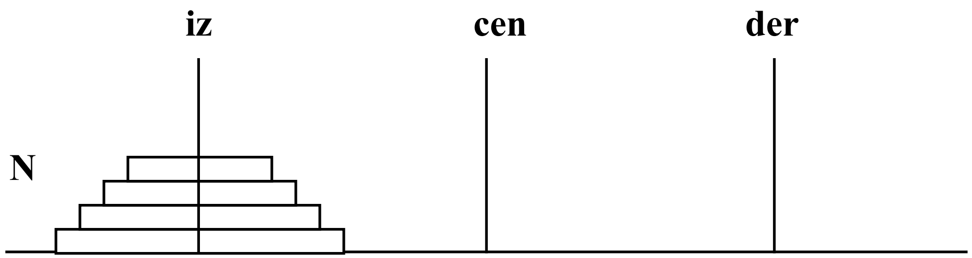
Tema 5. EJEMPLOS de PROGRAMAS

TEMA 5. EJEMPLOS de PROGRAMAS

En este tema se presentan 4 ejemplos para ilustrar cómo se representan y solucionan problemas conocidos en PROLOG.

5.1 Las Torres de Hanoi



El problema consiste en trasladar una torre de **N** discos de la varilla izquierda (**iz**) a la derecha (**der**) con ayuda de la varilla central (**cen**), con las dos restricciones siguientes: **a)-** Mover un disco en cada movimiento. **b)-** No colocar nunca un disco sobre otro menor.
La solución óptima se consigue en $2^N - 1$ movimientos. Deseamos conocer la lista de movimientos necesaria.

Representación del problema:
hanoi(N, A, B, C, L): "L es una lista de movimientos para trasladar una torre de N discos de A a B con ayuda de C"
mov(A,B): "El movimiento de pasar el disco más alto de A a B" (paso de un disco)

Programa:
```
hanoi(1, A, B, C, [mov(A,B)]).
hanoi(N, A, B, C, L) :-  N>1, M is N-1, hanoi(M, A, C, B, L1), hanoi(M, C, B, A, L2), concatenar(L1, [mov(A,B)|L2], L).
```

5.2 Reconocedor de Polinomios

El programa consiste en reconocer una expresión dada como un polinomio de una incógnita.

Representación:

polinomio (E,X): "La expresión E es un polinomio en la incógnita X"

(La expresión $x^2 - 3x + 2$ se representará en Prolog mediante el término **X^2 - 3*X + 2**, usando declaración de operadores).

Programa:

```
:- op(12, yfx, '+').
:- op(12, yfx, '-').
:- op(10, yfx, '*').      % declaración de operadores
:- op(10, yfx, '/').
:- op( 8, xfx, '^').
polinomio(Y,X) :- var(Y), ! , Y==X.
polinomio(T,X) :- constante(T).
polinomio(T1+T2, X) :- polinomio(T1,X), polinomio(T2,X).
polinomio(T1-T2, X)  :- polinomio(T1,X), polinomio(T2,X).
polinomio(T1*T2, X)  :- polinomio(T1,X), polinomio(T2,X).
polinomio(T1/N, X)  :- polinomio(T1,X), constante(N).
polinomio(T^N, X)  :-  polinomio(T,X), constante(N).
constante(N)  :-  integer(N).
```

Preguntas:

```
? polinomio(X^2 - 3*X + 2, X).          ? polinomio(X^2 - 3*Y + 2, X).
 si    (X = H3)                           no
```

5.3 Consulta a un Diccionario

El problema consiste en representar en Prolog un diccionario y las operaciones de crearlo, consultar una palabra y traducir un texto.

Representación:

Para representar el diccionario usaremos árboles binarios ordenados cuyos nodos consistan en pares de palabras (una palabra y su traducción). El orden vendrá dado por una clave de los nodos (la primera palabra del par) y un orden dado (el orden alfabético).

1. *Representación general de un árbol binario:*

```
arb_binario(vacio).
arb_binario(arb(R, HI, HD))  :- nodo(R),  arb_binario(HI), arb_binario(HD).
```

2. *Ordenación general según una clave (de los nodos) y un orden dado:*

```
ordenado(vacio, _ ).
ordenado(arb(R, HI, HD), Orden) :-        clave(R,X),  mayort(X, HI, Orden), menort(X, HD, Orden),
                                          ordenado(HI, Orden), ordenado(HD, Orden).
menort(X, vacio, _ ).
menort(X, arb(R, HI, HD), Orden) :-       clave(R,Y), menor(X, Y, Orden), menort(X, HI, Orden), menort(X, HD, Orden).
```

("mayort" será similar)

3. *Representación del diccionario:*

```
diccionario(D) :- arb_binario(D), ordenado(D, alf).
nodo(item(P,T)).
clave(item (P,T), P).
menor(X, Y, alf) :- X @< Y.
mayor (X, Y, alf) :- X @> Y.
```

