Universidad Autónoma de Madrid

Inteligencia Artificial

práctica 2

Nombres	Daniel Tijerina González
	Suemy Inagaki Pinheiro Fagundes
Grupo	2363

Madrid, 24 de Marzo de 2020

1 Evaluación del Valor de la heurística

1.1 Analise de los Requisitos

- city es un nombre de una ciudad
- heuristica es una lista de listas
- heuristica puedes ser una lista vacia

1.2 Bateria de Ejemplos

```
; Cuando la ciudad no esta en heuristic
(f-h 'Calais '((Nancy 50.0) (Paris 30.0))) ;NIL

; Cuando la ciudad esta en heuristic
(f-h 'Calais '((Calais 10.0) (Paris 30.0))) ;10.0

; Cuando hay mas de una lista con la ciudad
(f-h 'Calais '((Calais 10.0) (Calais 15.0))) ;10.0

; Cuando heuristica es una lista vacia
(f-h 'Calais '()) ;NIL
```

1.3 Pseudocódigo

```
Inicio Funcion (ciudad heuristica)
    Si ciudad es el primer elemento
    de una lista de heuristica
        entonces retorna el segundo elemento de la listas
Fin Funcion
```

1.4 Definición de la Función

```
(defun f-h (city heuristic)
        (second (assoc city heuristic)))
```

1.5 Tests

```
>> (f-h 'Calais '((Nancy 50.0) (Paris 30.0)))
NIL
>> (f-h 'Calais '((Calais 10.0) (Paris 30.0)))
10.0
>> (f-h 'Calais '((Calais 10.0) (Calais 15.0)))
10.0
>> (f-h 'Calais '())
NIL
```

1.6 Comentarios

No es posible pasar como parámetro una ciudad vacia pues habrá un error en el número de parámetros.

Cuando hay más de una lista con la ciudad procurada, la funcion retorna la primera lista encontrada.

2 Definición del operador Navigate

2.1 Analise de los Requisitos

- la funcion debe funcionar con lista vacia
- la funcion debe criar una lista con los nombres de las ciudades para donde puedes seguir

2.2 Bateria de Ejemplos

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS PARA FUNCION AUXILIAR
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Crea una lista de ciudades para donde puedes ir desde Toulouse
; Caso típico, hay respuesta

(crea-lista-succ 'Toulouse *trains*)  ;((TOULOUSE NANTES 80.0)
;(TOULOUSE LIMOGES 25.0) (TOULOUSE LYON 60.0) (TOULOUSE MARSEILLE 65.0))

; La funcion intenta crear una lista a partir de una ciudad que
; no esta en la lista
(crea-lista-succ 'blablabla *trains*) ;NIL

; La funcion intenta crear una lista cuando edges es una lista vacia
(crea-lista-succ 'Paris '()) ;NIL

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS PARA FUNCION NAVIGATE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Teste tipico
(navigate 'Paris *trains*)
;(#S(ACTION :NAME NIL :ORIGIN PARIS :FINAL CALAIS :COST 34.0)
; #S(ACTION :NAME NIL :ORIGIN PARIS :FINAL NANCY :COST 40.0)
; #S(ACTION :NAME NIL :ORIGIN PARIS :FINAL NEVERS :COST 48.0)
; #S(ACTION :NAME NIL :ORIGIN PARIS :FINAL ORLEANS :COST 23.0)
; #S(ACTION :NAME NIL :ORIGIN PARIS :FINAL ST-MALO :COST 40.0))

; Teste cuando la ciudad no existe
(navigate 'Oi *trains*) ;NIL
```

```
; Teste cuando la lista es vacia
(navigate 'Paris '()) ;NIL
```

2.3 Pseudocódigo

```
FUNCIÓN AUXILIAR*****************************************

Inicio Funcion (city edges)
    Si edges no es NULL
        Si city es igual al primero elemento de la
        primera lista de edge:
            retorna una lista con:
                primer elemento igual a primera lista de edge
                segundo elemento igual a llamada recursiva de la
                funcion con el resto de edge.
        Caso contrario:
            llamada recursiva de la funcion donde el primer parametro
            es city y el segundo parametro es el resto de edge.
Fin Funcion

FUNCION NAVIGATE*************************************

Inicio Funcion (city edge)
    Para cada elemento del retorno de la llamada de la funcion auxiliar:
        crear una accion donde:
            la origen es el primer elemento de cada lista
            el final es el segundo elemento de cada lista
            el cost es el tercer elemento de cada lista.
Fin Funcion
```

2.4 Código de la Función

```
(defun crea-lista-succ (city edges)
  (unless (null edges)
    (if (equal city (first (first edges)))
      (cons (first edges) (crea-lista-succ city (cdr edges)))
      (crea-lista-succ city (cdr edges)))))

(defun navigate (city edges)
  (mapcar #'(lambda (lista)
    (make-action
        :origin (first lista)
        :final (second lista)
        :cost (third lista)))
    (crea-lista-succ city edges)))
```

2.5 Tests

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; TESTE FUNCION AUXILIAR
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

>> (crea-lista-succ 'Toulouse *trains*)
((TOULOUSE NANTES 80.0) (TOULOUSE LIMOGES 25.0)
(TOULOUSE LYON 60.0) (TOULOUSE MARSEILLE 65.0))

>> (crea-lista-succ 'blablabla *trains*)
NIL

>> (crea-lista-succ 'Paris '())
NIL

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; TESTE FUNCION NAVIGATE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

>> (navigate 'Paris *trains*)
(#S(ACTION :NAME NIL :ORIGIN PARIS :FINAL CALAIS :COST 34.0)
 #S(ACTION :NAME NIL :ORIGIN PARIS :FINAL NANCY :COST 40.0)
 #S(ACTION :NAME NIL :ORIGIN PARIS :FINAL NEVERS :COST 48.0)
 #S(ACTION :NAME NIL :ORIGIN PARIS :FINAL ORLEANS :COST 23.0)
 #S(ACTION :NAME NIL :ORIGIN PARIS :FINAL ST-MALO :COST 40.0))

>> (navigate 'Oi *trains*)
NIL

>> (navigate 'Paris '())
NIL
```

2.6 Comentarios

La funcion auxiliar crea una lista con todas las listas de edge donde el primer elemento es la ciudad procurada.

