**MEMORIA PRÁCTICA 2**

**Autores: Victoria Pelayo e Ignacio Rabuñal**

**Grupo: 2301 pareja 8**

**Ejercicio1:**

En este ejercicio nos piden implementar dos funciones para evaluar el valor de la heurística, según el tiempo o el precio, de un estado.

**Pseudocódigo de las funciones:**

**F-H-TIME (state sensors)**

input:

state: nombre de la ciudad (estado) actual

sensors: lista de los estados y sus correspondientes heurísticas

los elementos de la lista son (state (h-time h-price))

Output:

Res: el valor de la heurística del tiempo

Proceso:

i = 0

para i < longitud de sensors:

si state = valor-estado(sensors[i]):

devolver valor-heurisitica-time (sensors[i])

**F-H-PRICE (state, sensors)**

input:

state: nombre de la ciudad (estado) actual

sensors: lista de los estados y sus correspondientes heurísticas

los elementos de la lista son (state (h-time h-price))

Output:

Res: el valor de la heurística del precio

Proceso:

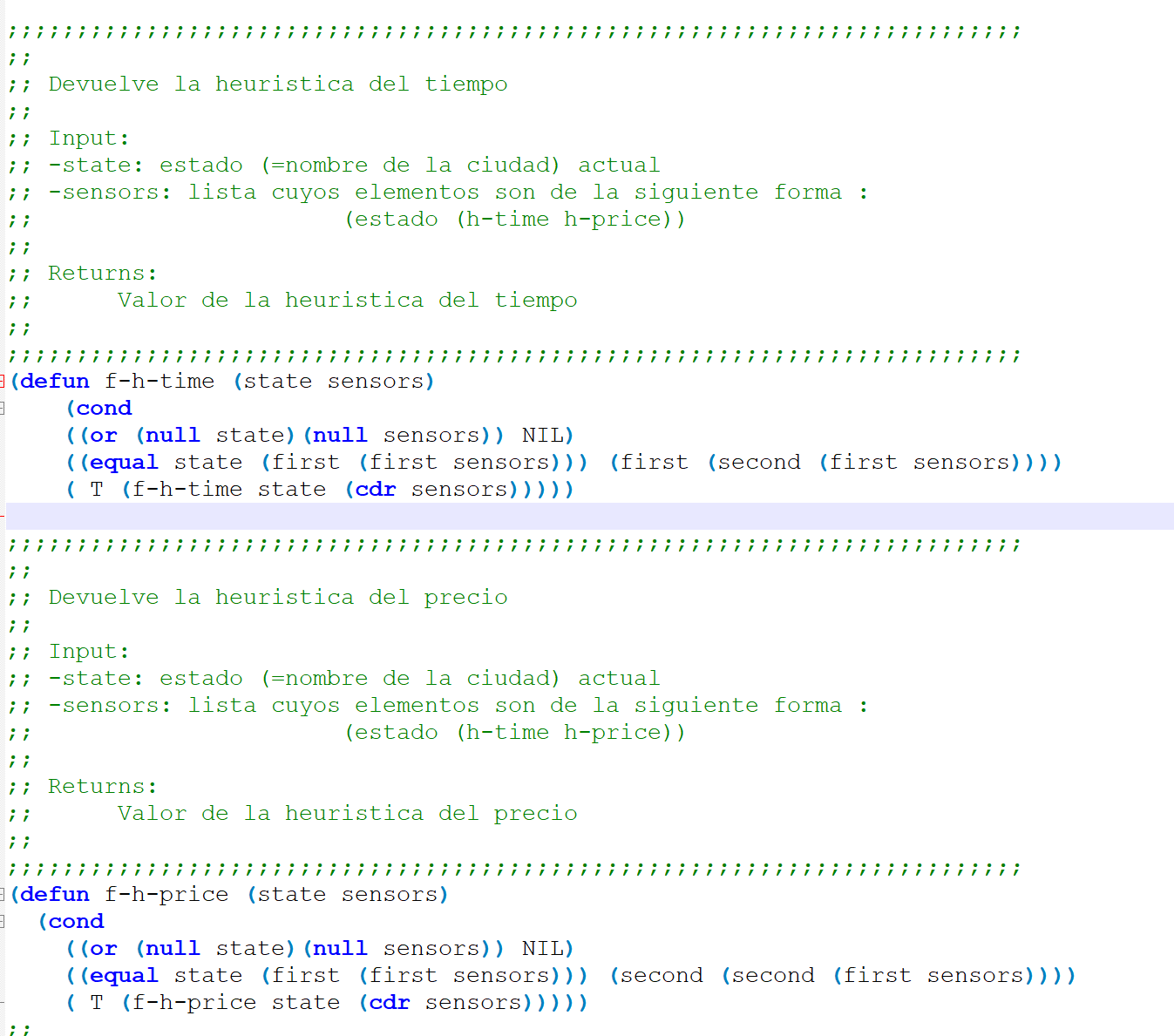
i = 0

para i < longitud de sensors:

