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

Nombre:.....

1. Problema 1 (5 puntos)

Un *JukeBox* consiste en una pila de n discos de vinilo equipado con un brazo electrónico que puede extraer un disco de cualquier posición, entre la segunda y la última, para ponerlo en la primera. Dada una disposición inicial de los n discos, se está considerando la posibilidad de usar algoritmos de búsqueda para minimizar el número de movimientos del brazo de robot que disponen todos los discos en un orden específico.

Se pide responder razonadamente las siguientes preguntas:

1. (**1 punto**) Representar el problema definiendo los estados y las acciones con sus precondiciones, efectos y coste.
2. (**1/2 puntos**) ¿Cuál es el tamaño del espacio de estados?
3. (**1/2 puntos**) ¿Cuál es el factor de ramificación mínimo desarrollado por cualquier algoritmo de búsqueda no informada? ¿Y el máximo?
4. (**1 punto**) Si se desea calcular la solución óptima, ¿qué algoritmo de búsqueda no informada sugerirías para su resolución? ¿Por qué?
5. (**1 punto**) Sugiere una función heurística h(·) que sea admisible e informada para el problema de minimizar el número de movimientos del brazo de robot necesarios para disponer los n discos en el orden especificado.
6. (**1 punto**) ¿Qué algoritmo de búsqueda heurística es el más indicado para resolver óptimamente el problema? ¿Por qué?

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 *Barman* se encarga de servir bebidas a los clientes de un bar. Dentro del bar, el robot se puede mover entre tres localizaciones diferentes: la barra, la sección de vasos, y la sección de dispensadores. La sección de dispensadores está compuesta por varios dispensadores, y cada uno sirve una bebida diferente. El robot inicialmente se encuentra en la barra para anotar las consumiciones que quieren los clientes. Una vez anotadas, el robot debe dirigirse primero a la sección de vasos, coger los vasos necesarios (uno por petición), ir a la sección de dispensadores, rellenar los vasos con las bebidas indicadas de los dispensadores adecuados y,

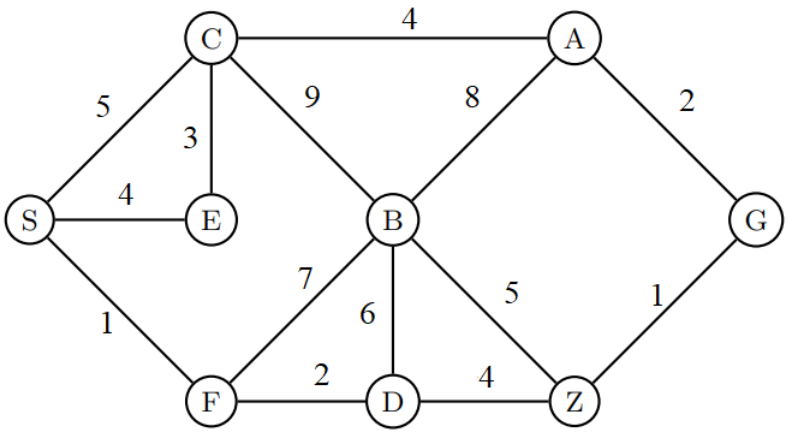

Figura 1: Representación del grafo de búsqueda.

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.

una vez llenos, servir las bebidas en la barra. El robot solo es capaz de transportar dos vasos como máximo a la vez, luego si hay más de dos peticiones, el robot primero deberá atender las dos primeras, después las dos siguientes y así sucesivamente.

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 clientes que solicitan cada uno una bebida diferente.
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**: En este problema un estado se representa simplemente con una ordenación particular de discos. A continuación se presentan definiciones estructuradas para cada uno de estos conceptos:
     - **JukeBox**: Cada configuración del JukeBox, u ordenación de discos, está caracterizada por una permutación `perm` de n posiciones, donde `perm[i]` representa el disco que ocupa la posición i-ésima. Esta representación asume, naturalmente, que cada disco está identificado de forma única, por ejemplo, con un número entero entre 0 y $(n-1)$.

     Por lo tanto, cualquier ordenación de discos es un estado diferente. En particular, el estado inicial se corresponde con la ordenación actual, y el estado final será el de la ordenación deseada.
   - **Acciones**: En este problema hay una única acción, `mover`, que toma como parámetros una posición entre 1 y $(n-1)$ i, y mueve el disco en la posición i-ésima a la primera, denotada como la posición 0.