La funcion principal crea una lista de acciones a partir de la lista retornada pela funcion auxiliar

3 Test para determinar si se ha alcanzado el objetivo

3.1 Analise de los Requisitos

- el primer parametro de la funcion debe ser do tipo node

- el segundo parametro es una lista de destinos.
- el tercer parametro es una lista de ciudades obliatorias.
- la funcion debe funcionar recibiendo listas vacias.
- la funcion retorna T o NIL

3.2 Bateria de Ejemplos

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS PARA LA FUNCION CREA-CAMINO
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
(crea-camino node-calais);(CALAIS REIMS NANCY PARIS NEVERS)

; Cuando node es null
(crea-camino NIL) ;NIL

; Cuando node no tiene parent
(crea-camino (make-node :city 'oi)) ;NIL

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS PARA LA FUNCION CIUDADES-OBLIGATORIAS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
(ciudades-obligatorias (crea-camino node-calais) *mandatory*) ;T

; Cuando mandatory es NULL
(ciudades-obligatorias (crea-camino node-calais) NIL) ;T

; Cuando path es una lista vacia
(ciudades-obligatorias NIL *mandatory*) ;NIL


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS PARA LA FUNCION PRINCIPAL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico - en el camino hay todas las ciudades obligatorias
(f-goal-test node-calais '(Calais Marseille) '(Paris)) ; T

; Caso típico - ni todas las ciudades obligatorias fueron visitadas
(f-goal-test node-calais '(Calais Marseille) '(Hola Tchau)) ;; NIL
```

```
; Caso típico - La ciudad no esta en la lista de destino
(f-goal-test (make-node :city 'oi) '(calais marseille) '(bla)) ;NIL

; Destinos es una lista vacia
(f-goal-test node-calais '() *mandatory*) ;NIL
```

3.3 Pseudocódigo

```
Inicio Funcion (node destination mandatory)
    Si
        (la ciudad de node estuviere en la lista destination) Y
        (todas las ciudades de mandatory estuvieren en el
        camino por donde ha pasado desde la origen hasta llegar a node)
    Entonces:
        retorna True
Fin Funcion
```

3.4 Código de la Función

```
;Crea una lista con todos los nodes por donde
;ha pasado para llegar a node

(defun crea-camino (node)
        (unless (null node)
                (if (null (node-parent node))
                        (list (node-city node))
                        (cons (node-city node) (crea-camino (node-parent node))))))

; Es una funcion que verifica si todas las ciudades
; de mandatory esta en path. T caso sea verdad y NIL
; caso contrario

(defun ciudades-obligatorias (path mandatory)
        (if (null mandatory)
            T
             (unless (null path)
                     (and (find (car mandatory) path)
                         (ciudades-obligatorias path (cdr mandatory))))))

; Funcion principal

(defun f-goal-test (node destination mandatory)
        (and (find (node-city node) destination)
             (ciudades-obligatorias (crea-camino node) mandatory)))
```

3.5 Tests

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS PARA LA FUNCION CREA-CAMINO
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
>> (crea-camino node-calais)
(CALAIS REIMS NANCY PARIS NEVERS)

; Cuando node es null
>> (crea-camino NIL)
NIL

; Cuando node no tiene parent
>> (crea-camino (make-node :city 'oi))
NIL

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS PARA LA FUNCION CIUDADES-OBLIGATORIAS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
>> (ciudades-obligatorias (crea-camino node-calais) *mandatory*)
T

; Cuando mandatory es NULL
>> (ciudades-obligatorias (crea-camino node-calais) NIL)
T

; Cuando path es una lista vacia
>> (ciudades-obligatorias NIL *mandatory*)
NIL

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS PARA LA FUNCION PRINCIPAL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico - en el camino hay todas las ciudades obligatorias
>> (f-goal-test node-calais '(Calais Marseille) '(Paris))
T

; Caso típico - ni todas las ciudades obligatorias fueron visitadas
>> (f-goal-test node-calais '(Calais Marseille) '(Hola Tchau))
NIL
```

```
; Caso típico - La ciudad no esta en la lista de destino
>> (f-goal-test (make-node :city 'oi) '(calais marseille) '(bla))
NIL

; Destinos es una lista vacia
>> (f-goal-test node-calais '() *mandatory*)
NIL
```

3.6 Comentarios

La funcion crea-camino recibe un node y crea una lista con todos los nodes por donde ha pasado el camino desde la origen hasta el node.

Es una funcion necesaria pues la funcion ciudades-obligatorias verifica si todas las ciudades de mandatory ya fueron visitadas. Para eso, recibe una lista de nodes visitados (retornada por la funcion crea-camino) y una lista de ciudades obligatorias.

La funcion principal verifica si el node es uno de los destinos y si todas las ciudades obligatorias fueron visitadas.

4 Predicado para determinar la igualdad entre estados de búusqueda

4.1 Analise de los Requisitos

- La funcion debe recibir dos parametros do tipo node.
- la funcion verifica si el estado de busqueda es igual
- Dos estados de búsqueda son iguales si los dos nodos son iguales.
- Dos estados de búsqueda son iguales si en los dos casos, han pasado por todas las ciudades obligatorias.

