Práctica 5
Inteligencia Artificial

David Morales Sáez

1. Comentar el código (línea a línea) de la función general-search.

```
(defun general-search (problem queuing-fn)
"Expand nodes according to the specification of PROBLEM until we find a
solution or run out of nodes to expand. The QUEUING-FN decides which nodes to
look at first. [p 73]"
        (let ((nodes (make-initial-queue problem queuing-fn))
                ; Inicializamos la variable 'nodes' con el problema actual definido
                node
                total)
                 ; Crea la variable 'node'
        (setq total 0)
        (loop (if (empty-queue? nodes) (RETURN nil))
                ; Hace un bucle que finalizar· cuando no queden nodos por
comprobar
                (setq node (remove-front nodes))
                 ; Guarda en la variable 'node' el primer nodo de 'nodes'
                (setq total (+ total 1))
                (if (goal-test problem (node-state node)) (RETURN node))
                 ; Comprueba si se da el caso final con el nodo actual, devuelve el
nodo
                (funcall queuing-fn nodes (expand node problem)))
        node))
        ; en caso contrario, carga en nodes el siguiente nodo a comprobar.
```

2. Modificar la función general-search para que vaya mostrando por pantalla el número de nodos visitados.

```
(defun general-search (problem queuing-fn)
"Expand nodes according to the specification of PROBLEM until we find a
solution or run out of nodes to expand. The QUEUING-FN decides which nodes to
look at first. [p 73]"
        (let ((nodes (make-initial-queue problem queuing-fn))
                ; Inicializamos la variable 'nodes' con el problema actual definido
                node
                total)
                 ; Crea la variable 'node'
        (setq total 0)
        (loop (if (empty-queue? nodes) (RETURN nil))
                ; Hace un bucle que finalizar· cuando no queden nodos por
comprobar
                (setq node (remove-front nodes))
                 ; Guarda en la variable 'node' el primer nodo de 'nodes'
                (setq total (+ total 1))
                (if (goal-test problem (node-state node)) (RETURN node))
                 ; Comprueba si se da el caso final con el nodo actual, devuelve el
nodo
                (funcall queuing-fn nodes (expand node problem)))
```

```
            (print total)
            node))
            ; en caso contrario, carga en nodes el siguiente nodo a comprobar.
```

3. Para el problema de los caníbales comparar los resultados de las búsquedas de primero en anchura y primero en profundidad para los siguientes estados iniciales:

2 misioneros, 2 caníbales, 1 bote

```
(setq prob-canib (make-cannibal-problem :initial-state (make-cannibal-state :m1 2
:c1 2 :b1 1)))
#<a CANNIBAL-PROBLEM>

            (setq result (breadth-first-search prob-canib))

            131
            131
```

3 misioneros, 3 caníbales, 1 bote

```
(setq prob-canib2 (make-cannibal-problem :initial-state (make-cannibal-state :m1
3 :c1 3 :b1 1)))
#<a CANNIBAL-PROBLEM>

            (setq result (breadth-first-search prob-canib2))
            11878
            11878
```

3 misioneros, 3 caníbales, 2 botes

```
(setq prob-canib3 (make-cannibal-problem :initial-state (make-cannibal-state :m1
3 :c1 3 :b1 2)))
#<a CANNIBAL-PROBLEM>

            (setq result (breadth-first-search prob-canib3))

            917
            917
```

4. Para el problema de las ciudades de Rumanía (ver Ilustración 1) comparar los resultados de primero en anchura y primero en profundidad para una ruta que termine en Bucharest, otra que termine en Pitesti, otra que termine en Craiova y otra que termine en Fagaras. La ciudad de inicio de cada ruta la elige el alumno.

Desde Timisoara

a: Bucharest

```
(setq prob1 (make-romanian-problem :initial-state 'Timisoara :goal 'Bucharest))
```

1: Anchura
32
(#<NODE f(317) = g(0) + h(317) state:TIMISOARA> #<NODE f(468) = g(118) + h(350) state:ARAD> #<NODE f(491) = g(258) + h(233) state:SIBIU> #<NODE f(512) = g(357) + h(155) state:FAGARAS> #<NODE f(568) = g(568) + h(0) state:BUCHAREST>)

2: Profundidad
Bucle infinito entre Arad y Zerind.

b: Pitesti

(setq prob2 (make-romanian-problem :initial-state 'Timisoara :goal 'Pitesti))

1: Anchura
34
(#<NODE f(230) = g(0) + h(230) state:TIMISOARA> #<NODE f(378) = g(118) + h(260) state:ARAD> #<NODE f(402) = g(258) + h(144) state:SIBIU> #<NODE f(435) = g(338) + h(97) state:RIMNICU> #<NODE f(435) = g(435) + h(0) state:PITESTI>)

2: Profundidad
Bucle infinito entre Arad y Zerind.

c: Craiova

(setq prob3 (make-romanian-problem :initial-state 'Timisoara :goal 'Craiova))

1: Anchura
35
(#<NODE f(200) = g(0) + h(200) state:TIMISOARA> #<NODE f(378) = g(118) + h(260) state:ARAD> #<NODE f(433) = g(258) + h(175) state:SIBIU> #<NODE f(462) = g(338) + h(124) state:RIMNICU> #<NODE f(484) = g(484) + h(0) state:CRAIOVA>)

2: Profundidad
Bucle infinito entre Arad y Zerind.

d: Fagaras

(setq prob4 (make-romanian-problem :initial-state 'Timisoara :goal 'Fagaras))

1: Anchura
13
(#<NODE f(215) = g(0) + h(215) state:TIMISOARA> #<NODE f(336) = g(118) + h(218) state:ARAD> #<NODE f(356) = g(258) + h(98) state:SIBIU> #<NODE f(357) = g(357) + h(0) state:FAGARAS>)

2: Profundidad
Bucle infinito entre Arad y Zerind.