uc3m | Universidad Carlos III de Madrid

Preparación Parcial 1

Inteligencia Artificial Colmenarejo

Curso 2022-2023

Instrucciones

- Lea atentamente todo el enunciado antes de comenzar a contestar.
- Cualquier dispositivo electrónico deberá permanecer APAGADO durante la prueba.
- Solamente se corregirán los ejercicios contestados con bolígrafo azul o negro. Por favor, no utilice lápiz.
- Solamente se corregirán las hojas con los datos de alumno rellenas.
- Por favor, rellénelos en todas las hojas que entregue.
- La duración del examen es de 100 MINUTOS.

Ejercicio 1 (5p/10p)

Se quiere desarrollar un robot capaz de mover los libros que están depositados en una mesa de la zona de devoluciones de una biblioteca de forma automática. Para ello, se asume que los alumnos podrán dejar los libros en una mesa (formando una o varias pilas) y que el robot debe unificarlos en una sola pila dentro de una zona de libros a colocar para que el operario los coloque donde corresponda.
Los operadores disponibles serán:

- Coger libro y desplazarse a “a colocar”.
- Dejar libro y desplazarse a devoluciones.

Se asumen los siguientes predicados:

- Apilado(<id>, <elemento inferior>): donde id será el identificador del libro y elemento inferior será sobre lo que esta apilado: ya sea otro libro, una pila de la zona de devoluciones o una pila de la zona de libros a colocar. Se asume que la pila (o pilas) de cada zona se encuentra en una única mesa que se representa mediante este mismo predicado, por ejemplo: Apilado(“Mesa”,“Colocar”).
- Cogido (<id>): donde id indica el libro que el robot tiene cogido.
- Localizado(<elemento>, <lugar>): donde el elemento podrá ser el robot, un libro o una de las mesas y el lugar podrá ser una de las dos zonas (devoluciones o a colocar).

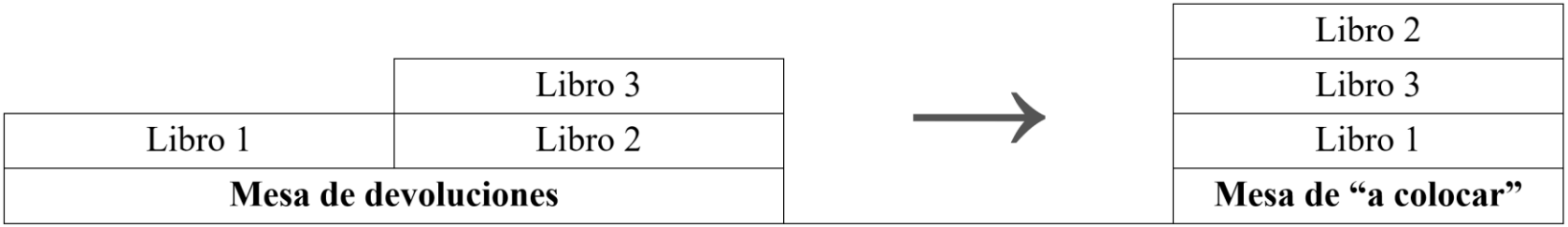
Ejercicio 1 (5p/10p)

El estado inicial del sistema tiene el robot en la zona de devoluciones y los libros están situados sobre la mesa de devoluciones como se muestra en la parte izquierda de la siguiente figura. El objetivo del problema es mover los libros para conseguir el resultado de la parte derecha.

Los hechos pueden representarse utilizando tanto variables no instanciadas (X, Y, Z …), como valores concretos con comillas.
Los tipos son:

- Libro = {‘L1’, ‘L2’, ‘L3’}
- Mesa = {‘M1’, ‘M2’}
- Robot = ‘R’
- Zona = {‘Devoluciones‘, ‘Colocar’}

EJ: Localizado(X, ‘Colocar’), será un elemento X en la zona Colocar.



Ejercicio 1 (5p/10p)

Resuelva las siguientes cuestiones:

- **Apartado A** (1 punto)
  - Proponga los hechos a introducir en el sistema según el estado inicial propuesto.
- **Apartado B** (1 punto)
  - Proponga los hechos para definir el estado meta según el estado objetivo propuesto.
- **Apartado C** (1.5 puntos)
  - Proponga las dos reglas para implementar los operadores descritos.
  - Nota: para realizar las reglas se aceptan operadores AND (∧), OR (∨) y NOT(¬).
- **Apartado D** (1.5 puntos)
  - Comprobar si es deducible el estado final mediante encadenamiento hacia delante.

Nota: no es necesario utilizar la tabla vista en ejercicios. Proponed un formato más corto comprensible si se considera necesario por tiempo. Deben quedar claras las modificaciones en cada paso.

Ejercicio 2 (5p/10p)

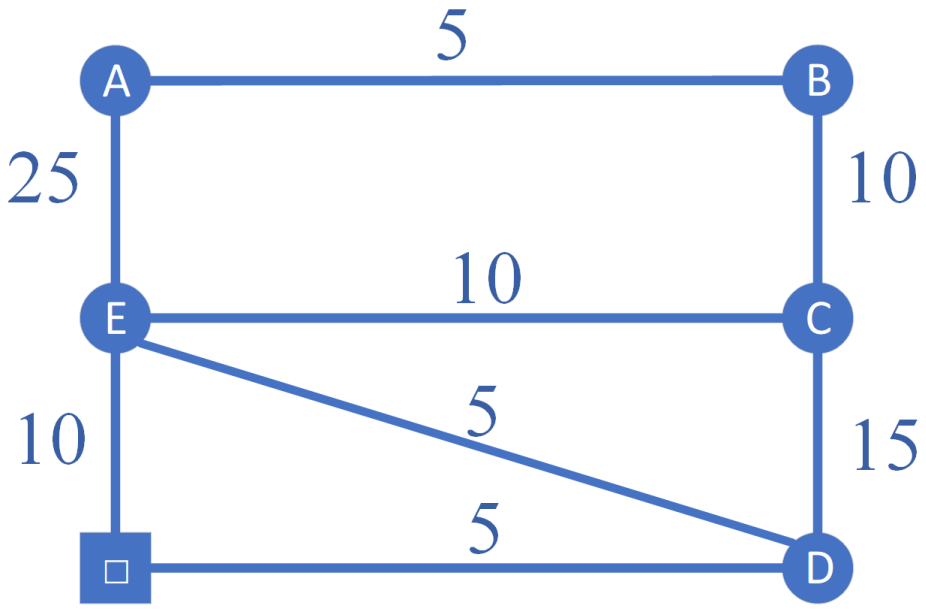
Una empresa suministradora de internet quiere instalar una línea de fibra óptica dentro de una urbanización. Para ello, debe colocar un único cable (sin ramificaciones, pero puede pasar por el mismo nodo varias veces) que conecte todos los postes telefónicos de la urbanización, utilizando la menor longitud de cable posible.