4.2 Bateria de Ejemplos

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS FUNCION CREA-LISTA-CIUDADES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
(crea-lista-ciudades *mandatory* (crea-camino node-calais)) ;(PARIS)

; Cuando Mandatory es NULL
(crea-lista-ciudades NIL (crea-camino node-calais)) ;NIL

; Cuando path es NULL
(crea-lista-ciudades *mandatory* NIL) ;NIL
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS FUNCION VERIFICA-CONDICION-BUSQUEDA
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
(verifica-condicion-busqueda '(A B C) '(A B C)) ; T

; Cuando las listas no estan igualmente ordenadas
(verifica-condicion-busqueda '(A C B) '(A B C)) ;NIL

; Cuando una de las listas es vacia
(verifica-condicion-busqueda '(A C B) '()) ;NIL

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS FUNCION F-SEARCH-STATE-EQUAL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso Típico - las ciudades son iguales y no hay mandatory
(f-search-state-equal (make-node :city 'oi) (make-node :city 'oi :parent node-calais) '(

; Caso en que las ciudades son iguales pero no pasaran por las mismas
; ciudades obligatorias
(f-search-state-equal (make-node :city 'oi :parent node-nevers) (make-node :city 'oi :pa

; Caso en que las ciudades son iguales y pasaran por las mismas ciudades
; obligatorias
(f-search-state-equal (make-node :city 'oi :parent node-nevers) (make-node :city 'oi :pa

; Caso en que un dos nodes es null
(f-search-state-equal (make-node :city 'oi) NIL '(Nevers)) ;NIL
```

4.3 Pseudocódigo

```
Inicio Funcion (node1, node2, mandatory)
    Si:
        (node1 es igual a node2) Y
        (todas las ciudades obligatorias que fueran visitadas
        por node1 también hubieren sido visitadas por node2)
    Entonces:
        Retorna T
    Caso contrario:
        Retorna NIL
Fim funcion
```

4.4 Codigo de la Función

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Funciones Auxiliares
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun crea-lista-ciudades (mandatory path)
   (unless (null path)
      (if (find (car path) mandatory)
         (cons (car path) (crea-lista-ciudades mandatory (cdr path)))
         (crea-lista-ciudades mandatory (cdr path)))))

(defun verifica-condicion-busqueda (path1 path2)
   (unless (or (null path1) (null path2))
      (if (and (null (cdr path1)) (null (cdr path2)))
            T
            (and (equal (car path1) (car path2))
                 (verifica-condicion-busqueda (cdr path1) (cdr path2))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Funcion Principal
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun f-search-state-equal (node-1 node-2 &optional mandatory)
   (if (null mandatory)
      (equal (node-city node-1) (node-city node-2))
          (and (equal (node-city node-1) (node-city node-2))
               (verifica-condicion-busqueda
                   (sort (crea-lista-ciudades mandatory (crea-camino node-1))
                         #'string-lessp)
                   (sort (crea-lista-ciudades mandatory (crea-camino node-2))
                         #'string-lessp)))))
```

4.5 Tests

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS FUNCION CREA-LISTA-CIUDADES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
>> (crea-lista-ciudades *mandatory* (crea-camino node-calais))
(PARIS)

; Cuando Mandatory es NULL
>> (crea-lista-ciudades NIL (crea-camino node-calais))
NIL
```

```
; Cuando path es NULL
>> (crea-lista-ciudades *mandatory* NIL)
NIL


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS FUNCION VERIFICA-CONDICION-BUSQUEDA
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
>> (verifica-condicion-busqueda '(A B C) '(A B C))
T

; Cuando las listas no estan igualmente ordenadas
>> (verifica-condicion-busqueda '(A C B) '(A B C))
NIL

; Cuando una de las listas es vacia
>> (verifica-condicion-busqueda '(A C B) '())
NIL

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS FUNCION F-SEARCH-STATE-EQUAL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso Típico - las ciudades son iguales y no hay mandatory
>> (f-search-state-equal (make-node :city 'oi) (make-node :city 'oi
:parent node-calais) '())
T

; Caso en que las ciudades son iguales pero no pasaran por las mismas
; ciudades obligatorias
>> (f-search-state-equal (make-node :city 'oi :parent node-nevers)
(make-node :city 'oi :parent node-paris) '(Paris))
NIL

; Caso en que las ciudades son iguales y pasaran por las mismas ciudades
; obligatorias
>> (f-search-state-equal (make-node :city 'oi :parent node-nevers)
(make-node :city 'oi :parent node-paris) '(Nevers))
T

; Caso en que un dos nodes es null
>> (f-search-state-equal (make-node :city 'oi) NIL '(Nevers))
```

```
NIL
```

4.6 Comentarios

Es una funcion que verifica si dos estados de busqueda son iguales.

Dos estados son iguales si los nodos son iguales y si las mismas ciudades obligatorias fueron visitadas por los dos nodos.

La funcion crea-lista-ciudades es una funcion que crea una lista de ciudades obligatorias por donde node ha pasado.

Crearemos una lista para cada node, node-1 y node-2.

La funcion verifica-condicion-busqueda es una funcion que verifica si dos listas de camino son iguales.

En este caso, los caminos son las listas retornadas por la funcion crea-lista-ciudades. O sea, es una lista de ciudades que están en mandatory y que cada nodo ha visitado.

Así, si las dos listas son iguales, la funcion retorna T, caso contrario retorna NIL.

Las dos listas pasadas como parametro deben tener el mismo tamaño, pues caso no lo tengan, eso ya es un indicativo de que las listas no son iguales. (Retorna NIL)

Como las listas ya estan ordenadas alfabeticamente, basta ahora verificar si los elementos son iguales, uno a uno.

5 Representación LISP del problema

5.1 Analise de los Requisitos

- El parametro travel debe utilizar las funciones ya definidas
- Las funciones utilizadas en travel debe recibir los parametros propuestos en el enunciado.

5.2 Bateria de Ejemplos

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS DE LAS FUNCIONES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(succ node-calais)
; (#S(ACTION :NAME NIL :ORIGIN CALAIS :FINAL PARIS :COST 34.0)
;  #S(ACTION :NAME NIL :ORIGIN CALAIS :FINAL REIMS :COST 35.0))

(goal-test node-calais) ;T

(call-sse node-calais node-calais-2) ;T

(call-f-h 'Calais) ;0.0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EL EJEMPLO DE UTILIZACION DEL PARAMETRO TRAVEL
```

```
; ESTARÁ EN LOS PROXIMOS EJERCICIOS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

5.3 Pseudocódigo

No hay pseudocodigo pues es solamente una declaracion de parametro.

5.4 Declaracion del Parametro

```
(defun succ (node)
        (navigate (node-city node) *trains*))

(defun goal-test (node)
        (f-goal-test node *destination* *mandatory*))

