Departamento de Informática
Universidad Carlos III de Madrid

Grado en Ingeniería Informática

Inteligencia Artificial

Febrero 2016. 1^a examen

Normas generales del examen

- El tiempo para realizar el examen es de **2 horas**
- Entregad cada problema en hojas separadas
- Solo se responderán preguntas sobre el examen los primeros 30 minutos
- Si se sale del aula, no se podrá volver a entrar durante el examen
- No se puede presentar el examen escrito a lápiz

Problema 1. (2 puntos)

En un sistema de producción se han introducido las siguientes reglas:

R1: IF A(X), X > 1, ∼ B(Y) THEN ∼A(X), A(X - 1), B(X)

R2: IF A(X), X > 1, B(Y) THEN ∼A(X), ∼B(Y), B(X + Y), A(X - 1)

R3: IF A(1), B(X) THEN ∼A(1), ∼B(X), A(X), `halt`

El símbolo ∼ significa borrar de la base de hechos y `halt` parar el sistema de producción.
La base de hechos o memoria de trabajo inicial contiene: A(5).
Se pide:

1. (1,5 puntos) Mostrar la secuencia de reglas ejecutadas detallando los hechos de la memoria de trabajo y el conjunto conflicto para cada ciclo de ejecución ordenado siguiendo una estrategia de resolución del conjunto conflicto LIFO (profundidad o último en entrar primero en salir).
2. (0,5 puntos) Si la base de hechos inicial contuviera el hecho A(7), ¿cual sería el contenido de la base de hechos al parar el sistema de producción?

Problema 2. (2 puntos)

El siguiente grafo representa un mapa entre ciudades, donde los números en los arcos indican la distancia por carretera. La heurística h expresa la distancia en línea recta desde cada nodo hasta la meta (nodo G). El estado inicial es A.

1.1

1. BH_0: {A(5)} CC_0: {R1(x=5)}

BH_1: {A(4), B(5)} CC_1: {R2(x=4, y=5)}

BH_2: {A(3), B(9)} CC_2: {R2(x=3, y=9)}

BH_3: {A(2), B(12)} CC_3: {R2(x=2, y=12)}

BH_4: {A(1), B(14)} CC_4: {R3(x=1, y=14)}

BH_5: {A(14)} CC_5: { Fin

2. BH_0 = {A(5), A(7)} CC_0 = {R1(x=5), R1(x=7)}

BH_1 = {A(5), A(6), B(7)} CC_1 = {R2(x=5, y=7), R2(x=6, y=7)}

BH_2 = {~~A(5)~~, A(5), B(13)} CC_2 = {R2(x=5, y=13)}

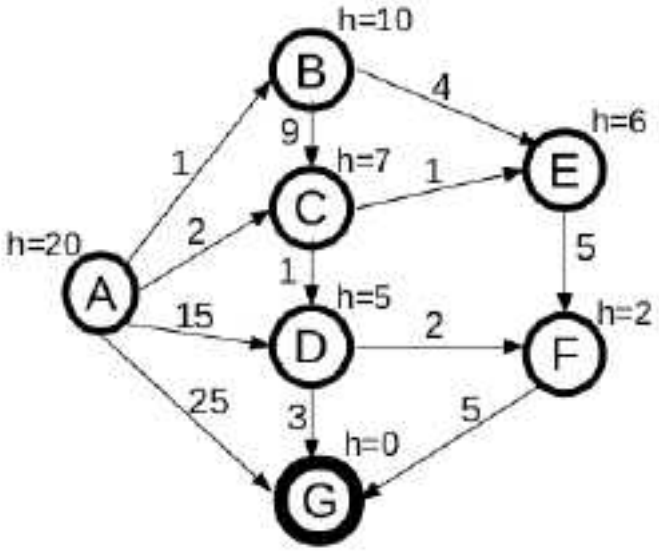
BH_3 = {A(4), B(18)} CC_3 = {R1(x=4, y=18)}

BH_4 = {A(3) B(22)} CC_4 = {R2(x=3, y=22)}

BH_5 = {A(2), B(25)} CC_5 = {R2(x=2, y=25)}

BH_6 = {A(1), B(27)} CC_6 = {R3(x=27)}

BH_7 = {A(27)}) Fin: halt.



