DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

Programación lógica.

Paradigmas y lenguajes de programación

Grupo: Zamba cálculo

Integrante	LU	Correo electrónico
Leandro Vega	698/11	leandrovega@gmail.com
Ignacio Niesz	722/10	ignacio.niesz@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Programación Lógica

Ejercicio 1

```
%% Ejercicio 1
%% tablero(+Filas,+Columnas,-Tablero) instancia una estructura de tablero en blanco
%% de Filas x Columnas, con todas las celdas libres.
tablero(N,M,T) :- length(T, N), freeVarLists(M, T).

%% freeVarLists(+Length, -List)
freeVarLists(_,[]).
freeVarLists(Length, [X|XS]) :- length(X, Length), freeVarLists(Length, XS).
```

Ejercicio 2

```
%% Ejercicio 2
%% ocupar(+Pos,?Tablero) sera verdadero cuando la posicion indicada este ocupada.
ocupar(pos(Y,X), T) :- nth0(Y, T, Row), nth0(X, Row, ocupada).
```

Ejercicio 3

```
%% Ejercicio 3
%% vecino(+Pos, +Tablero, -PosVecino) sera verdadero cuando PosVecino sea
%% un atomo de la forma pos(F', C') y pos(F',C') sea una celda contigua a
%% pos(F,C), donde Pos=pos(F,C). Las celdas contiguas puede ser a lo sumo cuatro
%% dado que el robot se movera en forma ortogonal.

% Vecino izquierdo
vecino(pos(Y,X),T,PosVecino) :- XVecino is X-1,
  onBounds(XVecino, Y, T),
  PosVecino = pos(Y,XVecino).

%% Vecino superior
vecino(pos(Y,X),T,PosVecino) :- YVecino is Y-1,
  onBounds(X, YVecino, T),
  PosVecino = pos(YVecino, X).

%% Vecino derecho
vecino(pos(Y,X),T,PosVecino) :- XVecino is X+1,
  onBounds(XVecino, Y, T),
  PosVecino = pos(Y,XVecino).

%% Vecino inferior
vecino(pos(Y,X),T,PosVecino) :- YVecino is Y+1,
  onBounds(X, YVecino, T),
  PosVecino = pos(YVecino, X).

%% onBounds(+X,+Y,+T)
onBounds(X,Y,T) :- rows(T,YLimit), columns(T,XLimit), X >= 0, Y >= 0, X < XLimit, Y < YLimit.

%% rows(+Tablero, -Filas)
rows(T,Rows) :- length(T, Rows).
%% columns(+Tablero, -Columnas)
columns(T,Columns) :- nth0(0, T, FirstRow), length(FirstRow, Columns).
```

Ejercicio 4

```
%% Ejercicio 4
%% vecinoLibre(+Pos, +Tablero, -PosVecino) idem vecino/3 pero ademas PosVecino
%% debe ser una celda transitable (no ocupada) en el Tablero
vecinoLibre(Pos,T,Vecino) :- vecino(Pos,T,Vecino), celda(Vecino,T,C), var(C).

%% celda(+Pos, +Tablero, -Elemento)
celda(pos(Y,X),T,E) :- nth0(Y, T, Row), nth0(X, Row, E).
```

Ejercicio 5

```
%% Ejercicio 5
%% camino(+Inicio, +Fin, +Tablero, -Camino) sera verdadero cuando Camino sea una lista
%% [pos(F1,C1), pos(F2,C2),..., pos(Fn,Cn)] que denoten un camino desde Inicio
%% hasta Fin pasando solo por celdas transitables.
%% Ademas se espera que Camino no contenga ciclos.
%% Notar que la cantidad de caminos es finita y por ende se tiene que poder recorrer
%% todas las alternativas eventualmente.
%% Consejo: Utilizar una lista auxiliar con las posiciones visitadas

camino(I,F,T,C) :- I = pos(Y1,X1),
  F = pos(Y2,X2),
  onBounds(X1,Y1,T),
  onBounds(X2,Y2,T),
  caminoAux(I,F,T,C,[]).

caminoAux(Pos,Pos,T,C,Aux) :- not(member(Pos, Aux)),
  Pos = pos(Y,X),
  onBounds(X,Y,T),
  C = [Pos].

caminoAux(I,F,T,C,Aux) :- not(member(I,Aux)),
   vecinoLibre(I,T,NewI),
  append([I], NewPath, C),
  caminoAux(NewI,F,T,NewPath, [I|Aux]).
```