(defun call-sse (node1 node2)
        (f-search-state-equal node1 node2 *mandatory*))

(defun call-f-h (city)
        (f-h city *heuristic*))

(defparameter *travel*
        (make-problem
                :cities *cities*
                :initial-city *origin*
                :f-h #'call-f-h
                :f-goal-test #'goal-test
                :f-search-state-equal #'call-sse
            :succ #'succ))
```

5.5 Tests

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS DE LAS FUNCIONES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

>> (succ node-calais)
(#S(ACTION :NAME NIL :ORIGIN CALAIS :FINAL PARIS :COST 34.0)
#S(ACTION :NAME NIL :ORIGIN CALAIS :FINAL REIMS :COST 35.0))

>> (goal-test node-calais)
T

>> (call-sse node-calais node-calais-2)
T
```

```
>> (call-f-h 'Calais)
0.0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EL TEST DE UTILIZACION DEL PARAMETRO TRAVEL
; ESTARÁ EN LOS PROXIMOS EJERCICIOS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

5.6 Comentários

Para adaptar las funciones para recibiren solamente los parametros descriptos en el enunciado, tuvimos que definir funciones auxiliares cuyas tareas son solamente llamar las funciones principales, donde algunos parámetros son parámetros globales especificos del problema.

6 Expansión de un nodo

6.1 Analise de los Requisitos

- La funcion recibe un node y un problema
- la funcion retorna una lista de nodes para donde el node recibido puede expandir.
- el parent de los nodos expandidos debe ser el nodo analisado.

6.2 Bateria de Ejemplos

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS DE LA FUNCION EXPAND-NODE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso tipico
(expand-node node-calais *travel*)
;(#S(NODE
;    :CITY PARIS
;    :PARENT #S(NODE
;                :CITY CALAIS
;                :PARENT NIL
;                :ACTION NIL
;                :DEPTH 0
;                :G 0
;                :H 0
;                :F 0)
;    :ACTION #S(ACTION :NAME NIL :ORIGIN CALAIS :FINAL PARIS :COST 34.0)
;    :DEPTH 1
;    :G 34.0
;    :H 0
```

```
;    :F 0)
; #S(NODE
;    :CITY REIMS
;    :PARENT #S(NODE
;                :CITY CALAIS
;                :PARENT NIL
;                :ACTION NIL
;                :DEPTH 0
;                :G 0
;                :H 0
;                :F 0)
;    :ACTION #S(ACTION :NAME NIL :ORIGIN CALAIS :FINAL REIMS :COST 35.0)
;    :DEPTH 1
;    :G 35.0
;    :H 0
;    :F 0))

; Caso en que no hay node
(expand-node NIL *travel*) ; NIL
```

6.3 Pseudocódigo

```
Inicio Funcion (node problem)
    crear una lista de acciones cuya la ciudad final es
    la ciudad para donde el nodo puede expandirse,
    utilizando la funcion disponible para eso en problem.

    Para cada elemento de la lista retornada, crear un node cuya:
        ciudad es el destino de la action retornada
        parent es el node
        action es el proprio elemento
        depth es el depth del node + 1
        g es el coste del node + el coste de la accion

    retornar la lista de nodes
Fim funcion
```

6.4 Definicion de la Funcion

```
(defun expand-node (node problem)
        (unless (null node)
            (mapcar #'(lambda (lista)
                (make-node
                    :city (action-final lista)
                    :parent (make-node
                            :city (action-origin lista)
```

```
                        :depth (node-depth node)
                        :g (node-g node))
                    :action lista
                    :depth (+ (node-depth node) 1)
                    :g (+ (node-g node) (action-cost lista))))
                (funcall (problem-succ problem) node))))
```

6.5 Tests

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS DE LA FUNCION EXPAND-NODE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso tipico
>> (expand-node node-calais *travel*)
(#S(NODE
    :CITY PARIS
    :PARENT #S(NODE
               :CITY CALAIS
               :PARENT NIL
               :ACTION NIL
               :DEPTH 0
               :G 0
               :H 0
               :F 0)
    :ACTION #S(ACTION :NAME NIL :ORIGIN CALAIS :FINAL PARIS :COST 34.0)
    :DEPTH 1
    :G 34.0
    :H 0
    :F 0)
 #S(NODE
    :CITY REIMS
    :PARENT #S(NODE
               :CITY CALAIS
               :PARENT NIL
               :ACTION NIL
               :DEPTH 0
               :G 0
               :H 0
               :F 0)
    :ACTION #S(ACTION :NAME NIL :ORIGIN CALAIS :FINAL REIMS :COST 35.0)
    :DEPTH 1
    :G 35.0
    :H 0
    :F 0))
; Caso en que no hay node
```

```
>> (expand-node NIL *travel*)
NIL
```

6.6 Comentários

La funcion es muy simple, su unica tarea es crear un node para cada elemento de la lista retornada cuando se llama problem-succ.

7 Gestión de nodos

7.1 Analise de los Requisitos

- La funcion debe poder inserir un node de acuerdo con la estrategia.
- la funcion debe recibir parametro do tipo node

7.2 Bateria de Ejemplos

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS PARA INSERT-NODE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defparameter node-a (make-node :city 'a :depth 3 :g 150 :f 0) )
(defparameter node-b (make-node :city 'b :depth 1 :g 50 :f 0) )
(defparameter node-c (make-node :city 'c :depth 0 :g 100 :f 0) )
(defparameter node-w (make-node :city 'w :depth 2 :g 120 :f 0) )

; Caso tipico
(mapcar #' (lambda (x) (node-city x)) (insert-node node-w (list node-b
 node-c node-a) (strategy-node-compare-p *uniform-cost*))) ; (B C W A)

 ;Caso en que la lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-node node-w '()
(strategy-node-compare-p *uniform-cost*))) ;(W)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS PARA INSERT-NODES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
(mapcar #' (lambda (x) (node-city x)) (insert-nodes (list node-w node-b)
(list node-c node-a) (strategy-node-compare-p *uniform-cost*))) ; (B C W A)

; Caso en que la primera lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-nodes '()
(list node-c node-a) (strategy-node-compare-p *uniform-cost*))) ;(C A)
```

```
; Caso en que la segunda lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-nodes (list node-a node-w)
'() (strategy-node-compare-p *uniform-cost*))) ;(W A)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS PARA INSERT-NODES-STRATEGY
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
(mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy (list node-w node-b)
(list node-c node-a) *uniform-cost*)) ; (B C W A)

; Caso en que la primera lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy '()
(list node-c node-a) *uniform-cost*)) ;(C A)

; Caso en que la segunda lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy (list node-a
node-w) '() *uniform-cost*)) ;(W A)
```

