Examen
Grado en Robótica. Curso 2021-2022.
Departamento de Electrónica y Computación

Nombre:.....

1. Problema 1 (5 puntos)

Una empresa de paquetería gestiona la entrega de paquetes en varias ciudades. Todas las mañanas llegan a cada ciudad los paquetes que deben distribuirse, y para los que se conoce la dirección de entrega. Las entregas se realizan con furgonetas que recogen en el almacén de cada ciudad un paquete, y sólo uno, y lo entregan en la dirección de destino. Todas las furgonetas son iguales en capacidad y rendimiento, y puede asumirse que hay siempre más furgonetas que paquetes a distribuir. Además, puede asumirse que nunca habrá colisiones entre diferentes furgonetas en la misma ciudad, para el cálculo de cada ruta. Por último, *ninguna furgoneta puede dirigirse a una ciudad diferente que aquella en la que recogió un paquete*.

La empresa está ahora interesada en minimizar la distancia recorrida total en todas las ciudades por todas las furgonetas, desde cada almacén hasta cada punto de entrega.

Se pide responder razonadamente las siguientes preguntas:

1. (**2 punto**) Representar el problema definiendo los estados y las acciones con sus precondiciones, efectos y coste.
2. (**2 punto**) Sugiere una función heurística h(·) que sea admisible e informada.
3. (**1/2 punto**) ¿Qué algoritmo de búsqueda heurística es el más indicado para resolver óptimamente el problema? ¿Por qué?
4. (**1/2 punto**) ¿Sería posible resolver óptimamente el problema con un algoritmo de búsqueda *no informada*? Razona tu respuesta.

2. Problema 2 (2 puntos)

Recorrer el grafo de la Figura 1 utilizando el algoritmo A* teniendo en cuenta los costes indicados en cada arco. En todos los casos, indicar en qué orden se visitan los nodos, distinguiendo nodos generados de nodos expandidos. Para cada nodo indicar, además, su valor correspondiente a la función de evaluación, la función de coste y su valor heurístico. Tomar como estado inicial el nodo S y como único estado meta el nodo G. Cada nodo del grafo tiene el valor heurístico descrito en la Figura 2.

3. Problema 3 (3 puntos)

Un robot con dos *grippers* es capaz de transportar bolas desde una habitación a otra dentro de una casa. El robot inicialmente se encuentra en una habitación, y recibe órdenes para transportar bolas (que pueden estar en la misma habitación que el robot o no) a otras habitaciones.

Dado este dominio, se pide resolverlo utilizando *planificación automática*. Para ello:

1. (1 punto) Empleando la **lógica de predicados**, indicar qué predicados sirven para describir cada estado.
2. (1 punto) Pon un ejemplo de **estado inicial** y **final** con dos robots y cuatro bolas.

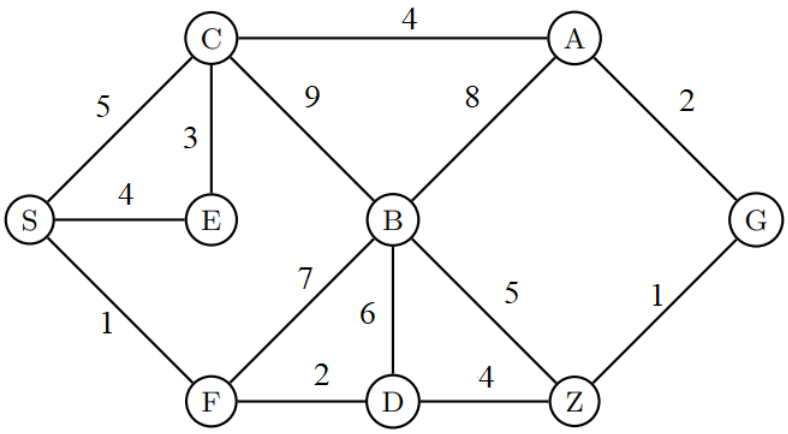

Figura 1: Representación del grafo de búsqueda.

n	S	A	B	C	D	E	F	G	Z
h(n)	7	2	5	6	3	7	7	0	1

Figura 2: Valores heurísticos.