     Básicamente, este operador no tiene precondiciones, de hecho, es posible mover cualquier disco en cualquier posición a la primera. Únicamente, para evitar redundancias y, también, para preparar el cálculo del factor de ramificación que se hará más tarde, se advierte que la única precondición es que el disco a mover debe estar entre la segunda y n-ésima posición, tal y como advertía el enunciado.

     Las postcondiciones describen el efecto de aplicar el operador que, en este caso, consiste en desplazar todas las posiciones del vector `perm` de las posiciones $0 : i-1$ a las posiciones $1 : i$, e insertar el contenido anterior de `perm[i]` en la posición 0.

   Por último, es importante determinar el coste del único operador identificado. Para ello, se observa que el enunciado pide explícitamente minimizar el número de movimientos del brazo de robot. Por lo tanto, independientemente de la posición del disco que se mueva, cada movimiento cuenta como una unidad y, por lo tanto, este será el coste del operador `mover`.

   También se podría suponer que el coste depende de la posición del disco, asignando más coste a aquellos movimientos que requieren más *recorrido* del brazo robótico. Esta solución también es válida si se justifica correctamente puesto que colateralmente va a permitir minimizar el número de movimientos. Sin embargo, este hecho no se cumpliría para configuraciones a las que podamos llegar a través de varios movimientos pero de menor coste que si simplemente hacemos un único movimiento de coste mayor.

2. Tal y como se indicaba en el apartado anterior, cada ordenación de discos diferente es un estado distinto. Como quiera que con n discos hay hasta $n!$ ordenaciones diferentes, este es precisamente el tamaño del espacio de estados.

3. El primer apartado advertía que, en preparación al cálculo del factor de ramificación, la única precondición del operador `mover` es que $1 \leq i < n$. Por lo tanto, en cada paso, es posible mover hasta $(n-1)$ discos y, de hecho, no se pueden mover más de $(n-1)$ discos. Por lo tanto, este es el factor de ramificación mínimo y máximo.

4. Con un factor de ramificación tan alto como $(n-1)$, y un espacio de estados de tamaño factorial, los algoritmos del primero en amplitud son impracticables puesto que consumirían la capacidad del ordenador más potente con valores muy pequeños de n, $n < 20$. Para enfatizar esta observación, se observa que

si la caracterización de cada estado consumiese sólo 1 byte, harían falta $2,265,816,562$ Gigabytes para almacenar todos los estados que se pueden engendrar con $n = 20$.

Por lo tanto, para resolver el problema óptimamente una alternativa sería el algoritmo del primero en profundización iterativa. Para ello, inicialmente se invoca el algoritmo del primero en profundidad con un umbral igual a la unidad, de modo que se enumeran todas las permutaciones u ordenaciones de discos que se pueden hacer con un solo movimiento. Si algún nodo terminal generado de esta manera replica la ordenación deseada, entonces el algoritmo acaba con la solución óptima. En otro caso, se incrementa el umbral de la siguiente iteración en otra unidad. Necesariamente el algoritmo debe terminar en un número finito de iteraciones puesto que cualquier ordenación es alcanzable desde cualquier otra con el operador mover propuesto en la formalización del espacio de estados. La ventaja de este algoritmo es que tiene un consumo de memoria lineal. Sin embargo, debe re-expandir todas las *transposiciones*.

5. Una heurística muy sencilla que puede obtenerse con la técnica de relajación de restricciones, consiste en relajar la restriccion de que el disco que se extrae de la pila debe colocarse el primero. En su lugar, se supone que puede colocarse en cualquier posición.

   Esto no significa, sin embargo, que el número de discos mal dispuestos sea una heurística admisible. De hecho, el número de discos mal dispuestos puede exceder el coste de resolver óptimamente una configuración determinada. Considérese, por ejemplo, la siguiente permutación de discos $\langle 0, 1, 4, 2, 3\rangle$ donde los tres últimos están mal dispuestos [1]. Sin embargo, con una sola aplicación del operador relajado (mover el disco 4 a la última posición), es posible ordenarlos todos resultando $\langle 0, 1, 2, 3, 4\rangle$. Asimismo, considérese ahora la permutación $\langle 1, 2, 3, 4, 0\rangle$, donde hay hasta 4 discos mal dispuestos, pero este problema puede resolverse óptimamente con una sola aplicación del operador real, esto es, $h* = 1$.