Para solucionar el problema la empresa busca la utilización de técnicas de búsqueda que proporcionen soluciones al problema.
El estado inicial propuesto solo tendrá conectada la centralita de fibra óptica, mientras que el estado final tendrá conectados todos los postes telefónicos con la centralita.

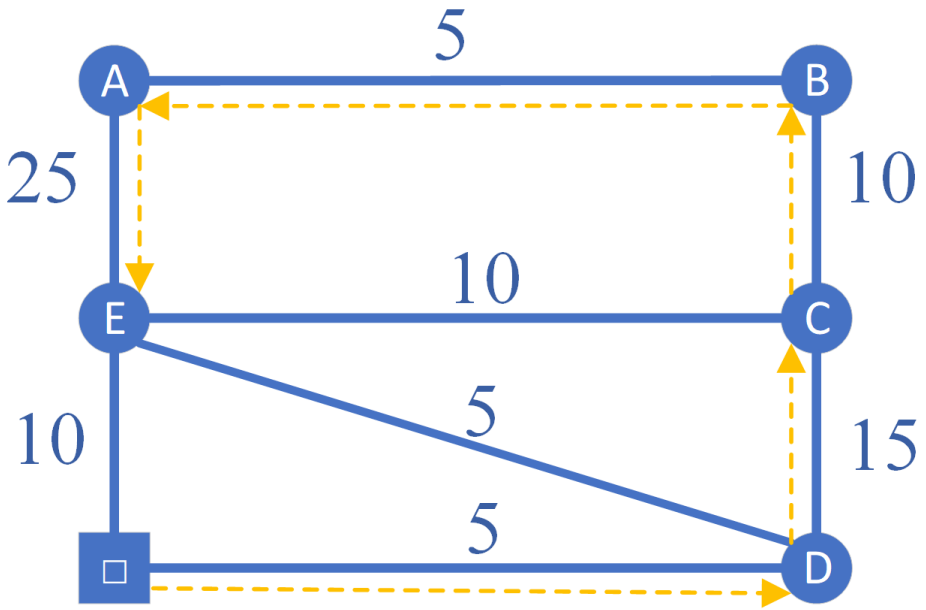
Ejercicio 2 (5p/10p)

La siguiente figura muestra un esquema con los distintos postes (círculos A-E), la centralita de suministro (cuadrado) y las calles que los interconectan (con la distancia):

Original



Ejemplo de solución



Ejercicio 2 (5p/10p)

Resuelva las siguientes cuestiones:

- **Apartado A** (1 punto)
  - Proponga una representación del estado dentro de este problema de búsqueda.
- **Apartado B** (1 punto)
  - Defina los operadores del problema de búsqueda.
- **Apartado C** (1.5 puntos)
  - Proponga una heurística válida para la resolución de este problema. Especifique el criterio de prioridad en caso de igualdad de nodos para la función de evaluación.
- **Apartado D** (1.5 puntos)
  - Utilizando la heurística anterior, obtenga la mejor solución con un algoritmo A*.

Nota1: es indispensable indicar para cada algoritmo de búsqueda, en cada paso realizado, los nodos que se generan, se expanden y se incluyen en la lista abierta y la lista cerrada. Se acepta back-propagation.

Nota2: por tiempo, no hace falta expandir nodos cuyo camino va y vuelve del mismo sitio. EJ: □, C, □

Ejercicio 1 (5p/10p)

El juego de hundir la flota consiste en localizar y derribar una serie de barcos de distintos tamaños situados por el rival dentro de una cuadricula NxN.