3. (1 punto) Describir brevemente e **informalmente** los operadores que utilizarías para solucionar el problema y modelar uno de ellos **formalmente** en PDDL.

4. Solución Problema 1

1. El espacio de estados es una formalización que habilita la aplicación de algoritmos de búsqueda (informados o no) para la resolución de problemas. Consiste únicamente, en la definición de estados y operadores que sirven para transitar entre ellos. Relacionando, después, el conjunto de posibles estados con otro de vértices V y el de operadores (convenientemente instanciados) con otro de arcos E, resulta entonces de forma natural la definición de un grafo, el *grafo de búsqueda* que se recorrerá eficientemente con el uso de *árboles de búsqueda*. Este ejercicio propone ejercitar todos estos conceptos y, en el primer apartado, el del espacio de estados.

   - **Estados** Un estado en este problema está caracterizado por información de cada ciudad, la lista de entregas que deben realizarse en cada ciudad y, por último, el estado de todas las furgonetas disponibles para los diferentes servicios. De hecho, cada entrega estará cualificada con información de la furgoneta que hace el servicio correspondiente:
     - **Entregas** Cada entrega está identificada con un código de la ciudad a la que llega y donde debe servirse `(ciudad)`. Además, contendrá información de la dirección de entrega `(direccion)` y un código único que sirva para distinguirla de otras entregas, `id`. Para poder gestionar la actividad diaria de la empresa de paquetería considerada en el enunciado, cada entrega debe asignarse a un vehículo `(furgoneta)` del que se mantendrá información adicional como se explica a continuación.
     - **Furgonetas** Cada furgoneta está identificada con un código único que la distingue de las demás `(furgoneta)`. Además, contendrá información de su posición actual distinguida con dos coordenadas que la localizarán perfectamente sobre el mapa de la ciudad `(xpos, ypos)` y el identificador del paquete que transporta, `paquete`. Por último, para habilitar la aplicación de algoritmos de búsqueda, debe contener un campo de distancia recorrida `(distancia)` que se actualizará en la expansión de cada nodo (con el operador `Mover` descrito más abajo)
     - **Mapa** Para poder determinar las rutas óptimas para cada furgoneta, el sistema debe conocer la topografía de cada ciudad que se representará con un mapa único con información de todas las ciudades consideradas por la empresa de paquetería. Cada mapa consiste simplemente en un par de coordenadas `(xpos, ypos)` que apuntan a los pares de coordenadas adyacentes a él.

     Si una furgoneta no tiene ningún paquete asignado, su identificador de paquete `(paquete)` será nulo. Análogamente, si una entrega no tiene asignado aún un vehículo, su referencia `furgoneta` será también nulo.
   - **Operadores** En la definición de operadores es importante describir sus precondiciones/postcondiciones y, además, el coste que tienen. En este problema pueden distinguirse dos operadores: el primero consiste en asignar furgonetas a servicios a primera hora de la mañana; el segundo consiste en determinar los movimientos de la furgoneta en cada ciudad de modo que haga el recorrido óptimo hasta su entrega.
     - **Asignar** Puesto que el enunciado advierte que hay una cantidad arbitrariamente grande de furgonetas, la asignación se hará una única vez por la mañana, de modo que la única precondición que debe verificarse es que se trate de una furgoneta disponible y una entrega sin furgoneta asignada. La postcondición es, simplemente, que el paquete asignado se escribe en el atributo `paquete` de la furgoneta escogida.
       Puesto que el objetivo del problema es minimizar la distancia total recorrida por todas las furgonetas en todas las ciudades, esta operación no afecta a ese coste (puede asumirse libremente que las furgonetas están todas estacionadas por la mañana junto al almacén, de modo que no hay que hacer ningún recorrido para recoger un paquete) y, por lo tanto, tendrá coste 0.
     - **Mover** El movimiento de cada furgoneta se decide en base a la información disponible en el mapa. Para cualquier posición determinada por el par `(xpos, ypos)` es posible moverse a cualquier posición apuntada por élla en el mapa, o sea, que sea adyacente a élla. Es una precondición de este operador que tenga asignado un paquete mediante la ejecución del operador

