PRACTICA 2. BÚSQUEDA

Fecha de publicación: 2016/03/01
Fechas de entrega: [grupos jueves: miércoles 2017/03/22, 23:55]
[grupos viernes: jueves 2017/03/23, 23:55]
Planificación: semana 1: Ejercicios 1, 2, 3, 4, 5, 6
semana 2: Ejercicios 6, 7, 8, 9, 10
semana 3: Ejercicios 11, 12 + memoria Versión: 2017/02/27

IMPORTANTE: Utilizad la versión Allegro CL 6.2 ANSI with IDE y compilad el código.
Forma de entrega: Según lo dispuesto en las normas de entrega generales, publicadas en Moodle

- El código debe estar en un único fichero.
- El código debe estar en un único fichero. La evaluación del código en el fichero no debe dar errores en Allegro CL 6.2 ANSI with IDE (recuerda: CTRL+A CTRL+E debe evaluar todo el código sin errores).

Material recomendado:

- En cuanto a estilo de programación LISP: http://www.cs.cmu.edu/Groups/AI/html/faqs/lang/lisp/part1/faq.html
- Como referencia de funciones LISP: http://www.lispworks.com/documentation/common-lisp.html

---

Descripción general

El problema de 'pathfinder' ('descubridor-de-caminos') es común en juegos de ordenador en los cuales los personajes pueden moverse en un espacio para alcanzar la posición de un objetivo. Por ejemplo, encontrar la salida de un laberinto. En esta práctica se propone desarrollar un módulo IA de uno de estos juegos para un jugador automático. El jugador automático debe dirigir el vuelo de una nave espacial *Pathfinder* que, partiendo de un planeta de origen, tiene como meta alcanzar alguno de los planetas destino que se le encomiendan como objetivo de su misión. En cada movimiento la nave debe navegar por la galaxia de planeta en planeta a través de una red de agujeros blancos unidireccionales o de una red de agujeros de gusano bidireccionales. Utilizar los agujeros conlleva un consumo energético para la nave mayor que cero. El módulo IA de la nave debe usar algoritmos de búsqueda para encontrar el camino a uno de los planetas destino, idealmente con el menor consumo energético posible. En cada planeta se ha desplegado un sistema de sensores que proporcionan una estimación heurística del coste para llegar al planeta destino.

A continuación se presenta un ejemplo de galaxia: la galaxia Messier 35 (M35).
Definiremos como variables globales:

- **Planetas**: Lista constante de símbolos representando los nombres de los planetas
  `(setf *planets* '(Avalon Davion ...)`

- Los **agujeros blancos y agujeros de gusano**: Listas constantes de tripletes formados por **planeta origen, planeta destino y gasto energético** (coste)

```
(setf *white-holes*
  '((Avalon Mallory 2) (Avalon Proserpina 12)
    (Davion Proserpina 14)...)

 (setf *worm-holes*
  '((Avalon Kentares 4) (Avalon Mallory 7)
    (Davion Katril 1) (Davion Sirtis 8)
    (Katril Davion 1) (Katril Mallory 5) (Katril Sirtis 10)          (
    (Kentares Avalon 4) (Kentares Proserpina 21)...)
```

- Los **sensores** (*heurística*): Lista constante de pares formados por el planeta y el valor de la heurística

```
(setf *sensors*
  '((Avalon 5) (Davion 1)…)
```

- **Planeta origen**: El estado inicial será el correspondiente a estar en el planeta origen de la galaxia.

```
(setf *planet-origin* 'Kentares)
```

