Universitat Oberta de Catalunya

uoc.edu

Intel·ligència Artificial

PAC1- Representació de problemes

Luis Enrique Arribas Zapater 17745245D

Parte 1 (40% puntuació)

El camino que elegirá el algoritmo será: A - B - F - H - Z. Puede comprobarse que es el de menor coste de los 5 posibles con un coste de 41.

Secuencia de ejecución del algoritmo A*

PASO	ACCION	COLA NODOS A EXPANDIR	LISTA NODOS EXPANDIDOS	CAMINO
0	Inicio	(A)	LISTA VACIA	A
1	Se añaden todos los hijos de A a la cola de nodos a expandir. El algoritmo ordena la cola en función del coste f(n) situando primero el de menor coste	(C D E B)	(A)	A
2	Se añaden todos los hijos de C a la cola de nodos a expandir. El algoritmo ordena la cola en función del coste f(n) situando primero el de menor coste	(D F E B)	(A C)	A-C
3	Se actualiza el valor de F, único hijo de D, en la cola de nodos a expandir. El algoritmo ordena la cola en función del coste f(n) situando primero el de menor coste. Se recalcula el coste de F (f(n)=70) y su atributo nodo-padre cambia de C a D	(F E B)	(A C D)	A-D
4	Se añaden los 2 hijos de F a la cola de nodos a expandir. El algoritmo ordena la cola en función del coste f(n) situando primero el de menor coste	(E B H G)	(A C D F)	A- D- F
5	Se expande E. Actualizamos g(H)=45 y g(F)=18 y nodo-padre(F)=E	(B H G)	(A C D F E)	A- E - F
6	Se expande B. g(F)=9, por tanto F(H)=89 y F(G)=94	(H G)	(A C D F E B)	A -B - F
7	Se añaden el hijo de H a la cola de nodos a expandir.	(Z G)	(A C D F E B H)	A-B-F-H
8	Se visita Z. En este punto el algoritmo ha encontrado solución pero continua su ejecución porque tiene nodos en la lista a expandir y ambas son condiciones excluyentes para terminar el algoritmo.	(G)	(A C D F E B H Z)	A-B-F-H-Z
9	Se expande G. No tiene hijos. Al haber solución y no quedar nodos en la cola con prioridad el algoritmo termina.		(A C D F E B H Z G)	A-B-F-H-Z
	En la página siguiente podemos ver el estado de la cola de nodos sin expandir a cada paso			

Paso 1

	g(n)		h(n)	f(n)
	Coste +	Coste acumulado		
C	10	0	60	70
D	10	0	65	75
E	15	0	70	85
B	4	0	90	94

Paso 2

	g(n)		h(n)	f(n)
	Coste +	Coste acumulado		
D	10	0	65	75
F	20	10	50	80
E	15	0	70	85
B	4	0	90	94

Paso 3

	g(n)		h(n)	f(n)
	Coste +	Coste acumulado		
F	10	10	50	70
E	15	0	70	85
B	4	0	90	94

Paso 4

	g(n)		h(n)	f(n)
	Coste +	Coste acumulado		
E	15	0	70	85
B	4	0	90	94
H	30	20	50	100
G	35	20	50	105

Paso 5

	g(n)		h(n)	f(n)
	Coste +	Coste acumulado		
B	4	0	90	94
H	30	15	50	95
G	35	18	50	103

Paso 6

	g(n)		h(n)	f(n)
	Coste +	Coste acumulado		
H	30	9	50	89
G	35	9	50	94

Estado de la cola con prioridad en cada paso:

Paso 7

	g(n)		h(n)	f(n)
	Coste +	Coste acumulado		
Z	2	39	50	91
G	35	9	50	94

Paso 8

	g(n)		h(n)	f(n)
	Coste +	Coste acumulado		
G	35	9	50	94

Parte 2 (4 x 15% puntuació)

a)

```
(defun heuristica (estat)
    (cond((equal estat 'A) 80)
         ((equal estat 'B) 90)
         ((equal estat 'C) 60)
         ((equal estat 'D) 65)
         ((equal estat 'E) 70)
         ((equal estat 'F) 50)
         ((equal estat 'G) 50)
         ((equal estat 'H) 50)
         ((equal estat 'Z) 0)
    (t 1000)))
```

b)