`Mover` (esto es, ¡no queremos que el algoritmo mueva vehículos que no transportan nada!). La postcondición de este operador consiste en sobreescribir el contenido de sus atributos `xpos` e `ypos` con la nueva posición, y actualizar la información de la distancia recorrida por esta furgoneta.
El coste del operador es igual a la distancia recorrida con su aplicación.

2. Para el diseño de una función heurística *admisible* y bien informada, se sugiere el uso de la técnica de *relajación de restricciones*. Habida cuenta de que cada algoritmo de búsqueda sólo resolverá el *enrutamiento* de cada furgoneta por separado, se propone una relajación que afecta a una única furgoneta.

   En el proceso de encontrar el camino óptimo entre dos puntos n y t, respectivamente, se sugiere relajar la condición (enunciada en el primer apartado) de que el movimiento de las furgonetas se hace necesariamente a posiciones adyacentes. El *problema relajado* resultante puede resolverse óptimamente simplemente asumiendo que el movimiento de cada furgoneta se hace a lo largo de una ruta mínima incluso si los puntos intermedios no son adyacentes. La ruta óptima del problema relajado es precisamente una línea recta que une a la furgoneta con su posición de destino, t, a partir de su posición original, n. Por lo tanto, la función heurística propuesta es precisamente la distancia *aérea* entre la posición actual de la furgoneta y la posición de destino:

$$h(n) = \sqrt{(n.xpos - t.xpos)^2 + (n.ypos - t.ypos)^2} \tag{1}$$

   donde, la solución óptima global resulta de resolver la aplicación de diferentes algoritmos de búsqueda heurística , cada uno de ellos guiado por la función indicada aquí.

3. La selección de un algoritmo de búsqueda informada para este caso depende, como en el caso de los algoritmos de búsqueda no informada de la dificultad de los problemas:

   - El algoritmo A* es admisible y garantiza, por lo tanto, que encontrará soluciones óptimas si la función heurística que lo guía también es admisible. Además, es un algoritmo rápido puesto que no reexpande nodos (y, con frecuencia, las ordenaciones de la lista abierta se pueden hacer en $O(1)$ con las estructuras de datos adecuadas si la función objetivo sólo toma valores enteros). Sin embargo, tiene un consumo de memoria exponencial.
   - El algoritmo IDA* podría reexpandir nodos pero no ordena nodos y, mucho más importante aún, tiene un consumo de memoria lineal en la profundidad de la solución. Además, también es un algoritmo de búsqueda admisible.

   Tal y como se indicó en el análisis de los algoritmos no informados, si los algoritmos del primero en amplitud agotaban allí toda la memoria disponible para problemas pequeños, en el caso de los algoritmos de búsqueda heurística ocurrirá lo mismo con los algoritmos del mejor primero como A*. Por lo tanto, se desestima el uso del algoritmo A* y, en su lugar, se propone el segundo, IDA*.

4. Hay que tener en cuenta que se trata de resolver un problema de *minimización* en un grafo de estados donde los operadores tienen *costes arbitrarios*. Por lo tanto, algoritmos como Amplitud no garantizan encontrar la solución óptima en el caso de que no tengamos *costes unitarios*. El algoritmo de búsqueda no informada que permite costes arbitrarios estudiado con este propósito es fundamentalmente *Dijkstra*, que consiste en un algoritmo de el mejor primero donde la función de evaluación, *f(n)* es, simplemente, el coste del camino desde el estado inicial hasta n, *g(n)*. Tiene un coste de memoria exponencial pero garantiza que cada vez que expande un nodo habrá encontrado la solución óptima hasta él puesto que los nodos se expanden en orden creciente de su valor de *f(n)* —o, equivalentemente, de *g(n)*.