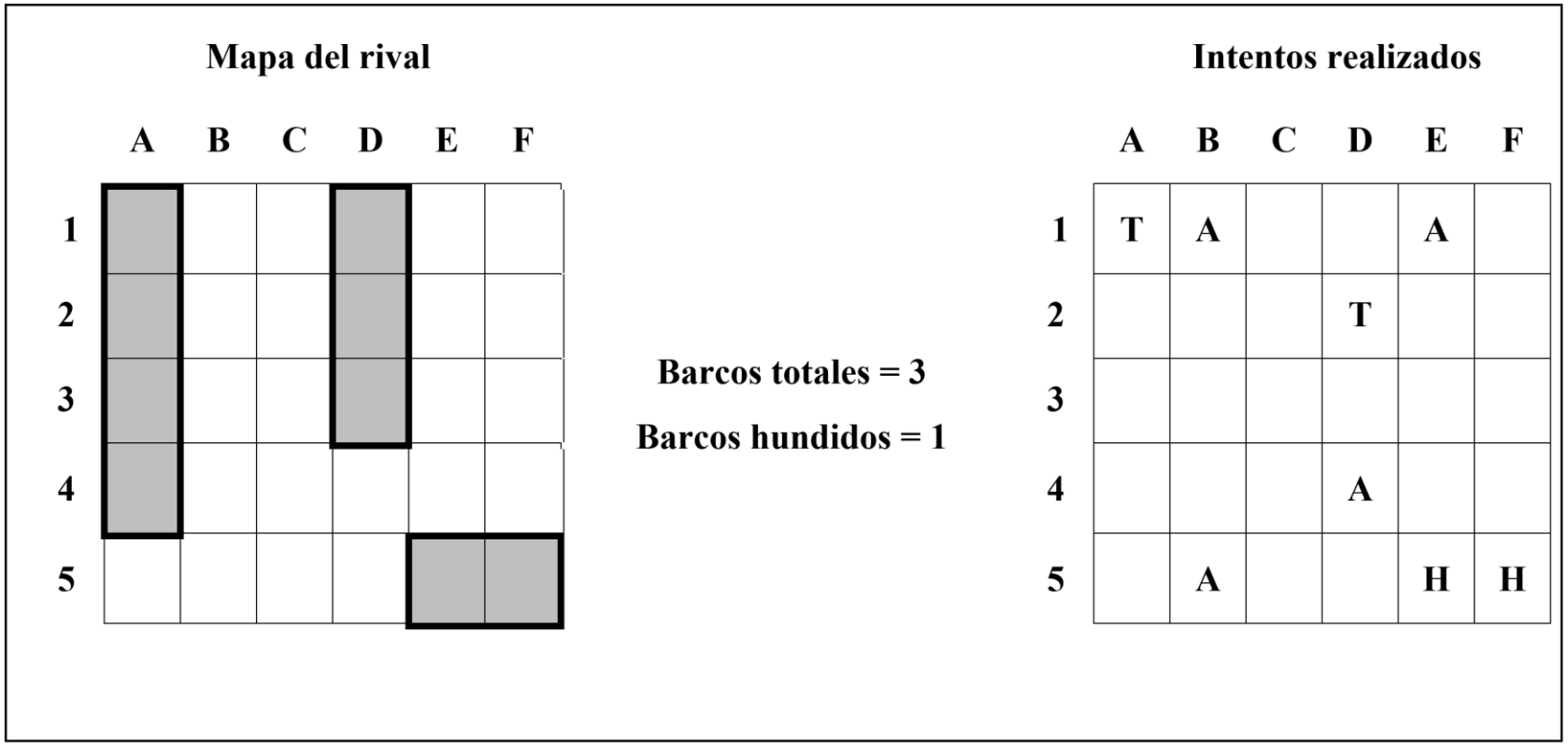
Un turno de juego consiste en “disparar” a una posición de la cuadrícula, pudiendo ocurrir lo siguiente:

- Si se impacta una casilla en la que no hay nada ningún barco, se marca el contenido de la casilla como Agua A y se cambia el turno.
- Si se impacta una casilla que está ocupando un barco, se marca el contenido de la casilla como Tocado T y se realiza un segundo disparo. Además, en el caso de que todas las casillas ocupadas por un barco estén marcadas con T el barco pasa a estar hundido, marcándose todas sus casillas con H.

La partida finaliza cuando las casillas de todos los barcos introducidos en la cuadrícula estén marcadas con H, es decir, cuando se hayan hundido todos los barcos.

Ejercicio 1 (5p/10p)

Una configuración posible sería la mostrada en la siguiente cuadrícula:



Nota: La posición de los barcos mostrada en el mapa del rival no sería conocido por el sistema inteligente, solo se muestra a efectos de un mejor entendimiento del problema.

Ejercicio 1 (5p/10p)

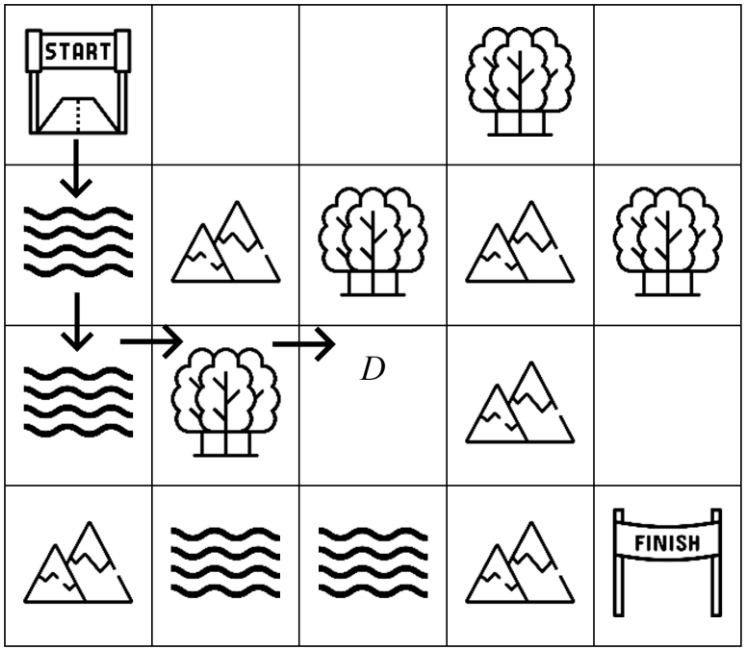
Resolver las cuestiones relativas a un sistema de inferencia que funciona como un jugador de este juego.

- **Apartado A** (1 punto)
  - Proponga los predicados necesarios para representar un estado del problema.
- **Apartado B** (1 punto)
  - Proponga los hechos a introducir en el sistema según el estado propuesto en la cuadricula mostrada de ejemplo.
- **Apartado C** (1 punto)
  - Proponga los hechos para definir el estado meta del ejemplo propuesto en la cuadrícula de ejemplo, aunque estos podrán estar situados en cualquier posición de esta.
- **Apartado D** (1 punto)
  - Proponga formalmente las reglas necesarias para realizar las acciones para acabar la partida mostrada como ejemplo, asumiendo que los hechos mostrados ya están en el sistema.
  - Nota: para realizar las reglas se aceptan operadores AND, OR y NOT.
- **Apartado E** (1 punto)
  - Comprobar si es deducible el estado final mediante encadenamiento hacia delante, haciendo uso de los hechos y reglas propuestos en los apartados B, C y D.