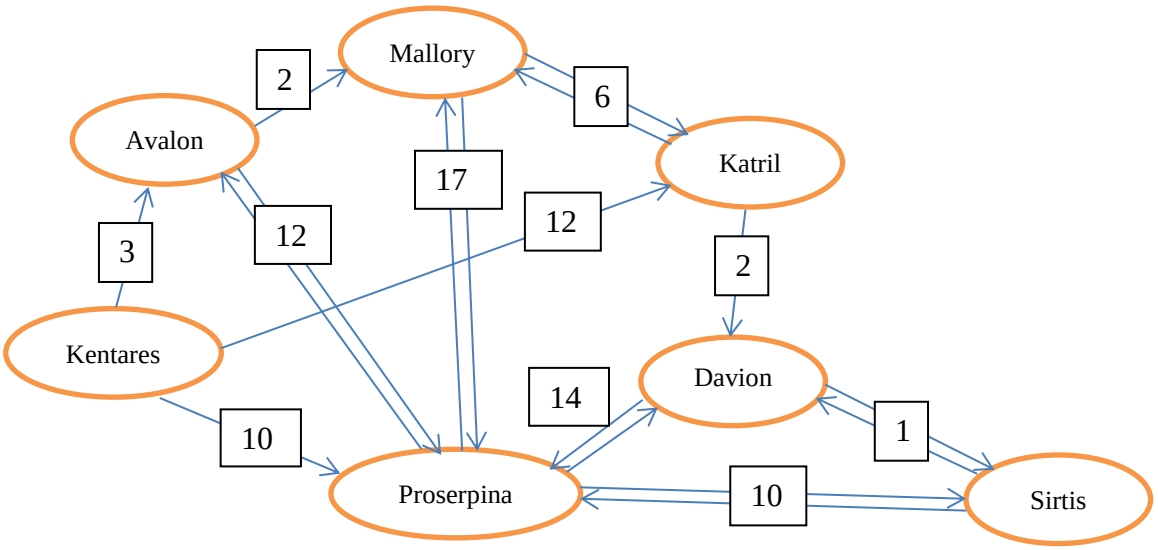
- **Planetas destino**: Lista constante de símbolos representando los nombres de los planetas destino.

```
(setf *planets-destination* '(Sirtis))
```

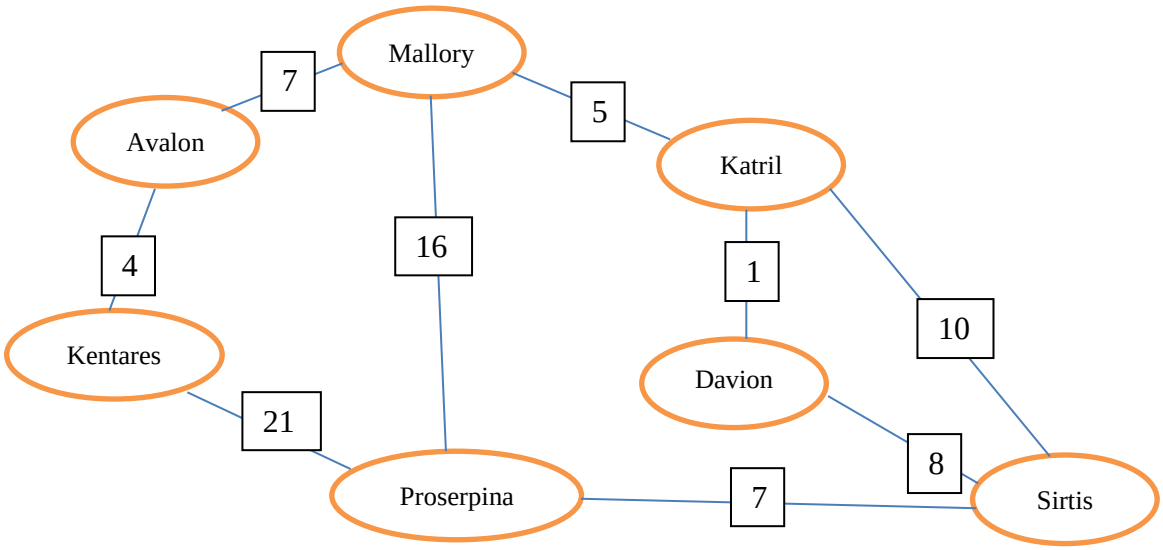
La tabla de la izquierda contiene los valores de los sensores en cada planeta (heurística) para llegar a Sirtis.