5. Solución problema 2

Para resolver este problema, se muestra en la siguiente tabla la evolución de las listas ABIERTA y CERRADA. Al lado de cada nodo, se muestra entre paréntesis el coste de llegar a ese nodo, su valor heurístico,

recuperado de la tabla que nos proporcionan, y la función de evaluación. Por claridad, estos valores se proporcionan únicamente cuando los nodos se encuentran en ABIERTA. Una vez pasan a CERRADA, solo se proporciona su nombre. Además, se puede comprobar que ABIERTA se mantiene ordenada en todo momento usando la función de evaluación.

ABIERTA	CERRADA
S(0;7;**7**)	∅
F(1;7;**8**) C(5;6;**11**) E(4;7;**11**)	S
D(3;3;**6**) C(5;6;**11**) E(4;7;**11**) B(8;5;**13**)	S F
Z(7;1;**8**) C(5;6;**11**) E(4;7;**11**) B(8;5;**13**)	S F D
G(8;0;**8**) C(5;6;**11**) E(4;7;**11**) B(8;5;**13**)	S F D Z

Al inicio, ABIERTA solo contiene S, y CERRADA se encuentra vacía. En la primera iteración, por tanto, se expande S y se generan en orden los nodos F, C, E. Los nodos C, E empatan en su función de evaluación, pero podemos resolver los empates por el nodo que está más a la izquierda en el árbol de búsqueda. Si suponemos que los nodos se generan alfabéticamente, podemos suponer que el nodo más a la izquierda es el C, y por lo tanto, el orden sería C, E, quedando en ese momento ABIERTA como F, C, E. En la siguiente iteración se expande el nodo F, lo que genera dos nuevos nodos: el D y el B, que se insertan en orden en ABIERTA. Se puede comprobar que la función $g(n)$ asociada al nodo D **NO** es 2. 2 es el coste de ir de F a D, pero la $g(n)$ **es el coste acumulado desde el estado inicial al estado** n **en el que nos encontramos**, por lo tanto su valor es de 3, que es la suma del coste de ir de S a F (coste 1) más el coste de ir de F a D (coste 2). Dicho de otra manera, el coste $g(n)$ de un nodo n, es la suma del coste g del padre de n, más el coste de realizar el movimiento del padre al nodo n.

En la última iteración, se puede ver que la ordenación de ABIERTA es: G, C, E, B, siendo el primer nodo el nodo G que a su vez es el nodo meta. Como quiera que A* **no para cuando se genera el nodo meta, sino cuando se procede a su expansión**, siendo este el caso, podemos decir que el camino óptimo para llegar del nodo S al G es: S, F, D, Z, G.

6. Solución problema 3

1. Como ocurre en todos los problemas que nos piden representar los conceptos que aparecen en el enunciado, no tiene por qué existir una única representación válida. En este caso se pide representar el problema utilizando lógica de predicados, y un conjunto válido de predicados podría ser:

   - `(at-robot ?robot ?room)`: Un determinado robot `?robot` está en la habitación `?room`. Este predicado es necesario para identificar la localización del robot dentro de la casa. Aunque el enunciado habla de un único robot, este predicado, sin embargo, nos permite generalizar para tener en cuenta más de un robot como veremos en el apartado 2.
   - `(at-ball ?ball ?room)`: Equivalente al predicado anterior pero para las bolas. Describe que una determinada bola `?ball` está en la habitación `?room`. Conviene notar que, si pensamos en PDDL, este predicado y el anterior podrían unificarse en uno solo, el predicado `(at ?object ?room)` donde `?object` podría ser un robot o una bola. De esta última forma podríamos generalizar nuestro dominio aún más. De hecho, por ejemplo, no tendríamos por qué modificarlo si más adelante aparecen otros objetos diferentes de las bolas que el robot puede transportar. Sin embargo, por simplicidad, mantendremos estos dos predicados: uno para identificar la posición del robot, y otro para identificar la posición de las bolas.
   - `(free ?robot ?gripper)`: El *gripper* `?gripper` de un determinado robot `?robot` está libre y puede utilizarse para transportar algún objeto, en este caso, una bola. El predicado habla de que el robot tiene solamente dos *grippers*, pero con este predicado, cuando definamos el problema, podríamos hacer que el robot tuviese dos o un número arbitrario de *grippers*.
   - `(carry ?robot ?ball ?gripper)`: El robot `?robot` está transportando una bola `?ball` con su *gripper* `?gripper`.