Se pide:

1. (1) Dibujar el árbol de búsqueda que generaría A*, indicando el orden de los nodos generados y expandidos y los valores de g y h. Para la generación de sucesores considerar el orden alfabético.
2. (1) ¿Es la función heurística admisible? ¿Por qué? ¿Es la solución encontrada por A* óptima? ¿Por qué?

Problema 3. (3 puntos)

El Rush Hour es un juego basado en un puzle donde existe un conjunto de vehículos (coches y camiones) que pueden ser desplazados en sentido horizontal o vertical en un tablero. El objetivo del juego consiste en conducir el coche de color rojo horizontal que representa al jugador a través de las diferentes casillas hasta que consigue escapar por la salida. Existe una única salida en una posición fija del tablero. Las restricciones del problema son las siguientes: (a) los coches ocupan un total de dos cuadrículas, mientras que los camiones ocupan tres, (b) el tamaño del tablero es constante (6x6), (c) puede utilizarse un máximo de C coches y T camiones, variables en cada problema, (d) los vehículos sólo pueden desplazarse en el sentido en el que están orientados (en horizontal o en vertical) y no pueden pasar por encima de otro vehículo.

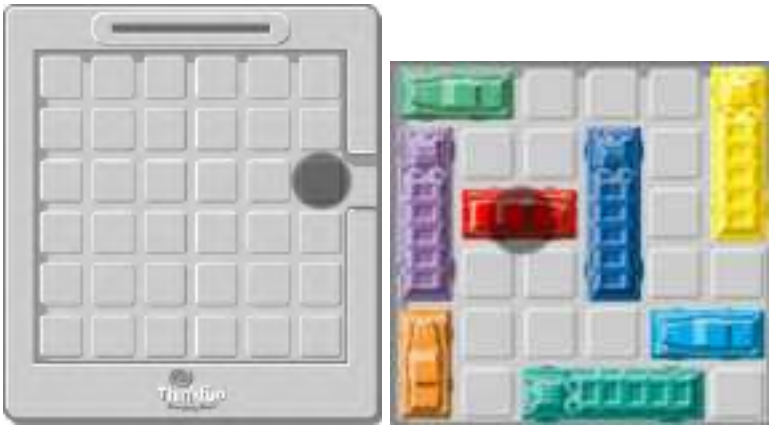


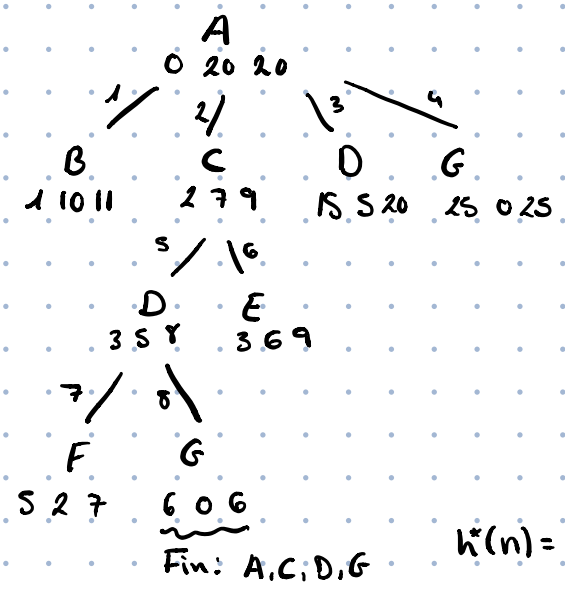
Figura 1: (Izquierda) Tablero vacío. El círculo sombreado representa la posición de la salida. (Derecha) Problema de ejemplo. El círculo sombreado indica el coche rojo del jugador.

