

1. Ejercicio de la familia simpson

Listas y estructuras

```
1 % persona(Nombre, Sexo, Pareja, Padres)
2 persona(abraham,hombre,mona,[]).
3 persona(mona,mujer,abraham,[]).
4 persona(clancy,hombre,jacqueline,[]).
5 persona(jacqueline,mujer,clancy,[]).
6 persona(herbert,hombre,_,[abraham,mona]).
7 persona(homero,hombre,marge,[abraham,mona]).
8 persona(marge,mujer,homero,[clancy,jacqueline]).
9 persona(patty,mujer,_,[clancy,jacqueline]).
10 persona(selma,mujer,_,[clancy,jacqueline]).
11 persona(bart,hombre,_,[homero,marge]).
12 persona(lisa,mujer,_,[homero,marge]).
13 persona(maggie,mujer,_,[homero,marge]).
14 persona(ling,mujer,_,[selma]).
15
```

REGLAS

```
% reglas
hombre(X):-persona(X,hombre,_,_).
mujer(X):-persona(X,mujer,_,_).
es_progenitor(P,H):-persona(H,_,_,L),member(P,L).
es_padre(P,H):-es_progenitor(P,H),hombre(P).
es_madre(M,H):-es_progenitor(M,H),mujer(M).
es_hijo(H,P):-es_progenitor(P,H),hombre(H).
es_hija(H,P):-es_progenitor(P,H),mujer(H).





es_hermano(H,X):-persona(X,_,_,L),member(P,L),es_progenitor(P,H),X\==H,hombre(H).
es_hermana(H,X):-persona(X,_,_,L),member(P,L),es_progenitor(P,H),X\==H,mujer(H).





es_abuelo(A,N):-es_progenitor(A,X),es_progenitor(X,N),hombre(A).
es_abuela(A,N):-es_progenitor(A,X),es_progenitor(X,N),mujer(A).


es_tio(T,S):-es_progenitor(P,S),es_hermano(T,P),hombre(T).
es_tia(T,S):-es_progenitor(P,S),es_hermana(T,P),mujer(T).

es_sobrino(S,T):-es_progenitor(P,S),(es_hermano(P,T);es_hermana(P,T)),hombre(S).
es_sobrina(S,T):-es_progenitor(P,S),(es_hermano(P,T);es_hermana(P,T)),mujer(S).
```

DEMOSTRACIONES

 <code>es_padre(Padre,homero)</code> Padre = abraham false	 <code>es_padre(Padre,marge)</code> Padre = clancy false
?- <code>es_padre(Padre,homero)</code>	?- <code>es_padre(Padre,marge)</code>
 <code>es_madre(Madre,patty)</code> Madre = jacqueline	 <code>es_madre(Madre,bart)</code> Madre = marge
?- <code>es_madre(Madre,patty)</code>	?- <code>es_madre(Madre,bart)</code>

 <code>es_esposo(Esposo,marge)</code> Esposo = homero	 <code>es_esposo(Esposo,mona)</code> Esposo = abraham
?- <code>es_esposo(Esposo,marge)</code>	?- <code>es_esposo(Esposo,mona)</code>
 <code>es_esposa(Esposa,clancy)</code> Esposa = jacqueline	 <code>es_esposa(Esposa,homero)</code> Esposa = marge
?- <code>es_esposa(Esposa,clancy)</code>	?- <code>es_esposa(Esposa,homero)</code>

 `es_hijo(Hijo,homero)`

Hijo = bart

Next 10 100 1,000 Stop

?- `es_hijo(Hijo,homero)`

 `es_hijo(Hijo,abraham)`

Hijo = homero

Hijo = herbert

?- `es_hijo(Hijo,abraham)`


 `es_hija(Hija,clancy)`

Hija = marge

Hija = selma

Hija = patty

?- `es_hija(Hija,clancy)`

 `es_hija(Hija,homero)`

Hija = lisa

Hija = magie


?- `es_hija(Hija,homero)`

 `es_hermano(Hermano,homero)`

Hermano = herbert

Next 10 100 1,000 Stop

?- `es_hermano(Hermano,homero)`

 `es_hermana(Hermana,bart)`

Hermana = lisa

Hermana = magie

Next 10 100 1,000 Stop

?- `es_hermana(Hermana,bart)`

Hermana = selma


Hermana = patty

?- `es_hermana(Hermana,marge)`

Hermana = magie

?- `es_hermana(Hermana,lisa)`

Sobrino = bart
false
?- es_sobrino(Sobrino,herbert)