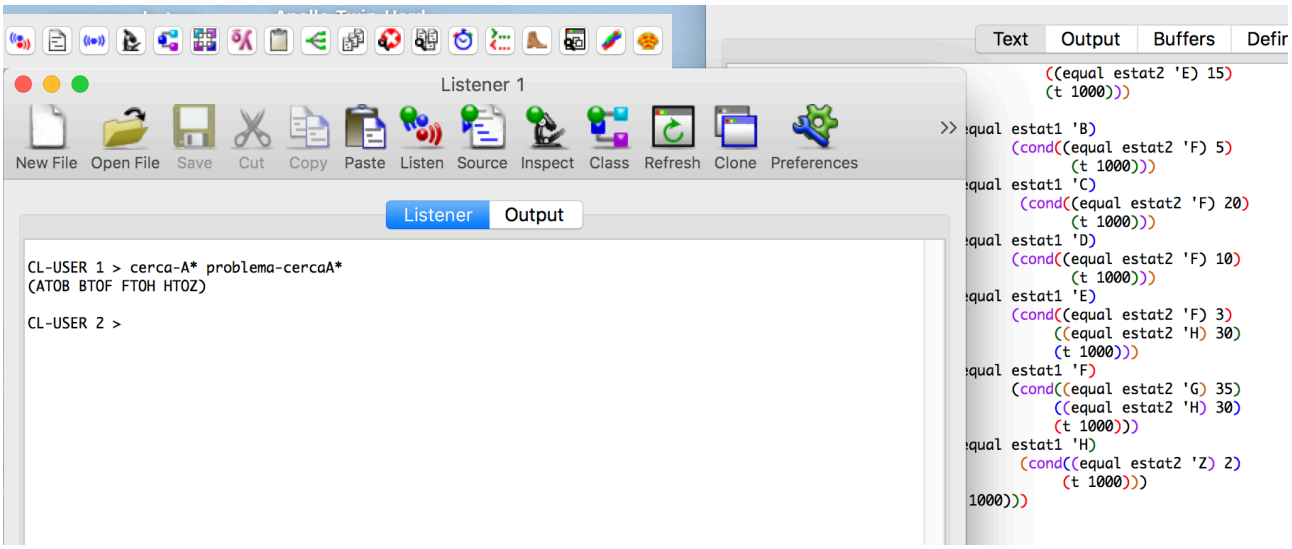
```
(defun cost (estat1 estat2)
            (cond((equal estat1 'A)
                        (cond((equal estat2 'B) 4)
                             ((equal estat2 'C) 10)
                             ((equal estat2 'D) 10)
                             ((equal estat2 'E) 15)
                             (t 1000)))

                  ((equal estat1 'B)
                          (cond((equal estat2 'F) 5)
                               (t nil)))
                  ((equal estat1 'C)
                           (cond((equal estat2 'F) 20)
                               (t 1000)))
                  ((equal estat1 'D)
                          (cond((equal estat2 'F) 10)
                               (t 1000)))
                  ((equal estat1 'E)
                          (cond((equal estat2 'F) 3)
                               ((equal estat2 'H) 30)
                               (t 1000)))
                  ((equal estat1 'F)
                          (cond((equal estat2 'G) 35)
                               ((equal estat2 'H) 30)
                               (t 1000)))
                  ((equal estat1 'H)
```

```
                    (cond((equal estat2 'Z) 2)
                         (t 1000)))
               (t 1000)))
```

c)

Se cambia la condición (t nil) por (t 1000) para evitar que la execución devuelva error al esperar un número y recibir un valor nulo.



d)

A continuación vemos los cambios realizados en el código:

```
(defun AtoB (estat info)
  (if (equal estat 'A) 'B 'buit))
(defun AtoC (estat info)
  (if (equal estat 'A) 'C 'buit))
(defun AtoD (estat info)
  (if (equal estat 'A) 'D 'buit))
(defun BtoD (estat info)
  (if (equal estat 'B) 'D 'buit))
(defun BtoG (estat info)
  (if (equal estat 'B) 'G 'buit))
(defun CtoB (estat info)
```

```
  (if (equal estat 'C) 'B 'buit))
(defun CtoE (estat info)
  (if (equal estat 'C) 'E 'buit))
(defun DtoE (estat info)
  (if (equal estat 'D) 'E 'buit))
(defun EtoH (estat info)
  (if (equal estat 'E) 'H 'buit))
(defun EtoF (estat info)
  (if (equal estat 'E) 'F 'buit))
(defun FtoG (estat info)
  (if (equal estat 'F) 'G 'buit))
(defun GtoH (estat info)
  (if (equal estat 'G) 'H 'buit))
(defun HtoI (estat info)
  (if (equal estat 'H) 'I 'buit))



(defvar tl-operadors
  (list (list 'AtoB #'AtoB)
        (list 'AtoC #'AtoC)
        (list 'AtoD #'AtoD)
        (list 'BtoD #'BtoD)
        (list 'BtoG #'BtoG)
        (list 'CtoB #'CtoB)
        (list 'CtoE #'CtoE)
        (list 'DtoE #'DtoE)
        (list 'EtoH #'EtoH)
        (list 'EtoF #'EtoF)
        (list 'FtoG #'FtoG)
        (list 'GtoZ #'Gtoz)
        (list 'HtoI #'HtoI)))
```

```
(defun heuristica (estat)
          (cond((equal estat 'A) 15)
               ((equal estat 'B) 2)
               ((equal estat 'C) 15)
               ((equal estat 'D) 18)
               ((equal estat 'E) 7)
               ((equal estat 'F) 5)
               ((equal estat 'G) 0)
               ((equal estat 'H) 9)
               ((equal estat 'I) 2)
          (t 1000)))


(defun cost (estat1 estat2)
                  (cond((equal estat1 'A)
                              (cond((equal estat2 'B) 20)
                                   ((equal estat2 'C) 20)
                                   ((equal estat2 'D) 1)
                                   (t 1000)))

                        ((equal estat1 'B)
                                (cond((equal estat2 'D) 4)
                                     ((equal estat2 'G) 4)
                                  (t 1000)))
                        ((equal estat1 'C)
                                 (cond((equal estat2 'B) 3)
                                      ((equal estat2 'E) 3)
                                  (t 1000)))
                        ((equal estat1 'D)
                                (cond((equal estat2 'E) 5)
                                  (t 1000)))
```

```
((equal estat1 'E)
        (cond((equal estat2 'F) 9)
             ((equal estat2 'H) 2)
             (t 1000)))
((equal estat1 'F)
        (cond((equal estat2 'G) 10)
             (t 1000)))
((equal estat1 'G)
        (cond((equal estat2 'H) 4)
             (t 1000)))
((equal estat1 'H)
         (cond((equal estat2 'I) 1)
              (t 1000)))
(t 1000)))
```

Y aquí vemos el resultado de la ejecución **(ATOB BTOG)** que no es el resultado óptimo, ya que podría recorrerse A-D-E-H-I-C-B-G con un coste de 23 mientras que el elegido por el algoritmo es de 24.