Suponiendo que se quiere modelar el puzle utilizando sistemas de producción. **Se pide**:

1. (1 punto) Diseñar qué se debería representar en la base de hechos o memoria de trabajo y poner un ejemplo.
2. (2 puntos) Definir las reglas del sistema de producción.

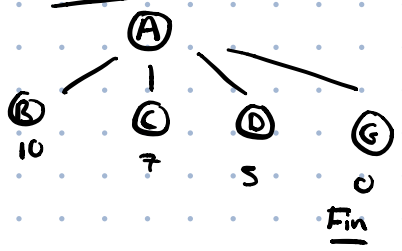
2.)

A*



Fin: A, C, D, G

Escalada



$h^*(n) = 6 < h(n) = 20$ No admisible, sobreestima.

Es optima, aunque no se garantiza al no ser admisible.

BH:

Vehiculo (X, Y, V, W, ori)
- X : valor horizontal arriba
- Y : valor vertical derecha
- W : valor vertical abajo
- V : valor horizontal derecha

ori ∈ ('v', 'h')

yo (x, y, w)
- x: hori derech
- y: vertical
- w: hori abajo

ocupado (x, y)
- x: hori
- y: verti.

BR:

MoverArriba:

~ocupado(x,y), vehiculo(x, y+1, x, z, 'v')

→ ~ocupado(x,z), ocupado(x, y+1), vehiculo(x, y, x, z-1)

MoverAbajo:

~ocupado(x,z), vehiculo(x, y, x, z-1, 'v')

→ ~ocupado(x,z), ocupado(x, z-1), vehiculo(x, y-1, x, z)

... Dch. Izq.

... Yo Dch. Izq

Fin: yo(6, 5, 3) → STOP

Problema 4. (3 puntos)

Suponer que en el problema anterior se quiere resolver utilizando algoritmos de búsqueda.
Se pide:

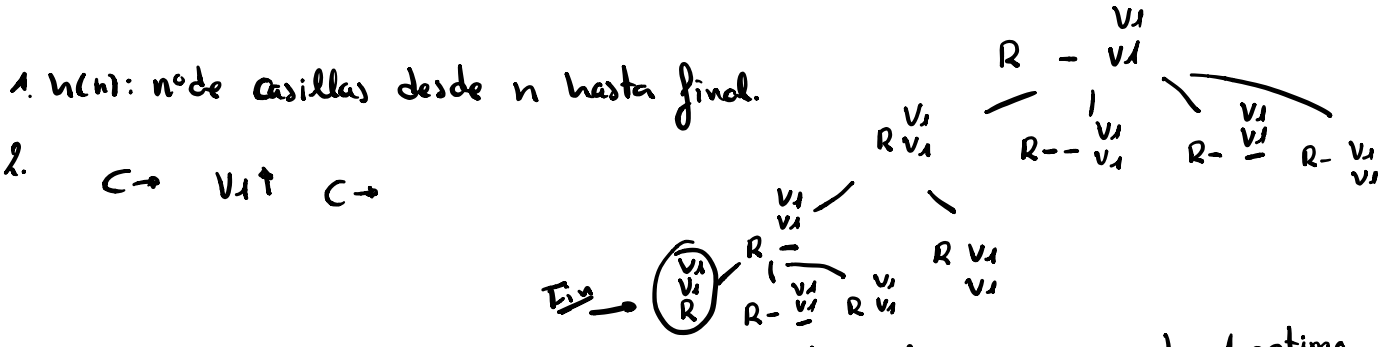
1. (1 puntos) Definir alguna heurística admisible.
2. (1.5 puntos) Considerar una versión simplificada del juego en la que el tablero tenga 4x4 casillas, los camiones ocupen 2 posiciones, los coches sólo 1 y la salida esté en la última columna de la fila del coche del jugador. Un problema simplificado podría ser el que se representa a continuación, donde sólo hay un camión vertical representado por v1 y el coche del jugador representado por R, que sólo se puede mover en horizontal.