si state = valor-estado(sensors[i]):

devolver valor-heurisitica-price (sensors[i])

Capturas del Código implementado:



**Ejemplos de ejecución ejercicio 1:**

**CG-USER**(21): (f-h-time 'Nantes \*estimate\*)

75.0

**CG-USER**(22): (f-h-time 'Marseille \*estimate\*)

145.0

**CG-USER**(35): (f-h-price 'Lyon \*estimate\*)

0.0

**CG-USER**(36): (f-h-price 'Madrid \*estimate\*)

NIL

**CG-USER**(37): (f-h-time 'Madrid \*estimate\*)

NIL

**Ejercicio2:**

En este ejercicio tenemos que implementar 4 funciones(navigate-canal-time, navigate-canal-price, navigate-train-time, navigate-train-price) que devolverán una lista de acciones que pueden realizar dado un estado actual. Por ejemplo, navigate-canal-time nos devolverá la lista de acciones posibles, partiendo de un estado, utilizando canales y tomará como heurística el tiempo.

Nos piden crear una función auxiliar navigate más genérica que recibe una función (“cfun”, una de las mencionadas anteriormente) y se encargará de devolver la lista de acciones posibles en función de la “cfun” pasada. Es importante la existencia de esta función, pues, evita repetir código.

Para este ejercicio también hemos creado otras tres funciones auxiliares: navigate-aux, que es la función recursiva de navigate; ciudad-permitida, que devuleve si una ciudad está permitida en el camino; get-time, que devuelve el coste del tiempo de un elemento determinado; get-price, que devuelve el coste del precio de un elemento determinado.

**Pseudocódigos de las funciones:**

**Ciudad-permitida(state, forbidden)**

Input: state (estado actual), forbidden (lista de estados prohibidos)

Output: T si está permitida y F si no es así

Proceso:

Si estado = null:

Devolver False

Si prohibidas = null:

Devolver False

Si estado = forbidden[0]:

Devolver True

Si no:

Devolver ciudad-permitida (state, resto de forbidden)

\*resto de forbidden = lista de forbidden a partir del elemento forbidden[1]

**Get-time (lst)**

Input: lst (lista de elementos de la forma (origin final (time price)) )

Output: Res(tiempo del primer elemento de la tabla)

Proceso:

Si lst = null:

Devolver null

Si no:

Devolver lst[0][2][0] (primer elemento del tercer elemento del primer elemento de la lista)

**Get-price (lst)**

Input: lst (lista de elementos de la forma (origin final (time price)) )

Output: Res(precio del primer elemento de la tabla)

Proceso:

Si lst = null:

Devolver null

Si no:

Devolver lst[1][2][0] (segundo elemento del tercer elemento del primer elemento de la lista)

**Navigate(state lst-edges cfun name(opcional) forbidden)**

Input: state(estado actual), lst-edges(lista de los nodos del grafo), cfun(nombre de la función para saber que coste usar), nombre (es opcional, nombre que le pondremos a la acción), forbidden (lista de los estados no permitidos)

Output: Res lista de acciones originadas a partir del estado dado

Proceso:

Res = navigate-aux(state, lst-edges, name, forbidden, () )

**Navigate-aux(state lst-edges cfun name(opcional) forbidden lst)**

Input: state(estado actual), lst-edges(lista de nodos del grafo), cfun(función para saber que coste obtener), name(combre de la acción), forbidden(lista de estados no permitidos), lst(lista de acciones)

Output: lst(lista de acciones realizadas a partir del estado)

Proceso:

Si (estado = null OR lst-edges = null):

Devolver null

Si (estado = lst-edges[0][0] AND True = ciudad-permitida(lst-edges[0][1], forbidden)):