7.3 Pseudocódigo

```
Inicio Funcion (nodes lst-nodes stra)
    Si nodes solo tiene un elemento, entonces:
        inserir el elemento de nodes en lst-nodes
        de acuerdo con la funcion de comparacion que puede ser acesada por stra.
    Caso contrario:
        llamada recursiva donde los parametros son:
            1- nodes sin el primer elemento
            2- nueva lst-nodes, donde el primer elemento de nodes
            fuera inserido tambien.
            3- stra
Fin Funcion
```

7.4 Código de la Funcion

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Funcion que insere un node a una lista
; de acuerdo con la funcion de comparacion
; pasada como parámetro
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun insert-node (node lst1 node-compare-p)
        (if (null lst1)
                (list node)
```

```
                (if (funcall node-compare-p node (car lst1))
                        (cons node lst1)
                        (cons (car lst1) (insert-node node (cdr lst1)
                        node-compare-p)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Funcion que insere una lista de nodes a otra
; lista ya ordenada
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun insert-nodes (nodes lst-nodes node-compare-p)
        (if (null nodes)
                lst-nodes
                (if (null (cdr nodes))
                        (insert-node (car nodes) lst-nodes node-compare-p)
                        (insert-nodes (cdr nodes) (insert-node (car nodes)
                        lst-nodes node-compare-p) node-compare-p))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Funcion que insere una lista de nodes no necesariamente
; ordenada a otra lista ya ordenada
; Recibe una estrategia
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun insert-nodes-strategy (nodes lst-nodes stra)
        (insert-nodes nodes lst-nodes (strategy-node-compare-p stra)))
```

7.5 Tests

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;TESTS PARA INSERT-NODE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defparameter node-a (make-node :city 'a :depth 3 :g 150 :f 0) )
(defparameter node-b (make-node :city 'b :depth 1 :g 50 :f 0) )
(defparameter node-c (make-node :city 'c :depth 0 :g 100 :f 0) )
(defparameter node-w (make-node :city 'w :depth 2 :g 120 :f 0) )

; Caso tipico
>> (mapcar #' (lambda (x) (node-city x)) (insert-node node-w (list
node-b node-c node-a) (strategy-node-compare-p *uniform-cost*)))
(B C W A)

>> (mapcar #' (lambda (x) (node-g x)) (insert-node node-w (list
node-b node-c node-a) (strategy-node-compare-p *uniform-cost*)))
(50 100 120 150)

;Caso en que la lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-node node-w '()
```

```
(strategy-node-compare-p *uniform-cost*)))
(W)

 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS PARA INSERT-NODES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes (list node-w node-b)
(list node-c node-a) (strategy-node-compare-p *uniform-cost*)))
(B C W A)

; Caso en que la primera lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes '() (list node-c node-a)
(strategy-node-compare-p *uniform-cost*)))
(C A)

; Caso en que la segunda lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes (list node-a
node-w) '() (strategy-node-compare-p *uniform-cost*)))
(W A)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS PARA INSERT-NODES-STRATEGY
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy (list node-w
node-b) (list node-c node-a) *uniform-cost*))
(B C W A)

; Caso en que la primera lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy '()
(list node-c node-a) *uniform-cost*))
(C A)

; Caso en que la segunda lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy (list
node-a node-w) '() *uniform-cost*))
(W A)
```

7.6 Comentários

La funcion insert-node es una funcion que insere un node a una lista de nodes ordenados, de acuerdo con la funcion de comparacion pasada como parametro. Esa funcion servirá como funcion auxiliar a la funcion insert-nodes, que insere una lista de nodes no necesariamente

ordenados a otra lista de nodes ya ordenados.

La funcion insert-nodes-strategy es la funcion principal. Su papel es llamar la funcion insert-nodes pasando como tercer parametro la funcion de comparacion que puede ser acesada por strategy-node-compare-p.

8 Definir la estrategia para la búsqueda A

8.1 Analise de los Requisitos

- los parámetros deben ser del tipo node

8.2 Bateria de Ejemplos

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;EJEMPLOS CON INSERT-NODE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defparameter node-a (make-node :city 'a :depth 3 :g 150 :f 0) )
(defparameter node-b (make-node :city 'b :depth 1 :g 50 :f 0) )
(defparameter node-c (make-node :city 'c :depth 0 :g 100 :f 0) )
(defparameter node-w (make-node :city 'w :depth 2 :g 120 :f 0) )

; Caso tipico
(mapcar #' (lambda (x) (node-city x)) (insert-node node-w (list
node-b node-c node-a) (strategy-node-compare-p *a-star*))) ; (W B C A)

(mapcar #' (lambda (x) (node-g x)) (insert-node node-w (list
node-b node-c node-a) (strategy-node-compare-p *a-star*))) ;(120 50 100 150)

;Caso en que la lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-node node-w '()
(strategy-node-compare-p *a-star*))) ;(W)

 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS CON INSERT-NODES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
(mapcar #' (lambda (x) (node-city x)) (insert-nodes (list node-w node-b)
(list node-c node-a) (strategy-node-compare-p *a-star*))) ;(B W C A)

; Caso en que la primera lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-nodes '() (list node-c node-a)
(strategy-node-compare-p *a-star*))) ;(C A)

; Caso en que la segunda lista es vacia
```

```
(mapcar #' (lambda (x) (node-city x)) (insert-nodes (list node-a node-w)
'() (strategy-node-compare-p *a-star*))) ;(W A)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS CON INSERT-NODES-STRATEGY
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
(mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy (list node-w
node-b) (list node-c node-a) *a-star*)) ;(B W C A)