Ejercicio 6

```
%% cantidadDeCaminos(+Inicio, +Fin, +Tablero, ?N) que indique la cantidad de caminos
%% posibles sin ciclos entre Inicio y Fin.
cantidadDeCaminos(I,F,T,N) :- count(camino(I,F,T,_), N).
count(P,Count) :- findall(1,P,L), length(L,Count).
```

Ejercicio 7

```
%% camino2(+Inicio, +Fin, +Tablero, -Camino) idem camino/4 pero se espera una heuristica
%% que mejore las soluciones iniciales.
%% No se espera que la primera solucion sea necesariamente la mejor.
%% Una solucion es mejor mientras menos pasos se deba dar para llegar a
%% destino (distancia Manhattan). Por lo tanto, el predicado debera devolver de a uno,
%% todos los caminos pero en orden creciente de longitud.
camino2(I,F,T,C):- I = pos(Y1,X1),
  F = pos(Y2,X2),
  onBounds(X1,Y1,T),
  onBounds(X2,Y2,T),
  camino2Aux(I,F,T,C,[]).

```

```
%% camino2Aux(+Inicio, +Fin, +Tablero, -Camino, +Visitados)
camino2Aux(Pos,Pos,T,C,Aux) :- not(member(Pos, Aux)),
  Pos = pos(Y,X),
  onBounds(X,Y,T),
  C = [Pos].

camino2Aux(I,F,T,C,Aux) :- not(member(I,Aux)),
   findall (Pos,( vecinoLibre(I,T,Pos),
        not(member(Pos,Aux))),VecinosLibres),
  manhattanOrdered(F,VecinosLibres,VecinosOrdenados),
  append([I], NewPath, C),
  camino2AuxVecinos(VecinosOrdenados,F,T,NewPath,[I|Aux]).

%% camino2AuxVecinos(+Vecinos, +Fin, +Tablero, -Camino, +Visitados)
camino2AuxVecinos([],_,_,_,_) :- false.
camino2AuxVecinos([V|_],F,T,C,Aux) :- camino2Aux(V,F,T,C,Aux).
camino2AuxVecinos([_|Vs],F,T,C,Aux) :- camino2AuxVecinos(Vs,F,T,C,Aux).

%manhattanOrdered(+Final,+Vecinos,-VecinosOrdenados)
manhattanOrdered(F,Vs,Os) :- applyDistanceMask(F,Vs,Ms),
   predsort(manhattanCompare,Ms,OrderedMs),
  removeDistanceMask(OrderedMs,Os).

%%manhattanOrdered(+Pos1,+Pos2,-ManhattanDistance)
manhattanDistance(pos(Y1, X1), pos(Y2,X2), Distance) :-
  Distance is (abs(Y2-Y1) + abs(X2-X1)).

%%applyDistanceMask(+Destino,+Posiciones,-PosicionesYDistancia)
applyDistanceMask(_,[], D):- D = [].
applyDistanceMask(F,[P|Ps],D) :- D = [manhattan(P,MDistance) | Ms],
  manhattanDistance(P,F,MDistance),
  applyDistanceMask(F,Ps,Ms).

%%removeDistanceMask(+PosicionesYDistancias, -Posiciones)
removeDistanceMask([],Pos) :- Pos = [].
removeDistanceMask([manhattan(P,_) | Ms], Pos) :- Pos = [P | Ps],
  removeDistanceMask(Ms,Ps).

%%manhattanCompare(-Order, +PosYDist1, +PosYDist2)
manhattanCompare(>, manhattan(_,D1), manhattan(_,D2)) :-
  D1 > D2.
manhattanCompare(<, manhattan(_,D1), manhattan(_,D2)) :-
  not(D1 > D2).
manhattanCompare(=, manhattan(_,_), manhattan(_,_)) :-
   false.
```

Ejercicio 8

```
%% camino3(+Inicio, +Fin, +Tablero, -Camino) idem camino2/4 pero se espera que
%% se reduzca drasticamente el espacio de busqueda.
%% En el proceso de generar los potenciales caminos, se pueden ir sacando algunas conclusiones.
%% Por ejemplo, si se esta en la celda (3,4) y se dieron ya 6 pasos desde el Inicio,
%% entonces no tiene sentido seguir evaluando cualquier camino que implique llegar a la celda (3,4)
%% desde Inicio en mas de 6 pasos.
%% Notar que dos ejecuciones de camino3/4 con los mismos argumentos deben dar los mismos resultados.
%% En este ejercicio se permiten el uso de predicados: dynamic/1, asserta/1, assertz/1 y retractall/1.
camino3(_,_,_,_).

shortestPath(P,D) :- not((minPath(P,D2), D2 < D)).
:- dynamic minPath/2.
minPath(pos(0,0),0).
```