-	-	-	-
-	-	-	v1
-	R	-	v1
-	-	-	-

   Dibujar el árbol de búsqueda que generaría una búsqueda en escalada (*hill-climbing*) utilizando como función de evaluación la heurística definida en el apartado 1.
3. (0,5 puntos) ¿Se puede asegurar que el algoritmo en escalada con la heurística admisible elegida encontrará la solución óptima a cualquier problema que se defina para este puzle?

4.)

1. h(n): nº de casillas desde n hasta final.

2. 

3. No se puede asegurar si es admisible, ya que escalada no lo siempre encuentre el optimo.

Solución Problema 1

1. La secuencia pedida es:

   BH_0={A(5)}, CC_0={(R1,X=5)}
   BH_1={A(4), B(5)}, CC_1={(R2,X=4,Y=5)}
   BH_2={A(3), B(9)}, CC_2={(R2,X=3,Y=9)}
   BH_3={A(2), B(12)}, CC_3={(R2,X=2,Y=12)}
   BH_4={A(1), B(14)}, CC_4={(R3,X=14)}
   BH_5={A(14)}, CC_5={}

2. La BH al parar el motor de inferencia tendrá como único hecho A(27).

Solución Problema 2

1. El árbol pedido se muestra en la Figura 2.

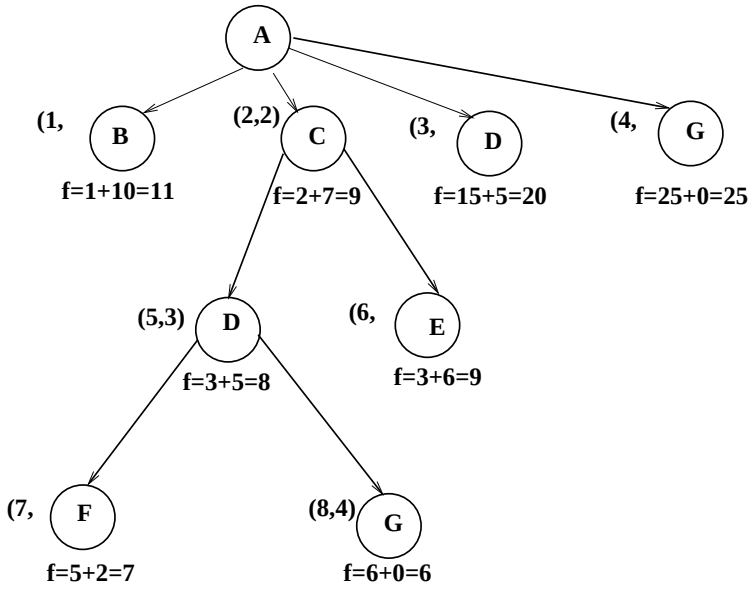


Figura 2: Árbol de búsqueda

2. Para que una heurística sea admisible se tiene que cumplir que **para todos los nodos** se cumple que no hay un camino desde ese nodo a la meta con coste (g) menor que su valor de h. En este caso, ni el nodo C ni el nodo D cumplen esta restricción, desde el nodo C existe un camino a G de coste 4 (C-D-G) y sin embargo h=7. Desde D se puede llegar a G con coste 3 y el valor de h es 5. Además, el valor heurístico de A es 20 y el algoritmo A* encuentra un camino de A a G de coste 6.

   En este ejemplo A* ha encontrado la solución óptima porque ha encontrado el camino entre los puntos A y G de menor coste posible en el grafo dado. Sin embargo para que el A* pueda garantizar optimalidad se tiene que cumplir que la heurística sea admisible, el coste mayor que 0 en todos los nodos y que haya un número finito de sucesores.