   Por lo tanto, una heurística sencilla se puede definir como el número de veces que se debe aplicar el operador relajado para obtener la configuración final deseada. Por ejemplo, en el caso de la permutación $\langle 0, 1, 4, 2, 3\rangle$, hay una única inversión, el disco 4, que debe moverse al final de la permutación y, por lo tanto, la heurística devolvería 1 —en vez de 3, el número de discos mal dispuestos. Considérese ahora el estado $\langle 1, 2, 3, 4, 0\rangle$. Se puede comprobar fácilmente que nuestra heurística devolvería 1, que coincide, de hecho, con el coste real.

6. La selección de un algoritmo de búsqueda informada para este caso depende, como en el caso de los algoritmos de búsqueda no informada, del número de transposiciones y también, naturalmente, 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* reexpande nodos en caso de que haya transpociones pero no ordena nodos y, mucho más importante aún, tiene un consumo de memoria lineal en la profundidad de la solución (que en nuestro caso es siempre igual a N). 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 valores pequeños de n, en el caso de los algoritmos de búsqueda heurística ocurrirá lo mismo con los algoritmos del mejor primero. Por lo tanto, se desestima el uso del algoritmo A* y, en su lugar, se propone el segundo, IDA*.

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,

[1] En todos los ejemplos de esta pregunta se asume que la ordenación deseada es $\langle 0, 1, 2, 3, 4\rangle$

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**)	$\emptyset$
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 ?wp)`: Un determinado robot `?robot` está en la posición `?wp`. Este predicado es necesario puesto que dependiendo de donde se encuentre el robot podrá hacer unas cosas u otras. Además, el enunciado nos dice explícitamente que el robot puede estar en tres localizaciones diferentes: la barra, la sección de vasos, y la sección de dispensadores. Para este predicado se ha decidido utilizar un parámetro `?robot` que realmente no sería necesario puesto que el enunciado nos habla de un único robot. Este predicado, sin embargo, nos permite generalizar el enunciado para tener en cuenta más de un robot.
   - `(at-vaso ?vaso ?wp)`: Además de robots, el enunciado nos habla de otros dos conceptos: vasos, y dispensadores. Este predicado sirve para indicar que un determinado vaso `?vaso` está en una determinada localización `?wp`.
   - `(at-dispensador ?dispensador ?wp)`: Por último, este predicado sirve para indicar que un determinado dispensador `?dispensador` está en una determinada localización `?wp`.
   - `(es-barra ?wp)`: Sirve para indicar que una determinada localización `?wp` se corresponde con la barra.
   - `(es-seccion-vasos ?wp)`: Análogo al anterior, sirve para indicar que una determinada localización `?wp` se corresponde con la sección de vasos.
   - `(es-seccion-dispensadores ?wp)`: Para indicar que una determinada localización `?wp` se corresponde con la sección de dispensadores.

- `(sirve ?dispensador ?bebida)`: Para indicar que un dispensador `?dispensador` sirve una determinada bebida `?bebida`.
- `(lleno ?vaso ?bebida)`: Sirve para indicar que un vaso `?vaso` está lleno con la bebida `?bebida`. Podriamos tener otro predicado que sirviese para indicar que un vaso está vacío, pero podemos obtener esa misma semántica negando este predicado.
- `(peticion ?cliente ?bebida)`: Modela que un cliente `?cliente` ha solicitado una determinada bebida `?bebida`.
- `(servido ?cliente ?bebida)`: Para indicar que el cliente `?cliente` ya ha sido servido con la bebida `?bebida` que había solicitado.

Hasta aquí tenemos todos los predicados necesarios para modelar gran parte del problema. Ahora bien, el enunciado también nos dice que: *El robot solo es capaz de transportar dos vasos como máximo*. Este hecho también se puede modelar también a través de predicados. En particular:

- `(vacio ?deposito)`: Sirve para indicar que un deposito `?deposito` está vacio y puede ser ocupado por un vaso. Conviene notar que con esta representación no estamos limitando que el número de depósitos sea dos como nos indica el enunciado. Sin embargo, posteriormente, en la declaración del problema, podríamos indicar que el robot tiene únicamente dos depósitos: `deposito1`, `deposito2`. De hecho, podríamos indicarle que tiene cualquier número de depósitos obteniendo de esta forma una representación más general.
- `(lleno-vaso ?deposito ?vaso)`: Sirve para indicar que el deposito `?deposito` del robot está ocupado por el vaso `?vaso`.