Ejercicio 2 (5p/10p)

Una carrera de triatlón consiste en una prueba en la que ir de un punto “A” a un punto “B” que incluye una etapa de natación, una de ciclismo y una de carrera a pie.

Alberto se enfrenta a una prueba de este tipo en la que debe atravesar el terreno mostrado en el mapa partiendo desde la casilla Start hasta la meta Finish pasando mínimo una vez por un bosque, una montaña y una zona de agua. Además, existe el terreno llano donde no hay restricción de paso.



Alberto ha practicado cuánto tarda en cada uno de los terrenos y sabe que tarda 10 minutos en atravesar el terreno llano, 20 en atravesar el bosque y 30 en una montaña o en agua.

Nota: está permitido el movimiento vertical y horizontal.

Ejercicio 2 (5p/10p)

Resuelva las siguientes cuestiones:

- **Apartado A** (1 punto)
  - ¿Cuál o cuáles de los algoritmos vistos en clase utilizarías (y cuáles no) para alcanzar una solución óptima en este problema? Explica el porqué de cada uno.
- **Apartado B** (1.5 puntos)
  - Proponga una representación del estado dentro de este problema de búsqueda.
- **Apartado C** (1.5 puntos)
  - Proponga una heurística válida para la resolución de este problema. Especifique el criterio de prioridad en caso de igualdad de nodos para la función de evaluación.
- **Apartado D** (1 punto)
  - Durante una ejecución del algoritmo seleccionado en el apartado A, se ha realizado el camino indicado en el mapa, llegando a la casilla D. Evalúe según su heurística (apartado C) y costes tanto el nodo actual como los nodos hijo posibles. Indique y justifique el nodo hijo escogido por el algoritmo.

Ejercicio 1 (5p/10p)

Una empresa ha tenido un hackeo masivo mediante fuerza bruta, obteniendo las contraseñas de todo el personal. Para evitar que vuelva a ocurrir, nos ha solicitado que, mediante un algoritmo que genere de forma automática una contraseña para cada usuario, que cumpla las siguientes restricciones:

- Longitud mínima de 4 caracteres.
- Longitud máxima de 8 caracteres.
- Más letras minúsculas que mayúsculas.
- Máximo dos caracteres especiales.
- Más números que letras minúsculas.
- No puede aparecer el mismo carácter dos veces seguidas.

Las contraseñas podrán ser formadas por los siguientes caracteres:

Letras mayúsculas: A, B, C	Números: 1, 2, 3
Letras minúsculas: a, b, c	Caracteres especiales: *, ^, &

Ejercicio 1 (5p/10p)

Responda a las cuestiones sobre cómo resolver el problema mediante representación e inferencia.

- **Apartado A** (1 punto)
  - Representa estado y acciones.
- **Apartado B** (2 puntos)
  - Proponga y explique una representación de los hechos necesarios en el problema y de las reglas que permitan generar una contraseña, cumpliendo las restricciones propuestas.
  - Nota: Si necesita una función auxiliar que proporcione información de la cadena, puede utilizarlas explicando su funcionamiento en texto. EJ: contarCaracteres(): función que proporciona la longitud actual de la contraseña.
- **Apartado C** (1 punto)
  - Explique y utilice un método de encadenamiento para analizar si las reglas propuestas permiten llegar a una contraseña válida.
- **Apartado D** (1 punto)
  - Ordene las reglas según la prioridad necesaria para que resuelva el problema.

SOL Ejercicio 1 (5p/10p)

Responda a las cuestiones sobre cómo resolver el problema mediante representación e inferencia.

- **Apartado A** Representa estado y acciones.

En este apartado se pide hacer un planteamiento del ejercicio como si fuese a tener una solución de búsqueda, siendo necesario indicar como se representa una situación concreta del problema (estado) y las acciones que permiten modificar un estado para generar otro nuevo.

El estado del problema será la cadena de la contraseña y cómo evoluciona a lo largo del tiempo mientras se construye. De forma adicional, se pueden almacenar variables que representen la cantidad de repeticiones de caracteres según su tipo, o el último carácter introducido.

Las acciones del problema utilizarán el estado añadiendo un carácter extra a la cadena de caracteres de la contraseña.

Las acciones serán 4, una por introducir un tipo de carácter, comprobando que todas las condiciones iniciales permiten introducir dicho carácter. Si se cuenta con las variables adicionales, se actualizarán teniendo en cuenta el carácter introducido.

SOL Ejercicio 1 (5p/10p)

Responda a las cuestiones sobre cómo resolver el problema mediante representación e inferencia.

- **Apartado B**

Este apartado tiene el mismo objetivo que el anterior, pero diseñado para representación e inferencia (incluyendo una representación formal). Además, se pide realizar la implementación de las reglas que resuelvan el problema.

Un hecho será lo definido anteriormente como estado. También se podría realizar un planteamiento de caracteres desacoplados, con un par cadenaCaracter(carácter, posición), aunque de forma más compleja.

Las reglas serán similares a las acciones anteriores. También se puede contemplar la creación de una regla adicional que valide de forma definitiva si cadena cumple las características y termina la ejecución:

AñadirMinus(X, "a"). Añade un caracter “a” minúscula al final de la cadena. En la solución habría que explicar que son necesarias otras dos versiones de esta regla: AñadirMinus(X, "b") y AñadirMinus(X, "c").

SOL Ejercicio 1 (5p/10p)

Responda a las cuestiones sobre cómo resolver el problema mediante representación e inferencia.