Solución Problema 3

1. Una posible forma para representar la base de hechos sería utilizar dos predicados:

   - `(casilla X Y Ocupada)`: donde X e Y son dos enteros que representa las coordenadas de la casilla y Ocupada puede tomar los valores si o no, dependiendo si la casilla está ocupada por un coche o libre.

- `(vehiculo Id Orientacion Xi Yi Xf Yf)`: donde Id representa el identificador del vehículo, Orientacion puede ser horizontal (h) o vertical (v), (Xi,Yi) representa las coordenadas del principio del vehículo y (Xf, Yf) las coordenadas del final. El principio se considera la parte que esté mas cerca de la casilla (1,1).

Con estos predicados y tomando como origen de coordenadas la esquina inferior izquierda, el ejemplo de la Figura 1 se representaría con los siguientes hechos:

```
(casilla 1 1 si) (casilla 2 1 no) (casilla 3 1 si) (casilla 4 1 si) (casilla 5 1 si) (casilla 6 1 no)
(casilla 1 2 si) (casilla 2 2 no) (casilla 3 2 no) (casilla 4 2 no) (casilla 5 2 si) (casilla 6 2 si)
(casilla 1 3 si) (casilla 2 3 no) (casilla 3 3 no) (casilla 4 3 si) (casilla 5 3 no) (casilla 6 3 no)
(casilla 1 4 si) (casilla 2 4 si) (casilla 3 4 si) (casilla 4 4 si) (casilla 5 4 no) (casilla 6 4 si)
(casilla 1 5 si) (casilla 2 5 no) (casilla 3 5 no) (casilla 4 5 si) (casilla 5 5 no) (casilla 6 5 si)
(casilla 1 6 si) (casilla 2 6 si) (casilla 3 6 no) (casilla 4 6 no) (casilla 5 6 no) (casilla 6 6 si)
(vehiculo dorado v 1 1 1 2) ;;coche dorado
(vehiculo verde h 3 1 5 1) ;;camion verde
(vehiculo cian h 5 2 6 2) ;;coche cian
(vehiculo morado v 1 3 1 5) ;;camion morado
(vehiculo azul v 4 3 4 5) ;;camion azul
(vehiculo ROJO h 2 4 3 4) ;;coche ROJO
(vehiculo amarillo v 6 4 6 6) ;;camion amarillo
(vehiculo verd2 h 1 6 2 6) ;;coche verde
```

2. Las reglas del sistema de producción serían:

```
(defrule fin
  (declare (salience 100))
  (vehiculo ROJO h 5 4 6 4)
=>
 (printout t "Conseguido " crlf)
 (halt)
)
(defrule derecha
  ?veh <- (vehiculo ?id h ?xi ?y ?xf ?y)
  ?cas1 <- (casilla ?xc ?y no) ;;casilla a ocupar
  ?cas2 <- (casilla ?xi ?y si) ;;casilla a liberar
  (test (= ?xc (+ ?xf 1)))
=>
 (retract ?veh ?cas1 ?cas2)
 (assert (vehiculo ?id h (+ ?xi 1) ?y (+ ?xf 1) ?y))
 (assert (casilla ?xc ?y si))
 (assert (casilla ?xi ?y no))
 (printout t "Mueve derecha " ?id " de " ?xf " a " (+ ?xf 1) crlf)
)

(defrule izquierda
  ?veh <- (vehiculo ?id h ?xi ?y ?xf ?y)
  ?cas1 <- (casilla ?xc ?y no) ;;casilla a ocupar
  ?cas2 <- (casilla ?xf ?y si) ;;casilla a liberar
  (test (= ?xc (- ?xi 1)))
=>
 (retract ?veh ?cas1 ?cas2)
 (assert (vehiculo ?id h (- ?xi 1) ?y (- ?xf 1) ?y))
 (assert (casilla ?xc ?y si))
 (assert (casilla ?xi ?y no))
 (printout t "Mueve izquierda " ?id " de " ?xi " a " (- ?xi 1) crlf)
)

(defrule abajo
  ?veh <- (vehiculo ?id v ?x ?yi ?x ?yf)
  ?cas1 <- (casilla ?x ?yc no) ;;casilla a ocupar
  ?cas2 <- (casilla ?x ?yf si) ;;casilla a liberar
  (test (= ?yc (- ?yi 1)))
```

```
=>
 (retract ?veh ?cas1 ?cas2)
 (assert (vehiculo ?id v ?x (- ?yi 1) ?x (- ?yf 1)))
 (assert (casilla ?x ?yc si))
 (assert (casilla ?x ?yf no))
 (printout t "Mueve abajo " ?id " de " ?yi " a " (- ?yi 1) crlf)
)

(defrule arriba
  ?veh <- (vehiculo ?id v ?x ?yi ?x ?yf)
  ?cas1 <- (casilla ?x ?yc no) ;;casilla a ocupar
  ?cas2 <- (casilla ?x ?yi si) ;;casilla a liberar
  (test (= ?yc (+ ?yf 1)))
=>
 (retract ?veh ?cas1 ?cas2)
 (assert (vehiculo ?id v ?x (+ ?yi 1) ?x (+ ?yf 1)))
 (assert (casilla ?x ?yc si))
 (assert (casilla ?x ?yi no))
 (printout t "Mueve arriba " ?id " de " ?yf " a " (+ ?yf 1) crlf)
)
```