GALAXIA M35: Agujeros blancos (unidireccionales)

Planeta: n	h(n)
Avalon	5
Mallory	7
Kentares	4
Davion	1
Proserpina	4
Katril	3
Sirtis	0



GALAXIA M35: Agujeros de gusano (bidireccionales)



Estructuras

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;    Problem definition
;;
(defstruct problem
  states              ; List of states
  initial-state       ; Initial state
  f-goal-test         ; reference to a function that determines whether
                      ; a state fulfills the goal
  f-h                 ; reference to a function that evaluates to the
                      ; value of the heuristic of a state
  operators)          ; list of operators (references to functions)
                      ; to generate succesors
;;
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;    Node in search tree
;;
(defstruct node
     state         ; state label
     parent        ; parent node
     action        ; action that generated the current node from its parent
     (depth 0)     ; depth in the search tree
     (g 0)         ; cost of the path from the initial state to this node
     (h 0)         ; value of the heuristic
     (f 0))        ; g + h
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;    Actions
;;
(defstruct action
     name          ; Name of the operator that generated the action
     origin        ; State on which the action is applied
     final         ; State that results from the application of the action
     cost )        ; Cost of the action
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;    Search strategies
;;
(defstruct strategy
     name               ; Name of the search strategy
     node-compare-p)    ; boolean comparison
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Funciones específicas del problema

EJERCICIO 1 (5 %): Test para determinar si se ha alcanzado el objetivo.

Codifique una función que compruebe si se ha alcanzado el objetivo según el siguiente prototipo:

```
(defun f-goal-test-galaxy (state planets-destination) ...)

(f-goal-test-galaxy 'Sirtis *planets-destination*) ;-> T (o equivalente)
(f-goal-test-galaxy 'Avalon *planets-destination*) ;-> NIL
(f-goal-test-galaxy 'Urano  *planets-destination*) ;-> NIL
```

EJERCICIO 2 (5 %.): Evaluación del valor de la heurística.

Codifique una función que calcula el valor de la heurística en el estado actual:

```
(defun f-h-galaxy (state sensors)...)

(f-h-galaxy 'Sirtis *sensors*) ;-> 0
(f-h-galaxy 'Avalon *sensors*) ;-> 5
```

EJERCICIO 3 (20 %): Operadores `navigate-worm-hole` y `navigate-white-hole`.

Los operadores `navigate-worm-hole` y `navigate-white-hole` devuelven una lista de acciones posibles desde el estado actual. Tienen los siguientes prototipos:

```
(defun navigate-worm-hole (state worm-holes)...)

(defun navigate-white-hole (state white-holes) ...)

(navigate-worm-hole 'Katril *worm-holes*) ;->
;(#S(ACTION :NAME NAVIGATE-WORM-HOLE :ORIGIN KATRIL :FINAL DAVION :COST 1)
; #S(ACTION :NAME NAVIGATE-WORM-HOLE :ORIGIN KATRIL :FINAL MALLORY :COST 5)
; #S(ACTION :NAME NAVIGATE-WORM-HOLE :ORIGIN KATRIL :FINAL SIRTIS :COST 10))

(navigate-white-hole 'Urano *white-holes*) ;-> NIL
```

Formalización del problema

EJERCICIO 4 (5 %): Definir estrategia para la búsqueda A*.

Inicializar una variable global cuyo valor sea la estrategia para realizar la búsqueda A*:

```
(setf *A-star*
  (make-strategy ...))
```

Ejemplo: Estrategia para búsqueda de coste uniforme:

```
(setf *uniform-cost*
  (make-strategy
     :name 'uniform-cost
     :node-compare-p 'node-g-<=))

(defun node-g-<= (node-1 node-2)
  (<= (node-g node-1)
      (node-g node-2)))
```

EJERCICIO 5 (5%): Representación LISP del problema.

Inicialice el valor de una estructura denominada `*galaxy-M35*` que representa el problema bajo estudio.

```
(setf *galaxy-M35*
  (make-problem
   :states            *planets*
   :initial-state     *planet-origin*
   :f-goal-test       #'(lambda (state)
                          (f-goal-test-galaxy state *planets-destination*))
   :f-h               ...
   :operators         (list ...)))
```

EJERCICIO 6 (15 %): Expandir nodo.