- **Apartado B**

AñadirMinus(X, "a"). Añade un caracter “a” minúscula al final de la cadena. En la solución habría que explicar que son necesarias otras dos versiones de esta regla: AñadirMinus(X, "b") y AñadirMinus(X, "c").

```
longitudCadena(X) < 8          ^   // Como mucho puede activarse si tiene 7 caracteres
numNumeros(X) > numMinus(X)+1  ^   // Más números que minúsculas (+1 para garantizarlo tras añadir)
lastCaracter(X) != "a"             // Que el anterior caracter no fuese una a
⇒
añadirCaracter(X, "a")         ^   // Añade el caracter 1 al final de la cadena
modified(X)                        // Permite continuar con la ejecución
```

AñadirMayus(X, "A"). Añade un caracter “a” minúscula al final de la cadena. En la solución habría que explicar que son necesarias otras dos versiones de esta regla: AñadirMinus(X, "b") y AñadirMinus(X, "c").

```
longitudCadena(X) < 8          ^   // Como mucho puede activarse si tiene 7 caracteres
numMinus(X) > numMayus(X)+1    ^   // Más minúsculas que mayúsculas (+1 para garantizarlo tras añadir)
lastCaracter(X) != "A"             // Que el anterior caracter no fuese una A
⇒
añadirCaracter(X, "A")         ^   // Añade el caracter 1 al final de la cadena
modified(X)                        // Permite continuar con la ejecución
```

SOL Ejercicio 1 (5p/10p)

Responda a las cuestiones sobre cómo resolver el problema mediante representación e inferencia.

- **Apartado B**

AñadirEspecial(X, "*"). Añade un caracter “a” minúscula al final de la cadena. En la solución habría que explicar que son necesarias otras dos versiones de esta regla: AñadirMinus(X, "b") y AñadirMinus(X, "c").

```
longitudCadena(X) < 8          ^   // Como mucho puede activarse si tiene 7 caracteres
numEspecial(X) < 2             ^   // Tras esta regla debe haber como mucho 2 caracteres especiales
lastCaracter(X) != "*"         ^   // Que el anterior caracter no fuese un *
⇒
añadirCaracter(X, "*")         ^   // Añade el caracter 1 al final de la cadena
modified(X)                        // Permite continuar con la ejecución
```

AñadirNumero(X, "1"). Añade un caracter “a” minúscula al final de la cadena. En la solución habría que explicar que son necesarias otras dos versiones de esta regla: AñadirMinus(X, "b") y AñadirMinus(X, "c").

```
longitudCadena(X) < 8          ^   // Como mucho puede activarse si tiene 7 caracteres
lastCaracter(X) != "1"         ^   // Que el anterior caracter no fuese un *
⇒
añadirCaracter(X, "1")         ^   // Añade el caracter 1 al final de la cadena
modified(X)                        // Permite continuar con la ejecución
```

SOL Ejercicio 1 (5p/10p)

Responda a las cuestiones sobre cómo resolver el problema mediante representación e inferencia.

- **Apartado B**

ValidarCadena(X). Comprueba que la cadena generada con el resto de las reglas es válida y es activada cuando las demás no se pueden ejecutar. No añade un modified, de forma que termina la ejecución.

```
longitudCadena(X) <= 8          ^   // Como mucho puede tener 8 caracteres
longitudCadena(X) >= 4          ^   // Como mínimo puede tener 4 caracteres
numMinus(X) > numMayus(X)       ^   // Más minúsculas que mayúsculas
numNumeros(X) > numMinus(X)     ^   // Más números que minúsculas
numEspecial(X) <= 2             ^   // Como máximo 2 números especiales
⇒
System.out.println("FIN")
```

SOL Ejercicio 1 (5p/10p)

Responda a las cuestiones sobre cómo resolver el problema mediante representación e inferencia.

- **Apartado C** Explique y utilice un método de encadenamiento para analizar si las reglas propuestas permiten llegar a una contraseña válida.

Existen dos métodos de encadenamiento: hacia adelante y hacia atrás. Ambos tienen como objetivo probar que el problema es resoluble con las reglas implementadas.

En este caso, optamos por encadenamiento hacia adelante. En este problema y esta solución, al únicamente tratar con un hecho, es muy simple. Se añaden los valores de las funciones auxiliares, pero no son necesarios.

	Hechos	Valor de funciones auxiliares	Regla aplicada	Modificaciones activadas
1	X("")	1. longitudCadena(X) = 0 2. numNumeros(X) = 0 3. numMayus(X) = 0 4. numMinus(X) = 0 5. numEspecial(X) = 0 6. lastCaracter(X) = ""	AñadirEspecial(X, "*")	X("*")

SOL Ejercicio 1 (5p/10p)

Responda a las cuestiones sobre cómo resolver el problema mediante representación e inferencia.

- **Apartado C** Explique y utilice un método de encadenamiento para analizar si las reglas propuestas permiten llegar a una contraseña válida.

2	X("*")	1. longitudCadena(X) = 1 2. numNumeros(X) = 0 3. numMayus(X) = 0 4. numMinus(X) = 0 5. numEspecial(X) = 1 6. lastCaracter(X) = "*"	AñadirEspecial(X, "^")	X("*^")
3	X("*^")	1. longitudCadena(X) = 2 2. numNumeros(X) = 0 3. numMayus(X) = 0 4. numMinus(X) = 0 5. numEspecial(X) = 2 6. lastCaracter(X) = "^"	AñadirNumero(X, "1") (ya no se puede lanzar la regla AñadirEspecial)	X("*^1")
4	X("*^1")	1. longitudCadena(X) = 3 2. numNumeros(X) = 1 3. numMayus(X) = 0 4. numMinus(X) = 0 5. numEspecial(X) = 2 6. lastCaracter(X) = "1"	AñadirNumero(X, "2") (aun no se puede lanzar ni AñadirMinus ni AñadirMayus, al igualar el número de caracteres)	X("*^12")

SOL Ejercicio 1 (5p/10p)

Responda a las cuestiones sobre cómo resolver el problema mediante representación e inferencia.

- **Apartado C** Explique y utilice un método de encadenamiento para analizar si las reglas propuestas permiten llegar a una contraseña válida.