Acción = nueva acción{

Name = name

Origin = state

Final = lst-edges[0][1]

Cost = cfun(lst-edges)

}

Añadir-a-lst(accion)

Añadir-a-lst(navigate-aux(state lst-edges cfun name forbidden lst))

Devolver lst

Si no:

Añadir-a-lst(navigate-aux(state lst-edges cfun name forbidden lst))

Devolver lst

En esta función de manera recursiva en la lista de grafos comprobamos si el grafo corresponde su estado actual al nuestro y si a donde lleva es una ciudad permitida, si esto es así añadimos una acción que creamos con estas características a nuestra lista de acciones y el resultado de aplicar esta función al resto de nodos de la lista. Si el primero no coincide devolvemos la lista tras añadirle el resultado de aplicar esta función al resto de la lista de nodos.

**Navigate-canal-time(state canals)**

Input: state(estado actual), canals(lista de canales)

Output: Res(lista de acciones al usar canales a partir del estado actual indicando el coste del tiempo)

Proceso:

Deolver navigate (state, canals, función get-time, “navigate-canal-time”, null)

**Navigate-canal-price(state canals)**

Input: state(estado actual), canals(lista de canales)

Output: Res(lista de acciones al usar canales a partir del estado actual indicando el coste del precio)

Proceso:

Deolver navigate (state, canals, función get-price, “navigate-canal-price”, null)

**Navigate-trains-price(state canals)**

Input: state(estado actual), trains(lista de caminos por trenes)

Output: Res(lista de acciones al usar canales a partir del estado actual indicando el coste del precio)

Proceso:

Deolver navigate (state, canals, función get-price, “navigate-train-price”, \*forbidden\*)

\*forbidden\* = lista de ciudades prohibidas por tren

**Navigate-trains-timee(state canals)**

Input: state(estado actual), trains(lista de caminos por trenes)

Output: Res(lista de acciones al usar canales a partir del estado actual indicando el coste del tiempo)

Proceso:

Deolver navigate (state, canals, función get-price, “navigate-train-price”, \*forbidden\*)

**Ejemplos de ejecución ejercicio 2:**

**CG-USER**(139): (navigate-canal-time 'Avignon \*canals\*)

(#S(ACTION :NAME NAVIGATE-CANAL-TIME :ORIGIN AVIGNON :FINAL MARSEILLE :COST 35.0))

**CG-USER**(140): (navigate-train-price 'Avignon \*trains\* '())

(#S(ACTION :NAME NAVIGATE-TRAIN-PRICE :ORIGIN AVIGNON :FINAL LYON :COST 40.0)

#S(ACTION :NAME NAVIGATE-TRAIN-PRICE :ORIGIN AVIGNON :FINAL MARSEILLE :COST 25.0))

**CG-USER**(141): (navigate-train-price 'Avignon \*trains\* '(Marseille))

(#S(ACTION :NAME NAVIGATE-TRAIN-PRICE :ORIGIN AVIGNON :FINAL LYON :COST 40.0))

**CG-USER**(142): (navigate-canal-time 'Orleans \*canals\*)

NIL

**CG-USER**(143): (navigate-canal-time 'Lyon \*canals\*)

(#S(ACTION :NAME NAVIGATE-CANAL-TIME :ORIGIN LYON :FINAL NANCY :COST 150.0) #S(ACTION :NAME NAVIGATE-CANAL-TIME :ORIGIN LYON :FINAL ROENNE :COST 40.0)

#S(ACTION :NAME NAVIGATE-CANAL-TIME :ORIGIN LYON :FINAL AVIGNON :COST 50.0))

**CG-USER**(144): (navigate-train-time 'Lyon \*trains\* \*forbidden\*)

(#S(ACTION :NAME NAVIGATE-TRAIN-TIME :ORIGIN LYON :FINAL TOULOUSE :COST 60.0) #S(ACTION :NAME NAVIGATE-TRAIN-TIME :ORIGIN LYON :FINAL ROENNE :COST 18.0))

**CG-USER**(145): (navigate-train-price 'Lyon \*trains\* \*forbidden\*)

(#S(ACTION :NAME NAVIGATE-TRAIN-PRICE :ORIGIN LYON :FINAL TOULOUSE :COST 95.0) #S(ACTION :NAME NAVIGATE-TRAIN-PRICE :ORIGIN LYON :FINAL ROENNE :COST 25.0))