Codifique la función de expansión de nodos, según el siguiente prototipo:

```
(defun expand-node (node problem)...)
```

El resultado debe ser una **lista de nodos.** Ejemplo:

```
(setf node-00
(make-node :state 'Proserpina :depth 12 :g 10 :f 20) )

(print
 (setf lst-nodes-00
   (expand-node node-00 *galaxy-M35*))) ;->
;;;
;;;(#S(NODE :STATE AVALON :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;        :ACTION #S(ACTION :NAME NAVIGATE-WHITE-HOLE :ORIGIN PROSERPINA :FINAL AVALON :COST 12) :DEPTH 13 :G 22 :H 5 :F 27)
;;; #S(NODE :STATE DAVION :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;        :ACTION #S(ACTION :NAME NAVIGATE-WHITE-HOLE :ORIGIN PROSERPINA :FINAL DAVION :COST 14) :DEPTH 13 :G 24 :H 1 :F 25)
;;; #S(NODE :STATE MALLORY :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;        :ACTION #S(ACTION :NAME NAVIGATE-WHITE-HOLE :ORIGIN PROSERPINA :FINAL MALLORY :COST 17) :DEPTH 13 :G 27 :H 7 :F 34)
;;; #S(NODE :STATE SIRTIS :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;        :ACTION #S(ACTION :NAME NAVIGATE-WHITE-HOLE :ORIGIN PROSERPINA :FINAL SIRTIS :COST 10) :DEPTH 13 :G 20 :H 0 :F 20)
;;; #S(NODE :STATE KENTARES :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;        :ACTION #S(ACTION :NAME NAVIGATE-WORM-HOLE :ORIGIN PROSERPINA :FINAL KENTARES :COST 21) :DEPTH 13 :G 31 :H 4 :F 35)
;;; #S(NODE :STATE MALLORY :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;        :ACTION #S(ACTION :NAME NAVIGATE-WORM-HOLE :ORIGIN PROSERPINA :FINAL MALLORY :COST 16) :DEPTH 13 :G 26 :H 7 :F 33)
;;; #S(NODE :STATE SIRTIS :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;        :ACTION #S(ACTION :NAME NAVIGATE-WORM-HOLE :ORIGIN PROSERPINA :FINAL SIRTIS :COST 7) :DEPTH 13 :G 17 :H 0 :F 17))
```

EJERCICIO 7 (10 %): Gestión de nodos.

Inserta una lista de nodos en otra lista de acuerdo con una estrategia:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  Insert a list of nodes into another list of nodes
;;;
;;;
(defun insert-nodes-strategy (nodes lst-nodes strategy)...)
```

Se supone que la lista `lst-nodes` está ordenada de acuerdo a dicha estrategia. La lista de nodos `nodes` no tiene por qué tener una ordenación especial. Ejemplo:

```
(setf node-01
   (make-node :state 'Avalon :depth 0 :g 0 :f 0) )
(setf node-02
   (make-node :state 'Kentares :depth 2 :g 50 :f 50) )
(print(insert-nodes-strategy (list node-00 node-01 node-02)
lst-nodes-00
                          *uniform-cost*));->

;;;
;;;(#S(NODE :STATE AVALON :PARENT NIL :ACTION NIL :DEPTH 0 :G 0 :H 0 :F 0)
;;; #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;; #S(NODE :STATE AVALON :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WHITE-HOLE :ORIGIN PROSERPINA :FINAL AVALON :COST 12) :DEPTH 13 :G 22 :H 5 :F 27)
;;; #S(NODE :STATE DAVION :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WHITE-HOLE :ORIGIN PROSERPINA :FINAL DAVION :COST 14) :DEPTH 13 :G 24 :H 1 :F 25)
;;; #S(NODE :STATE MALLORY :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WHITE-HOLE :ORIGIN PROSERPINA :FINAL MALLORY :COST 17) :DEPTH 13 :G 27 :H 7 :F 34)
;;; #S(NODE :STATE SIRTIS :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WHITE-HOLE :ORIGIN PROSERPINA :FINAL SIRTIS :COST 10) :DEPTH 13 :G 20 :H 0 :F 20)
;;; #S(NODE :STATE KENTARES :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WORM-HOLE :ORIGIN PROSERPINA :FINAL KENTARES :COST 21) :DEPTH 13 :G 31 :H 4 :F 35)
;;; #S(NODE :STATE MALLORY :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WORM-HOLE :ORIGIN PROSERPINA :FINAL MALLORY :COST 16) :DEPTH 13 :G 26 :H 7 :F 33)
;;; #S(NODE :STATE SIRTIS :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WORM-HOLE :ORIGIN PROSERPINA :FINAL SIRTIS :COST 7) :DEPTH 13 :G 17 :H 0 :F 17)
;;; #S(NODE :STATE KENTARES :PARENT NIL :ACTION NIL :DEPTH 2 :G 50 :H 0 :F 50))
```