5	X("*^12")	1. longitudCadena(X) = 4 2. numNumeros(X) = 2 3. numMayus(X) = 0 4. numMinus(X) = 0 5. numEspecial(X) = 2 6. lastCaracter(X) = "2"	AñadirMinus(X, "a") (Se puede añadir una minúscula sin que iguale la cantidad de números)	X("*^12a")
6	X("*^12a")	1. longitudCadena(X) = 5 2. numNumeros(X) = 2 3. numMayus(X) = 0 4. numMinus(X) = 1 5. numEspecial(X) = 2 6. lastCaracter(X) = "a"	AñadirNumero(X, "1") (Si se añadiese un minus igualaría a numNumeros). Lo mismo con numMayus a numMinus)	X("*^12a1")
7	X("*^12a1")	1. longitudCadena(X) = 6 2. numNumeros(X) = 3 3. numMayus(X) = 0 4. numMinus(X) = 1 5. numEspecial(X) = 2 6. lastCaracter(X) = "1"	AñadirMinus(X, "a")	X("*^12a1a")

SOL Ejercicio 1 (5p/10p)

Responda a las cuestiones sobre cómo resolver el problema mediante representación e inferencia.

- **Apartado C** Explique y utilice un método de encadenamiento para analizar si las reglas propuestas permiten llegar a una contraseña válida.

8	X("*^12a1a")	1. longitudCadena(X) = 7 2. numNumeros(X) = 3 3. numMayus(X) = 0 4. numMinus(X) = 2 5. numEspecial(X) = 2 6. lastCaracter(X) = "a"	AñadirMayus(X, "A")	X("*^12a1aA")
9	X("*^12abAB")	1. longitudCadena(X) = 8 2. numNumeros(X) = 3 3. numMayus(X) = 1 4. numMinus(X) = 2 5. numEspecial(X) = 2 6. lastCaracter(X) = "A"	ValidarCadena(X)	-

SOL Ejercicio 1 (5p/10p)

Responda a las cuestiones sobre cómo resolver el problema mediante representación e inferencia.

- **Apartado D** Ordene las reglas según la prioridad necesaria para que resuelva el problema.

La prioridad de las reglas, al igual que en la práctica, es un factor clave para asegurar que la solución del problema se consigue con garantías. Esta depende completamente de las reglas generadas. En este caso, como las reglas garantizan que se escriba una cadena correcta todo el rato, la prioridad de las reglas no afecta en exceso a la solución del problema.

La prioridad de las reglas del mismo tipo de carácter sí afecta, seleccionándose una vez la primera y otra la segunda. La tercera no se ejecuta.

Según donde se coloque la regla ValidarCadena, la solución tendrá más o menos caracteres, pero siempre entre 4 y 8.

1. AñadirMayus(X, "A")
2. AñadirMayus(X, "B")
3. AñadirMayus(X, "C")
4. AñadirMinus(X, "a")
5. AñadirMinus(X, "b")
6. AñadirMinus(X, "c")
7. AñadirEspecial(X, "*")
8. AñadirEspecial(X, "^")
9. AñadirEspecial(X, "&")
10. AñadirNumero(X, "1")
11. AñadirNumero(X, "2")
12. AñadirNumero(X, "3")
13. ValidarCadena(X)

Ejercicio 1 (5p/10p)

El juego de buscaminas consiste en localizar bombas dentro de una cuadrícula MxN. Para ello, el jugador podrá pulsar una casilla por turno, de dos formas distintas: indicando que la casilla pulsada el jugador la asume como casilla vacía, o si la casilla pulsada la asume como casilla bomba.

El juego acaba cuando se localizan todas las bombas (ganando el juego) o cuando por error se pulsa una para marcarla como casilla vacía, pero en realidad es una casilla bomba (perdiendo el juego).

Para ayudar al jugador a detectar las bombas, el juego da un valor numérico a las casillas pulsadas que sean vacías. Este valor numérico indica cuantas bombas hay presentes en las 8 casillas adyacentes a la casilla vacía y por tanto se puede suponer lo siguiente:

- Si una casilla muestra un número N de bombas adyacentes, y ya tiene ese número de bombas en alguna casilla adyacente, el resto de las casillas adyacentes a la inicial que no estén marcadas pueden suponerse como casillas vacías.
- Si una casilla muestra un número N de bombas y el número de casillas sin pulsar adyacentes coincide con la diferencia entre N y la suma de bombas marcadas alrededor de la casilla original, dichas casillas sin pulsar pueden suponerse como casilla de bomba.

Ejercicio 1 (5p/10p)

La imagen adjunta muestra un tablero 6x10 de buscaminas con 15 bombas (B) al descubierto. En las casillas vacías aparece el valor numérico o una B si es una casilla bomba. Las casillas resaltadas (con borde grueso) no son conocidas por el jugador en el instante actual de la partida, por lo que no están pulsadas. Se muestra la información de dichas casillas por facilitar la compresión y resolución del problema por parte de los alumnos.

B	3	B	1	1	1
2	B	2	2	2	B
2	2	2	1	B	2
2	B	3	2	2	1
2	B	4	B	2	1
1	3	B	4	2	B
0	3	B	B	3	2
0	2	B	4	B	2
1	2	1	2	2	B
B	1	0	0	1	1

Ejercicio 1 (5p/10p)

Dado este enunciado, responda a las siguientes cuestiones:

- **Apartado A (1.5 puntos)** Proponga los predicados necesarios para representar un estado del problema.

- **Apartado B (1.5 puntos)** Proponga las reglas necesarias para que un supuesto jugador artificial realice acciones que permitan marcar una casilla como bomba, y marcar una casilla como vacía.

  NOTA: es asumible que de forma externa a la IA exista un motor de juego que inserte predicados que informen de la situación actual. Es decir, no es necesario construir reglas que calculen cuantos números adyacentes tiene una casilla, simplemente se insertaría el predicado de forma automática.

- **Apartado C (2 puntos)** Ante la situación dada en la imagen de ejemplo, ¿sería posible que el jugador artificial utilizando acciones de marcar casilla como vacía y marcar casilla como bomba pueda llegar a la victoria?, ¿o por el contrario falta información? Demuéstrelo y explique cómo mediante una técnica aprendida en clase.

Ejercicio 1 SOL

Apartado A (1.5 puntos) Proponga los predicados necesarios para representar un estado del problema.