; Caso en que la primera lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy '()
(list node-c node-a) *a-star*)) ;(C A)

; Caso en que la segunda lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy
(list node-a node-w) '() *a-star*)) ;(W A)
```

8.3 Pseudocódigo

```
Inicio Funcion (node1 node2)
    Retorna el resultado de la comparacion:
        node1-f <= node2-f
Fin Funcion
```

8.4 Código de la Funcion

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Funcion de comparacion
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun func-a-star (node1 node2)
        (<= (node-f node1) (node-f node2)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Definicion de la Estrategia
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defparameter *A-star*
  (make-strategy
          :name '*A-star*
          :node-compare-p #'func-a-star))
```

8.5 Tests

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;TESTS CON INSERT-NODE
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defparameter node-a (make-node :city 'a :depth 3 :g 150 :f 0) )
(defparameter node-b (make-node :city 'b :depth 1 :g 50 :f 0) )
(defparameter node-c (make-node :city 'c :depth 0 :g 100 :f 0) )
(defparameter node-w (make-node :city 'w :depth 2 :g 120 :f 0) )

; Caso tipico
>> (mapcar #' (lambda (x) (node-city x)) (insert-node node-w (list
node-b node-c node-a) (strategy-node-compare-p *a-star*)))
(W B C A)

>> (mapcar #' (lambda (x) (node-g x)) (insert-node node-w (list
node-b node-c node-a) (strategy-node-compare-p *a-star*)))
(120 50 100 150)

;Caso en que la lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-node node-w '()
(strategy-node-compare-p *a-star*)))
(W)

 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS CON INSERT-NODES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes (list node-w node-b)
(list node-c node-a) (strategy-node-compare-p *a-star*)))
(B W C A)

; Caso en que la primera lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes '() (list node-c node-a)
(strategy-node-compare-p *a-star*)))
(C A)

; Caso en que la segunda lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes (list node-a node-w)
'() (strategy-node-compare-p *a-star*)))
(W A)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS CON INSERT-NODES-STRATEGY
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy (list node-w
```

```
node-b) (list node-c node-a) *a-star*))
(B W C A)

; Caso en que la primera lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy '()
(list node-c node-a) *a-star*))
(C A)

; Caso en que la segunda lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy
(list node-a node-w) '() *a-star*))
(W A)
```

9 Función de búsqueda

9.1 Analise de los Requisitos

- la funcion debe retornar un nodo como resultado
- la funcion debe ser capaz de solucionar el problema recibindo solamente problem y strategy como parametros
- cuando no hay soluciones la funcion retorna NIL

9.2 Bateria de Ejemplos

```
(node-g (graph-search *travel* *A-star*)) ;; 202.0
(node-city (graph-search *travel* *A-star*)) ;; CALAIS
(node-city (node-parent (graph-search *travel* *A-star*))) ;; PARIS
```

9.3 Pseudocódigo

```
1. Inicializa la lista open-nodes open-nodes open-nodes con el estado inicial
2. Inicializa la lista closed-nodes closed-nodes closed-nodes como NIL
3. repeat
4.    if open-nodes open-nodes es '()
5.       terminar [No hay solucion]
6.    extrae el primer nodo de open-nodes (nodo n)
7.    if goal-test goal-test es T para este nodo
8.       evalua la solucion y termina
9.    else
10.      if exp-cond(n)
11.         expande el nodo n
12.         inserta todos los nodos generados en open-nodes
13.         inserta n en la lista closed-nodes
14.         elimina n de la lista open-nodes
```

```
15.       fi
16.     fi
17. end
```

9.4 Código de la Funcion

```
;;;;;;;;;;;;;;;;;;;;;;
; FUNCIONES AUXILIARES
;;;;;;;;;;;;;;;;;;;;;;
(defun exp-cond (node closed-nodes)
  (or (not (find node closed-nodes))
    (< (node-g node) (node-g (find node closed-nodes)))))

(defun node-in-lst (open-nodes closed-nodes func)
  (if (null closed-nodes)
    open-nodes
    (unless (null open-nodes)
      (if (funcall func (car open-nodes) (find (car open-nodes) closed-nodes))
        (node-in-lst (remove (car open-nodes) open/-nodes) closed-nodes func)
        (node-in-lst (cdr open-nodes) closed-nodes func)))))


(defun graph-search-aux (open-nodes closed-nodes strategy problem)
  (unless (null open-nodes)
        (if (funcall (problem-f-goal-test problem) (car open-nodes))
              (car open-nodes)
              (if (exp-cond (car open-nodes) closed-nodes)
                    (graph-search-aux
                          (node-in-lst
                                (insert-nodes-strategy
                                    (expand-node (car open-nodes) problem)
                                    (cdr open-nodes)
                                    strategy)
                                (insert-node (car open-nodes) closed-nodes
                                    (strategy-node-compare-p strategy))
                                (problem-f-search-state-equal problem))
                          (insert-node (car open-nodes) closed-nodes
                            (strategy-node-compare-p strategy))
                          strategy
                          problem)))))

;;;;;;;;;;;;;;;;;;;;;;
; FUNCION PRINCIPAL
;;;;;;;;;;;;;;;;;;;;;;
(defun graph-search (problem strategy)
```

```
  (graph-search-aux (list (make-node :city (problem-initial-city problem)))
            '() strategy problem))
```

9.5 Tests