```
(print (insert-nodes-strategy (list node-00 node-01 node-02)
                          (sort (copy-list lst-nodes-00) #'<= :key #'node-g)
                          *uniform-cost*));->

;;;
;;;(#S(NODE :STATE AVALON :PARENT NIL :ACTION NIL :DEPTH 0 :G 0 :H 0 :F 0)
;;; #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;; #S(NODE :STATE SIRTIS :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WORM-HOLE :ORIGIN PROSERPINA :FINAL SIRTIS :COST 7) :DEPTH 13 :G 17 :H 0 :F 17)
;;; #S(NODE :STATE SIRTIS :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WHITE-HOLE :ORIGIN PROSERPINA :FINAL SIRTIS :COST 10) :DEPTH 13 :G 20 :H 0 :F 20)
;;; #S(NODE :STATE AVALON :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WHITE-HOLE :ORIGIN PROSERPINA :FINAL AVALON :COST 12) :DEPTH 13 :G 22 :H 5 :F 27)
;;; #S(NODE :STATE DAVION :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WHITE-HOLE :ORIGIN PROSERPINA :FINAL DAVION :COST 14) :DEPTH 13 :G 24 :H 1 :F 25)
;;; #S(NODE :STATE MALLORY :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WORM-HOLE :ORIGIN PROSERPINA :FINAL MALLORY :COST 16) :DEPTH 13 :G 26 :H 7 :F 33)
;;; #S(NODE :STATE MALLORY :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WHITE-HOLE :ORIGIN PROSERPINA :FINAL MALLORY :COST 17) :DEPTH 13 :G 27 :H 7 :F 34)
;;; #S(NODE :STATE KENTARES :PARENT #S(NODE :STATE PROSERPINA :PARENT NIL :ACTION NIL :DEPTH 12 :G 10 :H 0 :F 20)
;;;         :ACTION #S(ACTION :NAME NAVIGATE-WORM-HOLE :ORIGIN PROSERPINA :FINAL KENTARES :COST 21) :DEPTH 13 :G 31 :H 4 :F 35)
;;; #S(NODE :STATE KENTARES :PARENT NIL :ACTION NIL :DEPTH 2 :G 50 :H 0 :F 50))
```

Búsquedas

EJERCICIO 8 (15 %): Función de búsqueda.

Codifique la función de búsqueda, según el siguiente pseudocódigo:

```
open: lista de nodos generados, pero no explorados
closed: lista de nodos generados y explorados
strategy: estrategia de búsqueda implementada como una ordenación de la
lista open-nodes
goal-test: test objetivo (predicado que evalúa a T si un nodo cumple la
condición de ser meta)

; Realiza la búsqueda para el problema dado utilizando una estrategia
; Evalúa:
;    Si no hay solución: NIL
;    Si hay solución: un nodo que cumple el test objetivo
;
(defun graph-search (problem strategy)
      Inicializar la lista de nodos open-nodes con el estado inicial
      inicializar la lista de nodos closed-nodes con la lista vacía
      recursión:
         o si la lista open-nodes está vacía, terminar[no se han encontrado
           solución]
         o extraer el primer nodo de la lista open-nodes
         o si dicho nodo cumple el test objetivo
                evaluar a la solución y terminar.
           en caso contrario
                si el nodo considerado no está en closed-nodes o, estando en
                dicha lista, tiene un coste g inferior al del que está en
                closed-nodes
                     * expandir el nodo e insertar los nodos generados en
                     la lista
                     open-nodes de acuerdo con la estrategia strategy.
                     * incluir el nodo recién expandido al comienzo de la
                     lista
                     closed-nodes.
         o Continuar la búsqueda eliminando el nodo considerado de la lista
           open-nodes.