Nota: están en sintaxis de CLIP, retract significa borrar de la BH, assert añadir en la BH y printout escribe en pantalla. Las variables van precedidas por ?.

Se podría generalizar las 4 reglas de mover en una sola añadiendo en la BH conocimiento sobre los incrementos en la coordenas X e Y del principio y final de los vehículos y de las casillas que hay que liberar y ocupar. En todas las reglas de movimiento hay que modificar las coordenadas de un vehículo, ocupar una casilla y liberar otra. El programa en CLIPS completo sería

```
;;(casilla x y ocupada)
;;(vehiculo id orientacion xi yi xf yf)
(deffacts ini
  (casilla 1 1 si) (casilla 2 1 no) (casilla 3 1 si) (casilla 4 1 si) (casilla 5 1 si) (casilla 6 1 no)
  (casilla 1 2 si) (casilla 2 2 no) (casilla 3 2 no) (casilla 4 2 no) (casilla 5 2 si) (casilla 6 2 si)
  (casilla 1 3 si) (casilla 2 3 no) (casilla 3 3 no) (casilla 4 3 si) (casilla 5 3 no) (casilla 6 3 no)
  (casilla 1 4 si) (casilla 2 4 si) (casilla 3 4 si) (casilla 4 4 si) (casilla 5 4 no) (casilla 6 4 si)
  (casilla 1 5 si) (casilla 2 5 no) (casilla 3 5 no) (casilla 4 5 si) (casilla 5 5 no) (casilla 6 5 si)
  (casilla 1 6 si) (casilla 2 6 si) (casilla 3 6 no) (casilla 4 6 no) (casilla 5 6 no) (casilla 6 6 si)
  (vehiculo dorado v 1 1 1 2) ;;coche dorado
  (vehiculo verde h 3 1 5 1) ;;camion verde
  (vehiculo cian h 5 2 6 2) ;;coche cian
  (vehiculo morado v 1 3 1 5) ;;camion morado
  (vehiculo azul v 4 3 4 5) ;;camion azul
  (vehiculo ROJO h 2 4 3 4) ;;coche ROJO
  (vehiculo amarillo v 6 4 6 6) ;;camion amarillo
  (vehiculo verd2 h 1 6 2 6) ;;coche verde
)

(deffacts incrementos
     (mover h derecha 1 0 1 0) (mover h izquierda -1 0 -1 0)
     (ocupar derecha 1 0 1) (ocupar derecha 1 0 2)
     (ocupar izquierda -2 0 1) (ocupar izquierda -3 0 2)
     (liberar derecha 0 0 1) (liberar derecha 0 0 2)
     (liberar izquierda 1 0 1) (liberar izquierda 2 0 2)

     (mover v arriba 0 1 0 1) (mover v abajo 0 -1 0 -1)
     (ocupar arriba 0 1 1) (ocupar arriba 0 1 2)
     (ocupar abajo 0 -2 1) (ocupar abajo 0 -3 2)
     (liberar arriba 0 0 1) (liberar arriba 0 0 2)
     (liberar abajo 0 1 1) (liberar abajo 0 2 2)
)

(defrule fin
  (declare (salience 100))
```

```
  (vehiculo ROJO h 5 4 6 4)
=>
 (printout t "Conseguido " crlf)
 (halt)
)

(defrule mover
  ?veh <- (vehiculo ?id ?o ?xi ?yi ?xf ?yf)
  ?cas1 <- (casilla ?xc ?yc no) ;;casilla a ocupar
  ?cas2 <- (casilla ?xl ?yl si) ;;casilla a liberar
  (mover ?o ?m ?dxi ?dyi ?dxf ?dyf)
  (ocupar ?m  ?dox ?doy ?l)
  (liberar ?m ?dlx ?dly ?l)
  (test (= ?l (+ (- ?xf ?xi) (- ?yf ?yi)))) ;;longitud del vehiculo
  (test (= ?xc (+ ?xf ?dox)))
  (test (= ?yc (+ ?yf ?doy)))

  (test (= ?xl (+ ?xi ?dlx)))
  (test (= ?yl (+ ?yi ?dly)))
=>
 (retract ?veh ?cas1 ?cas2)
 (assert (vehiculo ?id ?o (+ ?xi ?dxi) (+ ?yi ?dyi) (+ ?xf ?dxf) (+ ?yf ?dyf)))
 (assert (casilla ?xc ?yc si))
 (assert (casilla ?xl ?yl no))
 (printout t "Mueve hacia " ?m " el vehiculo " ?id "  de longitud " ?l )
 (printout t ": ocupa " ?xc " " ?yc )
 (printout t "   libera " ?xl " " ?yl crlf)
 )
```