El estado del problema esta formado por la situación actual de las casillas de juego, por tanto es necesario representar las NxM casillas junto a la relación que tienen entre si.

Casilla(x,y,N): para representar las casillas y su valor, que puede ser un número en el rango 0-8, una B para marcar que tiene bomba, y una V para marcar que no ha sido seleccionada.

Ady(x,y,m,n): para representar las adyacencias entre casillas, es un equivalente a la representación del tablero.

Ejercicio 1 SOL

Apartado B (1.5 puntos) Proponga las reglas necesarias para que un supuesto jugador artificial realice acciones que permitan marcar una casilla como bomba, y marcar una casilla como vacía.

Hay dos posibilidades de reglas que son automáticas porque sabemos seguro el resultado:

Casilla(x,y,V) ^ Casilla(m,n,0) ^ Ady(x,y,m,n)→ marcar(x,y,Vacia)

Casilla(x,y,V) ^ Casilla(m,n,8) ^ Ady(x,y,m,n)→ marcar(x,y,Bomba)

Para los números entre 1 y 7 necesitan de esa información del motor de juego que posibilite saber cuantas casillas tienen bomba alrededor de la casilla (m,n). Por ejemplo, pensemos que el motor de juego nos inserta cuentaBombas(m,n,N) con el número de bombas alrededor de m,n.

Casilla(x,y,V) ^ Casilla(m,n,A) ^ Ady(x,y,m,n) ^ cuentaBombas(m,n,B) ^ ~mayorQue(A,B)→ marcar(x,y,Vacia)

Casilla(x,y,V) ^ Casilla(m,n,A) ^ Ady(x,y,m,n) ^ cuentaBombas(m,n,B) ^ mayorQue(A,B)→ marcar(x,y,Bomba)

Nota: también se usa el predicado mayorQue(A,B) para indicar A>B

Ejercicio 1 SOL

Apartado C (2 puntos) Ante la situación dada en la imagen de ejemplo, ¿sería posible que el jugador artificial utilizando acciones de marcar casilla como vacía y marcar casilla como bomba pueda llegar a la victoria?, ¿o por el contrario falta información? Demuéstrelo y explique cómo mediante una técnica aprendida en clase.

Es posible utilizar un encadenamiento:

1. El recuadro verde puede marcarse como B ya que todas las casillas a su alrededor indican bomba, siendo esa la única casilla posible

2. Los recuadros naranjas sabemos que no son bomba por los 0 que tienen adyacentes.

B	3	B	1	1	1
2	B	2	2	2	B
2	2	2	1	B	2
2	B	3	2	2	1
2	B	4	B	2	1
1	3	B	4	2	B
0	3	B	B	3	2
0	2	B	4	B	2
1	2	1	2	2	B
B	1	0	0	1	1

Ejercicio 1 SOL

Apartado C (2 puntos) Ante la situación dada en la imagen de ejemplo, ¿sería posible que el jugador artificial utilizando acciones de marcar casilla como vacía y marcar casilla como bomba pueda llegar a la victoria?, ¿o por el contrario falta información? Demuéstrelo y explique cómo mediante una técnica aprendida en clase.

3. Para los recuadros azules sabemos que no pueden tener bomba porque todos los números de alrededor ya han cumplido su cuota de bombas.

4. El último cuadro restante es el mismo caso que el cuadro verde, los números de alrededor necesitan esa bomba adicional por lo que el recuadro tiene que tener una bomba

B	3	B	1	1	1
2	B	2	2	2	B
2	2	2	1	B	2
2	B	3	2	2	1
2	B	4	B	2	1
1	3	B	4	2	B
0	3	B	B	3	2
0	2	B	4	B	2
1	2	1	2	2	B
B	1	0	0	1	1

Ejercicio 2 (5p/10p)

Sokoban es un juego de tipo puzzle cuyo objetivo es que el jugador (P), que se encuentra dentro de un mapa, mueva una serie de cajas (B) a lo largo del mapa hasta ser colocadas todas ellas en casillas de destino (O). En la imagen adjunta se muestra un mapa inicial de Sokoban.

El jugador (P) se puede mover en las cuatro direcciones (arriba, abajo, izquierda o derecha). Además, cuando en su movimiento colisione con una caja (B), la desplazará también en dicha dirección (siempre y cuando no tenga un muro (M) u otra caja (B) en dicha dirección).

Cuando todas las cajas (B) estén en zonas de destino (O), el juego terminará con victoria del jugador (P).

M	M	M	M	M	M	M
M	P					M
M				B		M
M						M
M		B			O	M
M	O					M
M	M	M	M	M	M	M

Ejercicio 2 (5p/10p)

Resuelva las siguientes cuestiones:

Apartado A (0.5 puntos) ¿Este problema es un problema de búsqueda? Justifique su respuesta.

Apartado B (1 punto) Defina un estado del problema, así como instanciar con él el estado inicial y final de esta partida.

Apartado C (1 punto) Diseñe el espacio de estados, tanto para este mapa como para cualquier posible mapa del juego.

Apartado D (1.5 puntos) Proponga y explique una heurística admisible al problema.

Apartado E (1 punto) Suponiendo que el mapa es más complejo y se quiere encontrar una solución de forma rápida, ¿la heurística del apartado anterior sería adecuada? Justifique su respuesta y, en caso afirmativo, proponga una heurística mejorada para este supuesto.

Ejercicio 2 SOL

Apartado A (0.5 puntos) ¿Este problema es un problema de búsqueda? Justifique su respuesta.

Sí, puesto tiene un estado inicial, según el mapa del juego, y un objetivo claro y definido, insertar las cajas en sus casillas de destino.

Ejercicio 2 SOL

Apartado B (1 punto) Defina un estado del problema, así como instanciar con él el estado inicial y final de esta partida.

En este tipo de ejercicio, de cosas moviéndose por un mapa, el estado debe representar, como mínimo, todos los movimientos posibles del jugador en dicho mapa. Además, es necesario conocer el estado de ciertos elementos dinámicos del tablero, como la posición de las cajas móviles, o si han sido colocadas en sus casillas de destino.