Ejemplo:
(graph-search *galaxy-M35* *A-star*);->
;
; #S(NODE :STATE SIRTIS
;         :PARENT
;         #S(NODE :STATE ...
```

EJERCICIO 9 (5 %): búsqueda A*.

Implementar el algoritmo de búsqueda A*, de acuerdo con la siguiente cabecera:

```
; Realiza la búsqueda A* para el problema dado
; Evalúa:
;    Si no hay solución: NIL
;    Si hay solución: el nodo correspondiente al estado-objetivo
;
```

Ejemplo:

```
(a-star-search *galaxy-M35*);->
;
; #S(NODE :STATE SIRTIS
;         :PARENT
;         #S(NODE :STATE ...
```

EJERCICIO 10 (5 %): Ver camino.

Codifique la función que muestra el camino seguido para llegar a un nodo:

```
(defun tree-path (node)...)
```

Ejemplos:

```
(tree-path nil) ;-> NIL
(tree-path #S(NODE :STATE MALLORY ...)) ;-> (KENTARES PROSERPINA MALLORY)
```

El camino visualizado no tiene por qué ser el resultado de una búsqueda.

EJERCICIO 11 (5 %): Secuencia de acciones de un camino.

Codifique la función que muestra la secuencia de acciones para llegar a un nodo:

```
(defun action-sequence (node)...)
```

Ejemplo:

```
(action-sequence (a-star-search *galaxy-M35*)) ;->
;;;(#S(ACTION :NAME NAVIGATE-WHITE-HOLE :ORIGIN KENTARES :FINAL AVALON :COST 3)…)
```

EJERCICIO 12 (5 %): Otras estrategias de búsqueda.

Diseñe una estrategia para realizar búsqueda en profundidad:

```
(setf *depth-first*
  (make-strategy
     :name 'depth-first
     :node-compare-p 'depth-first-node-compare-p))

(defun depth-first-node-compare-p (node-1 node-2)
     <codigo LISP para depth-first>)

(tree-path (graph-search *galaxy-M35* *depth-first*))
```

Diseñe una estrategia para realizar búsqueda en anchura:

```
(setf *breadth-first*
  (make-strategy
     :name 'breadth-first
     :node-compare-p 'breadth-first-node-compare-p))

(defun breadth-first-node-compare-p (node-1 node-2)
     <codigo LISP para breadth-first>)

(tree-path (graph-search *galaxy-M35* *breadth-first*))
```

Protocolo de corrección

[40%] Corrección automática

La corrección de la práctica se realizará utilizando distintos problemas de búsqueda y distintas "galaxias" diferentes de la del ejemplo del enunciado, por lo que se recomienda definir una batería de pruebas con distintos problemas.

[30%] Estilo

[30%] Memoria

En la memoria se debe incluir respuestas a las siguientes preguntas

1. ¿Por qué se ha realizado este diseño para resolver el problema de búsqueda? En concreto,
    1.1 ¿qué ventajas aporta?
    1. 2 ¿Por qué se han utilizado funciones lambda para especificar el test objetivo, la heurística y los operadores del problema?
2. Sabiendo que en cada nodo de búsqueda hay un campo “parent”, que proporciona una referencia al nodo a partir del cual se ha generado el actual ¿es eficiente el uso de memoria?
3. ¿Cuál es la complejidad espacial del algoritmo implementado?
4. ¿Cuál es la complejidad temporal del algoritmo?
5. Indicad qué partes del código se modificarían para limitar el número de veces que que se puede utilizar la acción “navegar por agujeros de gusano” (bidireccionales).