Examen

**Listas:**

1. **Reverse**

reverse([], []).

reverse(Lista, R):- append([L1, [E]], Lista), reverse(L1, L1R),

append([E], L1R, R).

1. **Mas Veces**

mas\_veces(Lista, E, N):- msort(Lista, ListaOrd), comprime(ListaOrd, R),

mayor(R, E, N).

mayor([], \_, 0).

mayor([(E,N)|R], E, N):- mayor(R, \_, N2), N >= N2.

mayor([(\_,N)|R], E2, N2):- mayor(R, E2, N2), N < N2.

1. **Elimina cada N**

elimina\_n(Lista, N, Lista):- length(Lista, L), L < N.

elimina\_n(Lista, N, R2):- N2 is N-1, length(L1, N2),

append(L1, [\_|L2], Lista),

elimina\_n(L2, N, R), append(L1, R, R2).

1. **Comprime**

comprime([], []).

comprime([E], [(E,1)]).

comprime([Cab, Cab|Resto], [(Cab, N2)|R] ):-

comprime([Cab|Resto], [(Cab,N)|R]), N2 is N + 1.

comprime([Cab1, Cab2|Resto], [(Cab1,1)|R]):- Cab1 \= Cab2, comprime([Cab2|Resto], R).

**Árboles**

1. **Lista Hojas (binario/generico)**

lista\_hojas(a(Et, []), [Et]).

lista\_hojas(a(\_, ListaHijos), R):- ListaHijos \= [], lista\_hojas(ListaHijos, R).

lista\_hojas([], []).

lista\_hojas([Cab|Resto], R):- lista\_hojas(Resto, RResto), lista\_hojas(Cab, RCa),

append([RCa, RResto], R).

arbol\_1( a(1, [a(2, [a(3,[]), a(4,[]), a(5,[])]), a(6, []), a(7, [a(8,[]), a(9,[])])]) ).

1. **Balanceado**

balanceado(nil).

balanceado(a(\_, HI, HD)):-

altura(HI, AI),

altura(HD, AD),

Dif is abs(AI - AD), Dif =< 1,

balanceado(HI),

balanceado(HD).

altura(nil, 0).

altura(a(\_, HI, HD), R):- altura(HI, AI), altura(HD, AD), A is max(AI, AD), R is A + 1.

arbol\_2( a(1, a(2, a(3, a(6, nil, nil), nil), nil), a(4, nil, a(5, nil, a(7, nil, nil)))) ).

arbol\_3( a(1, a(2, a(3, nil, nil), nil), a(4, nil, a(5, nil, nil))) ).

1. **Anchura (binario/generico)**

anchura(AB, R):- bin2gen(AB, AG), anchura(AG, R).

anchura(a(Et, ListaHijos), [Et|R]):- anchura(ListaHijos, R).

anchura([], []).

anchura([a(Et,ListaHijos)|Resto], [Et|R2]):- append(Resto, ListaHijos, R), anchura(R, R2).

bin2gen(a(Et, nil, nil), a(Et, [])).

bin2gen(a(Et, HI, HD), a(Et, [RI, RD])):- HI \= nil, HD \= nil, bin2gen(HI, RI), bin2gen(HD, RD).

bin2gen(a(Et, nil, HD), a(Et, [RD])):- HD \= nil, bin2gen(HD, RD).

bin2gen(a(Et, HI, nil), a(Et, [RI])):- HI \= nil, bin2gen(HI, RI).

**Grafos**

1. **Ciclos**

ciclo(Grafo, V, Ciclos):- findall(Camino, camino(Grafo, V, V, [], Camino), Ciclos).

ciclos(G, Ciclos):- G = g(Vertices, \_), member(V, Vertices),

findall(Ciclo, ciclo(G, V, Ciclo), Ciclos).

1. **Camino**

camino(EstadoIni, EstadoIni, \_, []).

camino(EstadoIni, EstadoFin, Visitados, [Mov|Camino]):-

mov(Mov, EstadoIni, EstadoTMP),

\+ member(EstadoTMP, Visitados),

camino(EstadoTMP, EstadoFin, [EstadoTMP|Visitados], Camino).

**Problemas de estados**

1. **Jarras**

* Definir el estado

estado(L5, L3)

* Definir estado inicial y final

Estado inicial = estado(0, 0)

Estado final = estado(4, \_)

* Definir los movimientos

mov(Nombre, EstadoAnterior, EstadoPosterior)

* Llenar 5L o 3L

mov(llenar\_5l, estado(\_, L3), estado(5, L3) ).

mov(llenar\_3l, estado(L5, \_), estado(L5, 3) ).

* Vaciar 5L o 3L

mov(vaciar\_5l, estado(\_, L3), estado(0, L3) ).

mov(vaciar\_3l, estado(L5, \_), estado(L5, 0) ).

* Pasar 5L a 3L o 3L a 5L

mov(pasar\_5la3l, estado(L5, L3), estado(0, T) ):- T is L5 + L3, T =< 3.

mov(pasar\_5la3l, estado(L5, L3), estado(L52 , 3) ):- T is L5 + L3, T > 3,

L52 is T - 3.

* Pasar 3L a 5L

mov(pasar\_3la5l, estado(L5, L3), estado(T, 0) ):- T is L5 + L3, T =< 5.

mov(pasar\_3la5l, estado(L5, L3), estado(5, L32) ):- T is L5 + L3, T > 5,

L32 is T - 5.

**CLP**

1. **Sudoku**

sudoku(Rows) :-

length(Rows, 9),

maplist(same\_length(Rows), Rows),

append(Rows, Vs), Vs ins 1..9,

maplist(all\_distinct, Rows),

transpose(Rows, Columns),

maplist(all\_distinct, Columns),

Rows = [As,Bs,Cs,Ds,Es,Fs,Gs,Hs,Is],

blocks(As, Bs, Cs),

blocks(Ds, Es, Fs),

blocks(Gs, Hs, Is).

blocks([], [], []).

blocks([N1,N2,N3|Ns1], [N4,N5,N6|Ns2], [N7,N8,N9|Ns3]) :-

all\_distinct([N1,N2,N3,N4,N5,N6,N7,N8,N9]),

blocks(Ns1, Ns2, Ns3).

1. **Planificador de horarios**