4. *Operación "insertar" palabra en un diccionario:*

```
insertar(X, vacio, arb(X, vacio, vacio)).
insertar(X, arb(X, HI, HD), arb(X, HI, HD)).
insertar(item(P,T), arb(item(P1,T1), HI, HD), arb(item(P1,T1), NHI, HD))  :- menor(P, P1, alf), insertar(item(P,T), HI, NHI).
insertar(item(P,T), arb(item(P1,T1), HI, HD), arb(item(P1,T1), HI, NHD))  :- mayor(P, P1, alf), insertar(item(P,T),HD,NHD).
```

5. *Operación "consultar" palabra en un diccionario obteniendo su traducción:*

```
consulta(P, arb(item(P,T), HI, HD), T).
consulta(P, arb(item(OP,OT), HI, HD), T)  :- menor(P, OP, alf), consulta(P, HI, T).
consulta(P, arb(item(OP,OT), HI, HD), T)  :- mayor(P, OP, alf), consulta(P, HD, T).
```

6. *Operación traducción de un texto: "traduce (Texto, Diccionario, Traducción)"*

```
traduce([ ], _ , [ ]).
traduce([PP | RTX], D, [PT | RTD]  :- consulta(PP, D, PT), traduce(RTX, D, RTD).
```

5.4 Búsqueda en Espacio de Estados

Un espacio de estados se caracteriza por los elementos siguientes:

- Un estado inicial
- Un conjunto de estados finales
- Un conjunto de movimientos

El problema consiste en encontrar una sucesión de movimientos que genere un camino desde el estado inicial a un estado final.

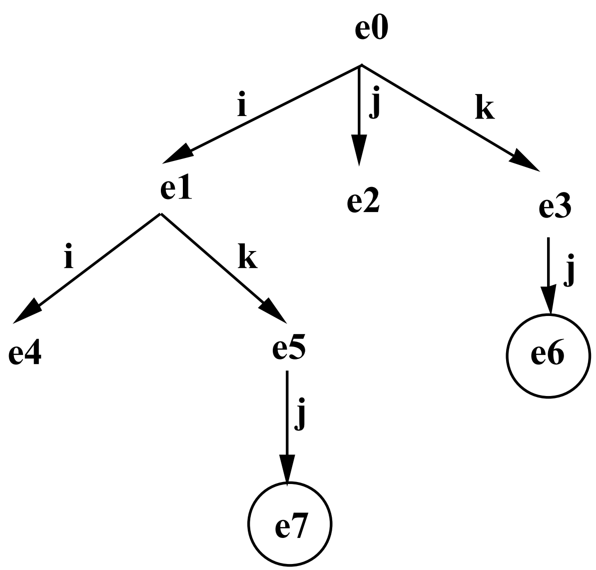
Representación:

inicial(E) : "E es un estado inicial"

final(E) : "E es un estado final"

movim(E, M, E') : "M es un movimiento que transforma el estado E en el estado E´"

Ejemplo:



inicial(e0).
final(e6).
final(e7).
movim(e0, i, e1).
movim(e0, j, e2).
movim(e0, k, e3).
movim(e1, i, e4).
movim(e1, k, e5).
movim(e3, j, e6).
movim(e5, j, e7).

El programa es independiente de la representación del espacio de estados. Veremos dos formulaciones distintas del programa.

***1ª formulación*:**

Elige de forma no determinista una longitud y trata de encontrar soluciones de esa longitud. Encuentra primero las soluciones más cortas (recorrido del árbol en anchura). El programa diverge intentando encontrar soluciones de mayor longitud no existentes.

Programa:

```
solucion1(S) :- genera(LM, [F | LE]), final(F), inversa(LM, S).
genera([ ], [E]) :- inicial(E).
genera([M | RM], [NE, E | RE]) :- genera(RM, [E | RE]),
                                  movim(E, M, NE),
                                  not(member(NE, [E | RE])).
```

En el ejemplo dado:

? solucion1(S)
S = [k, j] ;
S = [i, k, j] ;
diverge

***2ª formulación*:**

Trata de extender una solución parcial, comenzando por el estado inicial, a una solución. El efecto es el de una búsqueda en profundidad y por la izquierda. El programa termina si el espacio de estados es finito.

Programa:

```
solucion2(S)  :- inicial(E), prolongable([ ], [E], LM, LE), inversa(LM, S).
prolongable(LM, [E | RE], LM, [E | RE]) :- final(E).
prolongable(LM, [E | RE], LMP, LEP) :-  movim(E, M, NE),
                                        not(member (NE, [E | RE])),
                                        prolongable([M | LM], [NE, E | RE], LMP, LEP).
```

En el ejemplo dado:

? solucion2(S)
S = [i, k, j] ;
S = [k, j] ;
no