**Ejemplos de ejecución ejercicio 3**

**CG-USER**(104): (f-goal-test node-calais '(Calais Marseille) '(Paris Nancy))

T

**CG-USER**(105): (f-goal-test node-calais '(Calais Marseille) '(Paris Limoges))

NIL

**CG-USER**(106): (f-goal-test node-paris '(Calais Marseille) '(Paris))

NIL

**CG-USER**(107): (f-goal-test node-calais '(Calais Marseille) '(Paris Nancy))

T

**CG-USER**(108): (f-goal-test nil '(hola hola) '(ninguno))

NIL

\*Como decision hemos tomado que node-nevers no puede ser un nodo de estado final, pues representa que no ha habido ningún camino

**CG-USER**(109): (f-goal-test node-nevers nil nil)

NIL

\*\*function auxiliar

**CG-USER**(110): (get-camino node-calais nil)

(CALAIS REIMS NANCY PARIS)

**CG-USER**(111): (get-camino node-nevers nil)

NIL

**CG-USER**(116): (get-camino node-paris '(madrid barcelona))

(PARIS MADRID BARCELONA)5

**Ejemplos de ejecución ejercicio 4:**

**CG-USER**(144): (f-search-state-equal node-calais node-calais-2 '())

T

**CG-USER**(145): (f-search-state-equal node-calais node-calais-2 '(Reims))

NIL

**CG-USER**(146): (f-search-state-equal node-calais node-calais-2 '(Nevers))

T

**CG-USER**(147): (f-search-state-equal node-nancy node-paris '())

NIL

\*\*Ejemplos de la funcion auxilair

**CG-USER**(148): (no-visitados nil '(madrid) nil)

(MADRID)

**CG-USER**(149): (no-visitados '(madrid barcelona) '(madrid paris lyon) nil)

(PARIS LYON)

**CG-USER**(150): (no-visitados '(madrid) nil nil)

NIL

**Ejemplos de ejecución ejercicio 6**

\*\*Ejemplo de la función auxiliar expand-node-action

(Con este ejemplo comprobamos que esta función funcione bien)

Hemos ido pidiendo cada uno de los atributos de nodo porque en la terminal no salía el nodo al completo

**CG-USER**(95): (node-f (expand-node-action node-marseille-ex6 #S(ACTION :NAME NAVIGATE-TRAIN-TIME :ORIGIN MARSEILLE :FINAL TOULOUSE :COST 65.0) \*travel-fast\*))

205.0

**CG-USER**(96): (node-g (expand-node-action node-marseille-ex6 #S(ACTION :NAME NAVIGATE-TRAIN-TIME :ORIGIN MARSEILLE :FINAL TOULOUSE :COST 65.0) \*travel-fast\*))

75.0

**CG-USER**(97): (node-h (expand-node-action node-marseille-ex6 #S(ACTION :NAME NAVIGATE-TRAIN-TIME :ORIGIN MARSEILLE :FINAL TOULOUSE :COST 65.0) \*travel-fast\*))

130.0

**CG-USER**(98): (node-depth (expand-node-action node-marseille-ex6 #S(ACTION :NAME NAVIGATE-TRAIN-TIME :ORIGIN MARSEILLE :FINAL TOULOUSE :COST 65.0) \*travel-fast\*))

13

**CG-USER**(99): (node-action (expand-node-action node-marseille-ex6 #S(ACTION :NAME NAVIGATE-TRAIN-TIME :ORIGIN MARSEILLE :FINAL TOULOUSE :COST 65.0) \*travel-fast\*))

#S(ACTION :NAME NAVIGATE-TRAIN-TIME :ORIGIN MARSEILLE :FINAL TOULOUSE :COST 65.0)

**CG-USER**(100): (node-parent (expand-node-action node-marseille-ex6 #S(ACTION :NAME NAVIGATE-TRAIN-TIME :ORIGIN MARSEILLE :FINAL TOULOUSE :COST 65.0) \*travel-fast\*))

#S(NODE :STATE MARSEILLE :PARENT NIL :ACTION NIL :DEPTH 12 :G ...)