```
>> (node-g (graph-search *travel* *A-star*))
202.0

>> (node-city (graph-search *travel* *A-star*))
CALAIS

>> (node-city (node-parent (graph-search *travel* *A-star*)))
PARIS
```

10 Ver el camino seguido y la sequencia de acciones

10.1 Analise de Requisitos

- Las funciones reciben un parametro do tipo node
- la funcion solution path retorna una lista de ciudades
- la funcion action-sequence retorna una lista de aciones
-

10.2 Bateria de Ejemplos

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS PARA LA FUNCION SOLUTION-PATH
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
(solution-path node-calais-2) ;(NEVERS PARIS CALAIS)

;Caso en que nodo es null
(solution-path NIL) ;NIL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS PARA LA FUNCION ACTION-SEQUENCE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso Típico
(action-sequence node-calais)
;(#S(ACTION :NAME NIL :ORIGIN REIMS :FINAL CALAIS :COST 0)
; #S(ACTION :NAME NIL :ORIGIN NANCY :FINAL REIMS :COST 0)
; #S(ACTION :NAME NIL :ORIGIN PARIS :FINAL NANCY :COST 0)
```

```
; #S(ACTION :NAME NIL :ORIGIN NEVERS :FINAL PARIS :COST 0)
; #S(ACTION :NAME NIL :ORIGIN NIL :FINAL NEVERS :COST 0))

; Cuando Node es NUll
(action-sequence NIL) ;NIL
```

10.3 Pseudocodigo

```
;;;;;;;;;;;;;;;;;;;;
; Solution Path
;;;;;;;;;;;;;;;;;;;;
Inicio Funcion (node)
    Si node no es null
        llama la funcion crea-camino
        retorna el reverso de la lista retornada
Fin Funcion

;;;;;;;;;;;;;;;;;;;;
; Action Sequence
;;;;;;;;;;;;;;;;;;;;
Inicio Funcion (node)
    Si node no es null
        si node no tiene parent
            crea una acion con node
            retorna la acion
        caso contrario:
            cria una acion con node
            llamada recursiva funcion con node-parent
            retorna una lista con las dos cosas arriba
Fun Funcion
```

10.4 Codigo de la Funcion

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Solution Path
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun solution-path (node)
  (unless (null node)
    (reverse (crea-camino node))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Action Sequence
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun crea-lista-nodes (node)
  (unless (null (node-parent node))
```

```
    (cons node (crea-lista-nodes (node-parent node)))))

(defun crea-action (node)
  (unless (null node)
    (if (null (node-parent node))
      (make-action
        :final (node-city node)
        :cost (node-g node))
      (make-action
        :origin (node-city (node-parent node))
        :final (node-city node)
        :cost (node-g node)))))

(defun action-sequence (node)
  (unless (null node)
    (if (null (node-parent node))
      (list (crea-action node))
      (cons  (crea-action node) (action-sequence (node-parent node))))))
```

10.5 Tests

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS PARA LA FUNCION SOLUTION-PATH
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
(solution-path node-calais-2) ;(NEVERS PARIS CALAIS)

;Caso en que nodo es null
>> (solution-path NIL)
NIL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS PARA LA FUNCION ACTION-SEQUENCE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso Típico
>> (action-sequence node-calais)
(#S(ACTION :NAME NIL :ORIGIN REIMS :FINAL CALAIS :COST 0)
#S(ACTION :NAME NIL :ORIGIN NANCY :FINAL REIMS :COST 0)
#S(ACTION :NAME NIL :ORIGIN PARIS :FINAL NANCY :COST 0)
#S(ACTION :NAME NIL :ORIGIN NEVERS :FINAL PARIS :COST 0)
#S(ACTION :NAME NIL :ORIGIN NIL :FINAL NEVERS :COST 0))

; Cuando Node es NUll
>> (action-sequence NIL)
NIL
```

11 Otras estrategias de busqueda

11.1 Analise de Requisitos

- la funcion depth first node compare p debe comparar segun el valor de depth
- la funcion breadth first node compare p debe comparar segun el valor de depth de los parents de cada node.

11.2 Bateria de Ejemplos

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;EJEMPLOS CON INSERT-NODE - DEPTH FIRST
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defparameter node-a (make-node :city 'a :depth 3 :g 150 :f 0) )
(defparameter node-b (make-node :city 'b :depth 1 :g 50 :f 0) )
(defparameter node-c (make-node :city 'c :depth 0 :g 100 :f 0) )
(defparameter node-w (make-node :city 'w :depth 2 :g 120 :f 0) )

; Caso tipico
(mapcar #' (lambda (x) (node-city x)) (insert-node node-w (list
node-c node-b node-a) (strategy-node-compare-p *depth-first*))) ;(C B W A)

(mapcar #' (lambda (x) (node-g x)) (insert-node node-w (list
node-c node-b node-a) (strategy-node-compare-p *depth-first*))) ;(120 100 50 150)

;Caso en que la lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-node node-w '()
(strategy-node-compare-p *depth-first*))) ;(W)

 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EJEMPLOS CON INSERT-NODES - DEPTH FIRST
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
(mapcar #' (lambda (x) (node-city x)) (insert-nodes (list node-w node-b)
(list node-c node-a) (strategy-node-compare-p *depth-first*))) ;(C B W A)

; Caso en que la primera lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-nodes '() (list node-c node-a)
(strategy-node-compare-p *depth-first*))) ;(C A)

; Caso en que la segunda lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-nodes (list node-a node-w)
'() (strategy-node-compare-p *depth-first*))) ;(W A)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;EJEMPLOS CON INSERT-NODES-STRATEGY - DEPTH FIRST
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
(mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy (list node-w
node-b) (list node-c node-a) *depth-first*)) ;(C B W A)

; Caso en que la primera lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy '()
(list node-c node-a) *depth-first*)) ;(C A)

; Caso en que la segunda lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy
(list node-a node-w) '() *depth-first*)) ;(W A)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;EJEMPLOS CON INSERT-NODE - BREADTH FIRST
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defparameter node-a (make-node :city 'a :depth 0 :g 150 :f 0) )
(defparameter node-b (make-node :city 'b :parent node-a :depth 1 :g 50 :f 0) )
(defparameter node-c (make-node :city 'c :parent node-a :depth 1 :g 100 :f 0) )
(defparameter node-d (make-node :city 'd :parent node-b :depth 2 :g 150 :f 0) )
(defparameter node-e (make-node :city 'e :parent node-b :depth 2 :g 50 :f 0) )
(defparameter node-w (make-node :city 'w :parent node-a :depth 1 :g 120 :f 0) )


