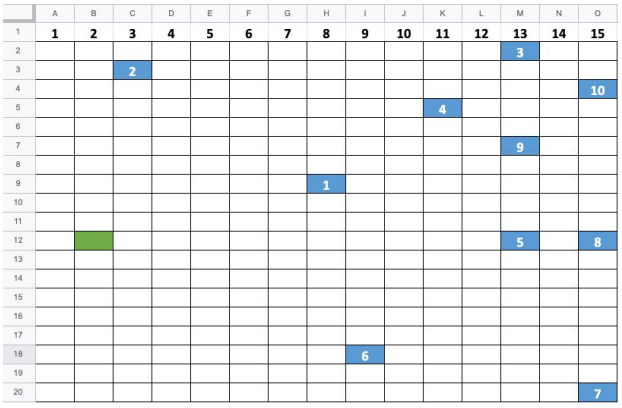
Todos los algoritmos que usan “Graph Search” tardan lo mismo o menos que los “Tree Search”, menos en el Greedy BFS que tarda más.

Estos algoritmos quedarían ordenados de la siguiente manera teniendo en cuenta su complejidad temporal (orden de complejidad descendente):

1. BFS
2. IDS
3. UCS
4. DFS
5. GBFS
6. DFS + límite = 10
7. DFS + límite = 6

En este caso, podemos comprobar que el orden expuesto se asemeja a los resultados obtenidos en cuanto al tiempo de resolución de cada algoritmo y mapa.
Los algoritmos de complejidad espacial lineal deberían de ser menos complejos que los exponenciales, observando las fórmulas creemos que el siguiente orden sería el adecuado de mayor a menor complejidad, pero realmente no podemos comprobarlo debido a que al implementar un enfoque divido del problema algunas características de la ejecución de los algoritmos no las hemos podido conservar:

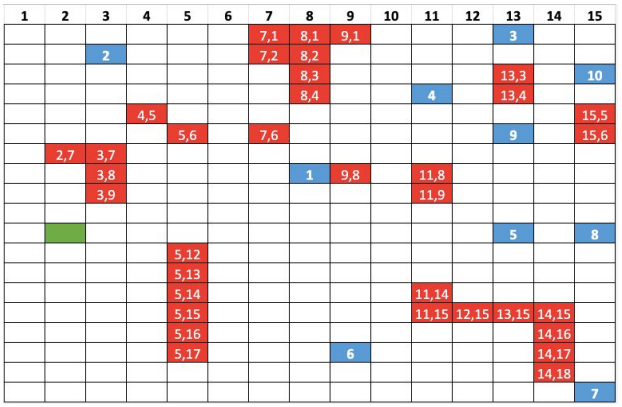
1. BFS
2. UCS
3. GBFS
4. IDS
5. DFS
6. DFS + límite = 10
7. DFS + límite = 6



Mapas 3 y 4

	name	solution	runtime	actions	cost
1	Iterative Deeping Search - It:9	TRUE	118.96 secs	85	85
2	Iterative Deeping Search - It:9 + GS	TRUE	461.25 secs	85	85
3	Greedy Best-First Search	TRUE	0.19 secs	85	85
4	Greedy Best-First Search + GS	TRUE	0.21 secs	85	85

	name	solution	runtime	actions	cost
1	Greedy Best-First Search	TRUE	791.73 secs	109	109
2	Greedy Best-First Search + GS	TRUE	10152.14 secs	109	109



Estas instancias del problema solo pueden ser resultas en un tiempo razonable mediante los algoritmos que aparece en la parte superior, el resto de algoritmos puede que encuentren solución en una gran cantidad de tiempo o lleguen a bloquearse. Se puede observar como la complejidad temporal y espacial se eleva exponencialmente al ejecutar los algoritmos en mapas un poco más complejos.

En el 3º mapa, el Iterative Deeping Search y el Greedy BFS son los únicos algoritmos que encuentran solución en una cantidad de tiempo razonable. Ambos algoritmos realizan 85 movimientos para encontrar la resolución. Cabe destacar que existe una gran diferencia temporal de resolución entre los dos algoritmos.

En el 4º mapa, el algoritmo Greedy Best-First Search es el único que encuentra solución en un tiempo razonable. Se resuelve en 109 movimientos en ambos algoritmos.

En ambos mapas, los algoritmos Iterative BFS GS y Greedy BFS GS, tardan mucho más que los Tree Search.