Con estos dos predicados controlamos la situación del depósito del robot en todo momento, puesto que puede estar vacio, o con un vaso.

Por último, en cuanto a los predicados `at-robot`, `at-vaso` y `at-dispensador`, si pensamos en PDDL, podrían haber sido unificados en un solo predicado general `at`, es decir, podríamos haber creado un predicado `(at ?object ?wp)` para indicar que un determinado objeto `?object`, ya sea robot, vaso o dispensador, se encuentran en la localización `?wp`.

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

```
(es-barra barra)
(es-seccion-vasos seccion_vasos)
(es-seccion-dispensadores seccion_disp)
(at-robot kenny barra)
(at-vaso vaso1 seccion_vasos)
(at-vaso vaso2 seccion_vasos)
(at-dispensador disp1 seccion_disp)
(at-dispensador disp2 seccion_disp)
(sirve disp1 fanta)
(sirve disp2 cerveza)
(peticion cl1 cerveza)
(peticion cl2 cerveza)
(vacio dep1)
(vacio dep2)
```

Con este inicial, estamos indicando que tenemos tres zonas en el bar: la barra, la sección de vasos y la sección de dispensadores; que nuestro robot `kenny` se encuentra en la barra, que tenemos dos vasos en la sección de vasos, y dos dispensadores en la sección de dispensadores, uno sirve *fanta* y otro *cerveza*, que tenemos dos clientes que piden cerveza y el robot dispone de dos depósitos para transportar los vasos.

En cuanto al estado final, sería igual al anterior, puesto que el robot debe acabar en la barra donde servirá las bebidas solicitadas. Para indicar explícitamente este hecho, a la lista de predicados anterior habría que añadirle los predicados:

`(servido cl1 cerveza)` `(servido cl2 cerveza)`

3. En cuanto a los operadores, con la representación utilizada, tendríamos 4 operadores diferentes:

   - `(mover ?robot ?from ?to)`: Para indicar que el robot se mueve de una determinada localización de partida `?from` a una localización de destino `?to`. Para ello, el robot se debe encontrar en la localización de partida y, una vez acabada la acción, el robot dejará de estar en la localización de partida y pasará a la de destino.
   - `(coger-vaso ?robot ?dep ?vas ?sec_vasos)`: El robot `?robot` que tiene un deposito `?dep` coge un vaso `?vas` de la sección de vasos `?sec_vasos`. Las precondiciones serían que el robot tiene el deposito vacio, y que se encuentra en la sección de vasos, al igual que el vaso que quiere coger. Los efectos serían que el depósito del robot pasa a estar `(lleno-vaso ?dep ?vas)`.
   - `(dispensar ?robot ?dep ?vas ?disp ?bebida ?sec_disp)`: El robot `?robot`, que tiene un deposito `?dep` con un vaso `?vas`, sirve la bebida `?bebida` del dispensador `?disp` que se encuentra en la seccion de dispensadores `?sec_disp`. Las precondiciones serían que el robot se encuentra en la sección de dispensadores, al igual que el dispensador, que su deposito tiene un vaso, y que el dispensador dispensa la bebida indicada por parámetro. El efecto sería que `(lleno ?vas ?bebida)`.
   - `(servir ?robot ?cliente ?bebida ?barra)`: El robot sirve la bebida al cliente. Las precondiciones serían que el cliente haya solicitado la bebida pasada por parámetro, que el robot se encuentre en la barra, y que tenga un deposito con la bebida solicitada. El efecto sería que el cliente se encuentra servido, es decir, `(servido ?cliente ?bebida)`.

   Podríamos tener, además, una acción `(anotar ?cliente ?bebida)` que tuviera como efecto `(peticion ?cliente ?bebida)`, puesto que el enunciado nos dice explícitamente que el robot **anota** los pedidos de los clientes. Sin embargo, en esta modelización se da por hecho que los predicados `peticion` ya se encuentran en el estado inicial, y no es necesario activar ninguna acción para incluirlos.

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

```
(:action dispensar
:parameters (?r - robot ?d - deposito ?v - vaso ?di - dispensador ?b - bebida ?s - seccion_dispensadores)
:precondition (and (at-robot ?r ?s) (at-dispensador ?di ?s) (sirve ?di ?b) (lleno-vaso ?d ?v))
:effect
(and (lleno ?v ?b)))
```