Solución Problema 4

1. Una heurística admisible es contar el número de coches que se interponen entre el coche rojo y la salida. Una mas informada sería, sumar a la cantidad anterior 1 por cada casilla que haya entre el coche rojo y la salida.

2. Los operadores de este problema son 4 Izquierda(X), Derecha(X), Arriba(X) y Abajo(X), que representan los posibles movimientos de un vehículo X. Si se establece el orden de generación de sucesores Derecha, Izquierda, Arriba, Abajo y se empiezan las instanciaciones con el coche rojo, el árbol que generaría el algoritmo en escalada sería:

(0,1) Inicial			
-	-	-	-
-	-	-	v1
-	R	-	v1
-	-	-	-

(1,2) Derecha(R)				(2,-) Izquierda(R)				(3,-) Arriba(v1)				(4,-) Abajo(v1)			
-	-	-	-	-	-	-	-	-	-	-	v1	-	-	-	-
-	-	-	v1	-	-	-	v1	-	-	-	v1	-	-	-	-
-	-	R	v1	R	-	-	v1	-	R	-	-	-	R	-	v1
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	v1
h=2				h=4				h=2				h=3			

(5,3) Arriba(v1)				(6,-) Abajo(v1)			
-	-	-	v1	-	-	-	-
-	-	-	v1	-	-	-	-
-	-	R	-	-	-	R	v1
-	-	-	-	-	-	-	v1
h=1				h=2			

(7,-) Derecha(R)			
-	-	-	v1
-	-	-	v1
-	-	-	R
-	-	-	-

3. No, el algoritmo en escalada es un algoritmo avaricioso que no guarda todos los estados y no puede garantizar optimalidad aunque la heurística sea admisible.