Una de las soluciones más sencillas es almacenar todo el mapa. Para ello, es necesario modelar es el estado de las celdas o casillas en cada instante, siendo el estado una lista de todas las casillas que forman el mapa.

Una forma de modelar cada casilla vista en clase es: Casilla(X, Y, Tipo)

- Donde X e Y son las coordenadas del mapa, por ejemplo, comenzando en X=1 Y=1 en la casilla superior izquierda, y aumentando en valor al descender (Y) / moverse a la derecha (X).
- En cuanto al tipo, será un carácter de los siguientes: Muro (M), Vacío (V), Caja (C), Caja en destino (C'), Destino sin caja (D), Jugador (P)

Otras soluciones que no modelen la posición de los destinos deberán añadir un contador de cajas colocadas. En este caso se ha utilizado "Caja en destino" para agilizarlo pudiendo ser inferido analizando el estado.

Apartado C (1 punto) Diseñe el espacio de estados, tanto para este mapa como para cualquier posible mapa del juego.

La forma sencilla es calcular qué puede haber en cada casilla del mapa. Según el problema, puede haber: N filas, M columnas, K tipos por casilla, y L cajas por partida.

Si somos estrictos, sabemos que el número L será el mismo para las cajas como para las casillas destino, por lo que se puede ajustar en detalle dicha fórmula. Además, en el instante inicial habrá L casillas de destino, pero este número disminuirá hasta 0 cuando se alcance la meta.

También sabemos que los muros son estáticos, por lo que, a efectos de combinatoria no afectan (no puede haber en ellos ni cajas, ni jugador, ni destino).

En el caso de que se coloquen cajas la combinatoria varía ligeramente.

En el mapa proporcionado la combinatoria sería:

7 filas, 7 columnas, 24 muros, 2 cajas, 2 destino, 1 player: 61 casillas en total.

61-24=37 casillas útiles. 37-5=32 vacío.

Se puede especificar más el valor, teniendo en cuenta que, si una caja está en una celda, la combinatoria de la siguiente caja (o jugador) no puede contemplar dicha celda, al estar ya ocupada.

celdaDisponible puede ser tanto vacío, como celda objetivo. Nunca muro. Como

Ejercicio 2 SOL

Apartado C (1 punto) Diseñe el espacio de estados, tanto para este mapa como para cualquier posible mapa del juego.

La forma sencilla es calcular qué puede haber en cada casilla del mapa. Según el problema, puede haber: N filas, M columnas, K tipos por casilla, y L cajas por partida.
Si somos estrictos, sabemos que el número L será el mismo para las cajas como para las casillas destino, por lo que se puede ajustar en detalle dicha fórmula. Además, en el instante inicial habrá L casillas de destino, pero este número disminuirá hasta 0 cuando se alcance la meta.
También sabemos que los muros son estáticos, por lo que, a efectos de combinatoria no afectan (no puede haber en ellos ni cajas, ni jugador, ni destino).
En el caso de que se coloquen cajas la combinatoria varía ligeramente.
En el mapa proporcionado la combinatoria sería:
7 filas, 7 columnas, 24 muros, 2 cajas, 2 destino, 1 player: 61 casillas en total.
61-24=37 casillas útiles. 37-5=32 vacío.

Ejercicio 2 SOL

Apartado C (1 punto) Diseñe el espacio de estados, tanto para este mapa como para cualquier posible mapa del juego.

Se puede especificar más el valor, teniendo en cuenta que, si una caja está en una celda, la combinatoria de la siguiente caja (o jugador) no puede contemplar dicha celda, al estar ya ocupada.
celdaDisponible puede ser tanto vacío, como celda objetivo. Nunca muro. Como generalización puede considerarse MxN (filas y columnas).

$$2^{(M\cdot N)-O-J-B}\cdot(M\cdot N)^{O}\cdot(M\cdot N)^{J}\cdot(M\cdot N)^{B}$$

Forzar mismo número de O que de B.

$$(celdasDisponibles)\cdot(celdasDisponibles-1)\cdot\cdots$$

Si somos específicos, existirán situaciones imposible de alcanzar según la instancia inicial del mapa, aunque representables en el espacio de estados.

Ejercicio 2 SOL

Apartado D (1.5 puntos) Proponga y explique una heurística admisible al problema.

Una heurística admisible se obtiene a partir de una versión relajada del problema. En este caso, podemos tener en cuenta la distancia Manhattan del jugador a una posición a cada bola y desde cada bola a su destino más cercano. En este caso dejamos de lado el hecho de que cada caja deba ir a una única celda destino.

Además, se deja de lado el hecho de que el movimiento de desplazar una caja no es idéntico al movimiento del jugador, es más complejo al requerir dos casillas de margen.

También son aceptadas otras soluciones más simples, como la distancia del jugador a la posición destino, pese a ser peores heurísticas para el problema, aunque admisibles.

Ejercicio 2 SOL

Apartado E (1 punto) Suponiendo que el mapa es más complejo y se quiere encontrar una solución de forma rápida, ¿la heurística del apartado anterior sería adecuada? Justifique su respuesta y, en caso afirmativo, proponga una heurística mejorada para este supuesto.

Este apartado está directamente relacionado con el anterior. Puesto que el anterior realiza una heurística admisible, es probable sea demasiado cercana a un coste real del problema, y, por ende, que no guíe adecuadamente al algoritmo de búsqueda para tomar decisiones que realmente le acerquen a una hipotética solución. Por tanto, este apartado pretende que se expliquen mejoras (partiendo de la heurística explicada en el apartado D) cuyo valor sea más realista al del problema, sin tener que garantizar que la heurística no sobreestime el coste.

Algunas propuestas válidas podrían ser:

- Hacer la suma de ir de P a cada bola más ir de la bola a su lugar destino más cercano.
- Concatenar de destino más cercano a la bola.