**CG-USER**(101): (node-state (expand-node-action node-marseille-ex6 #S(ACTION :NAME NAVIGATE-TRAIN-TIME :ORIGIN MARSEILLE :FINAL TOULOUSE :COST 65.0) \*travel-fast\*))

TOULOUSE

Ejemplos de ejecución de la función expand-node:

**\*ejemplo del enunciado**

CL-USER(53): (defparameter lst-nodes-ex6

(expand-node node-marseille-ex6 \*travel-fast\*))

LST-NODES-EX6

CL-USER(54): (print lst-nodes-ex6)

(#S(NODE :STATE TOULOUSE

:PARENT #S(NODE :STATE MARSEILLE

:PARENT NIL

:ACTION NIL

:DEPTH 12

:G 10

:H 0

:F 20)

:ACTION #S(ACTION :NAME NAVIGATE-TRAIN-TIME

:ORIGIN MARSEILLE

:FINAL TOULOUSE

:COST 65.0)

:DEPTH 13

:G 75.0

:H 130.0

:F 205.0))

(#S(NODE :STATE TOULOUSE

:PARENT #S(NODE :STATE MARSEILLE

:PARENT NIL

:ACTION NIL

:DEPTH 12

:G ...)

:ACTION #S(ACTION :NAME NAVIGATE-TRAIN-TIME

:ORIGIN MARSEILLE

:FINAL TOULOUSE

:COST 65.0)

:DEPTH 13

:G ...))

**\*introduciendo un problema no valido**

CL-USER(77): (expand-node node-paris nil)

NIL

**\*introduciendo un nodo no valido**

CL-USER(78): (expand-node nil \*travel-cheap\*)

NIL

**\*Paris con la heuristica sobre el precio.**

**Hemos elegido parís porque tiene muchas conexiones con otras ciudades.**

CL-USER(79): (expand-node node-paris \*travel-cheap\*)

(#S(NODE :STATE REIMS

:PARENT #S(NODE :STATE PARIS

:PARENT #S(NODE

:STATE

NEVERS

:PARENT

NIL

:ACTION

NIL

:DEPTH

0

:G

...)

:ACTION NIL

:DEPTH 0

:G ...)

:ACTION #S(ACTION :NAME NAVIGATE-CANAL-PRICE

:ORIGIN PARIS

:FINAL REIMS

:COST 10.0)

:DEPTH 1

:G ...)

#S(NODE :STATE NANCY

:PARENT #S(NODE :STATE PARIS

:PARENT #S(NODE

:STATE

NEVERS

:PARENT

NIL

:ACTION

NIL

:DEPTH

0

:G

...)

:ACTION NIL

:DEPTH 0

:G ...)

:ACTION #S(ACTION :NAME NAVIGATE-CANAL-PRICE

:ORIGIN PARIS

:FINAL NANCY

:COST 10.0)

:DEPTH 1

:G ...)

#S(NODE :STATE CALAIS

:PARENT #S(NODE :STATE PARIS

:PARENT #S(NODE

:STATE

NEVERS

:PARENT

NIL

:ACTION

NIL

:DEPTH

0

:G

...)

:ACTION NIL

:DEPTH 0

:G ...)

:ACTION #S(ACTION :NAME NAVIGATE-TRAIN-PRICE

:ORIGIN PARIS

:FINAL CALAIS

:COST 60.0)

:DEPTH 1

:G ...)

#S(NODE :STATE NANCY

:PARENT #S(NODE :STATE PARIS

:PARENT #S(NODE

:STATE

NEVERS

:PARENT

NIL

:ACTION

NIL

:DEPTH

0

:G

...)

:ACTION NIL

:DEPTH 0

:G ...)

:ACTION #S(ACTION :NAME NAVIGATE-TRAIN-PRICE

:ORIGIN PARIS

:FINAL NANCY

:COST 67.0)

:DEPTH 1

:G ...)

#S(NODE :STATE NEVERS

:PARENT #S(NODE :STATE PARIS

:PARENT #S(NODE

:STATE

NEVERS

:PARENT

NIL

:ACTION

NIL

:DEPTH

0

:G

...)

:ACTION NIL

:DEPTH 0

:G ...)

:ACTION #S(ACTION :NAME NAVIGATE-TRAIN-PRICE

:ORIGIN PARIS

:FINAL NEVERS

:COST 75.0)

:DEPTH 1

:G ...)

#S(NODE :STATE ORLEANS

:PARENT #S(NODE :STATE PARIS

:PARENT #S(NODE

:STATE

NEVERS

:PARENT

NIL

:ACTION

NIL

:DEPTH

0

:G

...)