Hasta aquí tenemos todos los predicados necesarios para modelar el problema.

2. En este caso nos piden un estado inicial y uno final para dos robots y cuatro bolas. En cuando al estado inicial, podríamos modelarlo de la siguiente manera:

```
(at-robot robot1 room1)
(free robot1 rgripper1)
(free robot1 lgripper1)
(at-robot robot2 room2)
(free robot2 rgripper2)
(free robot2 lgripper2)
(at-ball ball1 room1)
(at-ball ball2 room2)
(at-ball ball3 room3)
(at-ball ball4 room4)
```

   Con este estado inicial, estamos indicando que tenemos dos robots, `robot1` y `robot2`, en las habitaciones `room1` y `room2` respectivamente, y que, además, tenemos cuatro bolas, `ball1`, `ball2`, `ball3`, `ball4`, en las habitaciones, `room1`, `room2`, `room3` y `room4`. Además, cada robot está equipado con dos *grippers* que inicialmente están libres.

   En cuanto a un posible estado final:

```
(at-robot robot1 room1)
(free robot1 rgripper1)
(free robot1 lgripper1)
(at-robot robot2 room2)
(free robot2 rgripper2)
(free robot2 lgripper2)
(at-ball ball1 room2)
(at-ball ball2 room3)
(at-ball ball3 room2)
(at-ball ball4 room1)
```

   donde establecemos que los robots deben acabar en la misma habitación en la que empezaron y además deben hacerlo con los *grippers* libres, y se ha cambiado de habitación a las bolas.

3. En cuanto a los operadores, con la representación utilizada, tendríamos tres operadores diferentes:

   - `(move ?robot ?from ?to)`: Para indicar que el robot se mueve de una determinada habitación de partida `?from` a una habitación de destino `?to`. Para ello, el robot se debe encontrar en la habitación de partida y, una vez acabada la acción, el robot dejará de estar en la habitación de partida y pasará a la de destino.
   - `(pick ?robot ?ball ?room ?gripper)`: El robot `?robot` coge la bola `?ball` de la habitación `?room` utilizando el *gripper* `?gripper`. Las precondiciones serían que el robot tiene el *gripper* `?gripper` libre, y que se encuentra en la misma habitación `?room` que la bola `?ball` que quiere coger. Los efectos serían que el robot pasa a estar `(carry ?robot ?ball ?gripper)`, que el *gripper* deja de estar libre, y que la bola deja de estar en la habitación.
   - `(drop ?robot ?ball ?room ?gripper)`: Equivalente al operador anterior, solo que en este caso en lugar de recoger la bola de la habitación, el robot la suelta. Las precondiciones serían que el robot `?robot` efectivamente está llevando la bola `?ball` con su *gripper* `?gripper` (lo que modelamos con el predicado `(carry ?robot ?ball ?gripper)`), y que el robot está en la habitación `?room`. Los efectos serían que la bola `?ball` pasa a estar en la habitación `?room`, el robot deja de transportar la bola, y el *gripper* correspondiente pasa a estar libre.

   Por último, como el enunciado nos pide además modelizar uno de ellos de manera formal mediante sintaxis PDDL:

```
(:action drop
       :parameters (?r - robot ?obj - ball ?room - room ?g - gripper)
       :precondition  (and  (carry ?r ?obj ?g) (at-robot ?r ?room))
       :effect (and (at ?obj ?room)
            (free ?r ?g)
            (not (carry ?r ?obj ?g)))))
```