 ; Caso tipico
(mapcar #' (lambda (x) (node-city x)) (insert-node node-w
(list node-b node-c node-d node-e) (strategy-node-compare-p *breadth-first*)))
;(W B C D E)

(mapcar #' (lambda (x) (node-depth x)) (insert-node node-w
(list node-b node-c node-d node-e) (strategy-node-compare-p *breadth-first*)))
;(1 1 1 2 2)

;Caso en que la lista es vacia
(mapcar #' (lambda (x) (node-city x)) (insert-node node-w '()
(strategy-node-compare-p *breadth-first*))) ; (W)
```

11.3 Pseudocodigo

```
;;;;;;;;;;;;;;;
; DEPTH-FIRST
;;;;;;;;;;;;;;;
```

```
Inicio Funcion (node1 node2)
    Si depth de node 1 <= depth node 2
        retorna T
    Caso contrario
        retorna NIL
Fin Funcion

;;;;;;;;;;;;;;
; BREADTH-FIRST
;;;;;;;;;;;;;;
Inicio Funcion (node1 node2)
    Si depth de parent de node 1 <= depth de parent de node 2
        retorna T
    Caso contrario
        retorna NIL
Fin Funcion
```

11.4 Codigo de la Funcion

```
(defun depth-first-node-compare-p (node-1 node-2)
        (<= (node-depth node-1) (node-depth node-2))
)

(defparameter *depth-first*
        (make-strategy
                :name 'depth-first
                :node-compare-p #'depth-first-node-compare-p))

(defun breadth-first-node-compare-p (node-1 node-2)
        (if (or (null (node-parent node-1)) (null (node-parent node-2)))
                (null (node-parent node-1))
                (<= (node-depth (node-parent node-1))
                    (node-depth (node-parent node-2)))
        )
)

(defparameter *breadth-first*
        (make-strategy
                :name 'depth-first
                :node-compare-p #'breadth-first-node-compare-p))
```

11.5 Tests

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;TESTS CON INSERT-NODE - DEPTH FIRST
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defparameter node-a (make-node :city 'a :depth 3 :g 150 :f 0) )
(defparameter node-b (make-node :city 'b :depth 1 :g 50 :f 0) )
(defparameter node-c (make-node :city 'c :depth 0 :g 100 :f 0) )
(defparameter node-w (make-node :city 'w :depth 2 :g 120 :f 0) )

; Caso tipico
>> (mapcar #' (lambda (x) (node-city x)) (insert-node node-w (list
node-c node-b node-a) (strategy-node-compare-p *depth-first*)))
(C B W A)

>> (mapcar #' (lambda (x) (node-g x)) (insert-node node-w (list
node-c node-b node-a) (strategy-node-compare-p *depth-first*)))
(120 100 50 150)

;Caso en que la lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-node node-w '()
(strategy-node-compare-p *depth-first*)))
(W)

 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS CON INSERT-NODES - DEPTH FIRST
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes (list node-w node-b)
(list node-c node-a) (strategy-node-compare-p *depth-first*)))
(C B W A)

; Caso en que la primera lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes '() (list node-c node-a)
(strategy-node-compare-p *depth-first*)))
(C A)

; Caso en que la segunda lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes (list node-a node-w)
'() (strategy-node-compare-p *depth-first*)))
(W A)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; TESTS CON INSERT-NODES-STRATEGY - DEPTH FIRST
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Caso típico
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy (list node-w
```

```
node-b) (list node-c node-a) *depth-first*))
(C B W A)

; Caso en que la primera lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy '()
(list node-c node-a) *depth-first*))
(C A)

; Caso en que la segunda lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-nodes-strategy
(list node-a node-w) '() *depth-first*))
(W A)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;TESTS CON INSERT-NODE - BREADTH FIRST
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defparameter node-a (make-node :city 'a :depth 0 :g 150 :f 0) )
(defparameter node-b (make-node :city 'b :parent node-a :depth 1 :g 50 :f 0) )
(defparameter node-c (make-node :city 'c :parent node-a :depth 1 :g 100 :f 0) )
(defparameter node-d (make-node :city 'd :parent node-b :depth 2 :g 150 :f 0) )
(defparameter node-e (make-node :city 'e :parent node-b :depth 2 :g 50 :f 0) )
(defparameter node-w (make-node :city 'w :parent node-a :depth 1 :g 120 :f 0) )


 ; Caso tipico
>> (mapcar #' (lambda (x) (node-city x)) (insert-node node-w
(list node-b node-c node-d node-e) (strategy-node-compare-p *breadth-first*)))
(W B C D E)

>> (mapcar #' (lambda (x) (node-depth x)) (insert-node node-w
(list node-b node-c node-d node-e) (strategy-node-compare-p *breadth-first*)))
(1 1 1 2 2)

;Caso en que la lista es vacia
>> (mapcar #' (lambda (x) (node-city x)) (insert-node node-w '()
(strategy-node-compare-p *breadth-first*)))
(W)
```

12 Heuristica de Coste

12.1 Problema 1

```
(defparameter *heuristic-new*
 '((Orleans 0.0) (Paris 23.0) (Nantes 37.0)
```

```
        (Limoges 55.0) (Calais 57.0) (St-Malo 57.0)
        (Nancy 63.0)  (Nevers 71.0)  (Brest 72.0)
         (Reims 73.0) (Toulouse 80.0) (Lyon 140.0)
        (Marseille 145.0) (Roenne 158.0)  (Avigon 161.0))
)
```

13 Respondendo a las preguntas

- 1- a) Las vantajas de su realizar este diseño es que es posible resolver cualquier problema de búsqueda en grafo y es fácil de modificar los parametros, sin necesidade de alterar todo el codigo para que resolva otros problemas.
- 1- b) Para facilitar la visualizacion del reusultado, ya que muchas vecer retornan una estructura o una lista de estructuras y no es tan facil entenderlos. Así, utilizando lambda podemos verificarlos sin dificultades.
- 2) Creo que no, pues para acceder al parent del parent, necesito acceder primero al parent para despues al parent del parent. Así, creo que el uso de la memoria sería más eficiente si hubiese tal vez una lista con las "generaciones"de parents.