:ACTION NIL

:DEPTH 0

:G ...)

:ACTION #S(ACTION :NAME NAVIGATE-TRAIN-PRICE

:ORIGIN PARIS

:FINAL ORLEANS

:COST 38.0)

:DEPTH 1

:G ...)

#S(NODE :STATE ST-MALO

:PARENT #S(NODE :STATE PARIS

:PARENT #S(NODE

:STATE

NEVERS

:PARENT

NIL

:ACTION

NIL

:DEPTH

0

:G

...)

:ACTION NIL

:DEPTH 0

:G ...)

:ACTION #S(ACTION :NAME NAVIGATE-TRAIN-PRICE

:ORIGIN PARIS

:FINAL ST-MALO

:COST 70.0)

:DEPTH 1

:G ...))

**Ejemplos de ejecución ejercicio7:**

\*\*función auxiliar insert-node:

**CG-USER**(91): (defparameter aux (insert-node node-paris-ex7 (list node-nancy-ex7) (strategy-node-compare-p \*uniform-cost\*)))

AUX

**CG-USER**(92): (mapcar #'(lambda (x) (node-state x)) aux)

(PARIS NANCY)

(a esta función se le llama desde otras el control de errores de los otros dos parámetros ya se habría realizado previamente)

**CG-USER**(93): (insert-node nil (list node-nancy-ex7) (strategy-node-compare-p \*uniform-cost\*))

NIL

Insert-nodes-strategy básicamente llama a insert-nodes, también realiza los primeros controles de errores, pasándole la función comparación, por lo que directamente vamos a mostrar los resultados de ejecutar esta función:

**CG-USER**(97): sol-ex7

(#S(NODE :STATE PARIS :PARENT NIL :ACTION NIL :DEPTH 0 :G ...) #S(NODE :STATE NANCY :PARENT NIL :ACTION NIL :DEPTH 2 :G ...)

#S(NODE :STATE TOULOUSE

:PARENT #S(NODE :STATE MARSEILLE :PARENT NIL :ACTION NIL :DEPTH 12 :G ...)

:ACTION #S(ACTION :NAME NAVIGATE-TRAIN-TIME :ORIGIN MARSEILLE :FINAL TOULOUSE :COST 65.0)

:DEPTH 13

:G ...))

**CG-USER**(98): (mapcar #'(lambda (x) (node-state x)) sol-ex7)

(PARIS NANCY TOULOUSE)

**CG-USER**(99): (mapcar #'(lambda (x) (node-g x)) sol-ex7)

(0 50 75.0)

Si no introducimos ninguna estrategia válida:

CG-USER(132): (insert-nodes-strategy (list node-paris-ex7 node-nancy-ex7)

lst-nodes-ex6

nil)

NIL

Si no introducimos nodos a ordenar:

CG-USER(133): (insert-nodes-strategy nil

lst-nodes-ex6

\*uniform-cost\*)

(#S(NODE :STATE TOULOUSE

:PARENT #S(NODE :STATE MARSEILLE :PARENT NIL :ACTION NIL :DEPTH 12 :G ...)

:ACTION #S(ACTION :NAME NAVIGATE-TRAIN-TIME :ORIGIN MARSEILLE :FINAL TOULOUSE :COST 65.0)

:DEPTH 13

:G ...))

CG-USER(134): (equal (insert-nodes-strategy nil

lst-nodes-ex6

\*uniform-cost\*) lst-nodes-ex6)

T

Si no introducimos una lista de nodos ordenados:

CG-USER(136): (insert-nodes-strategy (list node-paris-ex7 node-nancy-ex7) nil

\*uniform-cost\*)

(#S(NODE :STATE PARIS :PARENT NIL :ACTION NIL :DEPTH 0 :G ...) #S(NODE :STATE NANCY :PARENT NIL :ACTION NIL :DEPTH 2 :G ...))

CG-USER(137): (mapcar #'(lambda (x) (node-state x)) (insert-nodes-strategy (list node-paris-ex7 node-nancy-ex7) nil

\*uniform-cost\*)

)

(PARIS NANCY)

CG-USER(138): (mapcar #'(lambda (x) (node-g x)) (insert-nodes-strategy (list node-paris-ex7 node-nancy-ex7) nil

\*uniform-cost\*))

(0 50)

**Ejemplos de ejecución ejercicio 8:**