Sobrino = bart
false 
?- es_sobrino(Sobrino,selma)

Sobrino = bart
Sobrino = bart
Sobrino = magie
Sobrino = magie
false
?- es_sobrino(Sobrino,herbert)



Sobrino = bart
Sobrino = bart
Sobrino = magie
Sobrino = magie
Sobrino = ling
Sobrino = ling
?- es_sobrino(Sobrino,patty)

Tio = herbert
false
?- es_tio(Tio,bart)

Tio = herbert
false
?- es_tio(Tio,magie)

Tia = selma
Tia = patty
false
?- es_tia(Tia,bart)

Tia = marge
Tia = patty
false
?- es_tia(Tia,ling)

 <code>es_abuelo(Abuelo,bart)</code> Abuelo = abraham Abuelo = clancy false	 <code>es_abuelo(Abuelo,lisa)</code> Abuelo = abraham Abuelo = clancy false
<code>?- es_abuelo(Abuelo,bart)</code>	<code>?- es_abuelo(Abuelo,lisa)</code>
Abuela = mona Abuela = jacqueline false	Abuela = jacqueline alse
<code>?- es_abuela(Abuela,lisa)</code>	<code>?- es_abuela(Abuela,ling)</code>

Nieto = bart false	Nieto = bart false
<code>?- es_nieto(Nieto,clancy)</code>	<code>?- es_nieto(Nieto,abraham)</code>
 <code>es_nieta(Nieta,clancy)</code> Nieta = lisa Nieta = magie Nieta = ling	 <code>es_nieta(Nieta,abraham)</code> Nieta = lisa Nieta = magie alse
<code>?- es_nieta(Nieta,clancy)</code>	<code>?- es_nieta(Nieta,abraham)</code>

2. Ejercicio de rutas entre ciudades de Canada

Listas y estructuras

```
1 % =====
2 % RUTAS ENTRE CIUDADES DE CANADÁ
3 % =====
4
5 ciudad(vancouver, [conexion(edmonton,16), conexion(calgary,13)]).
5 ciudad(edmonton, [conexion(saskatoon,12), conexion(calgary,4)]).
7 ciudad(calgary, [conexion(regina,14), conexion(edmonton,4)]).
3 ciudad(saskatoon, [conexion(calgary,9), conexion(winnipeg,20)]).
9 ciudad(regina, [conexion(winnipeg,4), conexion(saskatoon,7)]).
9 ciudad(winnipeg, []).
```

```
1
```

REGLAS

```
%reglas
existe_conexion(O,D):- ciudad(O,L),member(conexion(D,_),L).
valor(O,D,C):- ciudad(O,L),member(conexion(D,C),L).
existe_arista(A,B):- existe_conexion(A,B);existe_conexion(B,A).
costo_intermedio(A,C,T):- valor(A,B,C1),valor(B,C,C2),T is C1+C2.
es_posible_viajar(A,C):- existe_conexion(A,B),existe_conexion(B,C).
costo_viaje(O,D,C):- valor(O,D,C).
costo_viaje(O,D,C):- valor(O,X,C1),costo_viaje(X,D,C2),C is C1+C2.
```


DEMOSTRACIONES

 `existe_conexión(saskatoon, Destino)`

Destino = calgary

Destino = winnipeg

?- `existe_conexión(saskatoon, Destino)`

 `valor(calgary, Destino, Costo).`

Costo = 14,

Destino = regina

Costo = 4,

Destino = edmonton

?- `valor(calgary, Destino, Costo).`

 `existe_arista(calgary, Existe)`

Existe = regina

Existe = edmonton

Existe = vancouver

Existe = saskatoon

?- `existe_arista(calgary, Existe)`


```
es_posible_viajar(edmonton, _, calgary).  
true  
es_posible_viajar(edmonton, T, calgary).  
T = saskatoon  
?- es_posible_viajar(edmonton, T, calgary).
```

Conclusiones:

El ejercicio permitió comprender cómo las estructuras y listas en Prolog facilitan la organización de información compleja, como las relaciones familiares. Representar la familia Simpson mediante hechos estructurados hizo que las consultas fueran más claras y el código más flexible. Además, se reforzó el razonamiento lógico al aplicar reglas para deducir parentescos sin usar programación imperativa, lo que demuestra la potencia del paradigma lógico.

En el ejercicio de las ciudades, se aplicaron estructuras y listas para modelar conexiones y rutas entre distintas localidades. Esto permitió calcular costos de viaje y verificar posibles trayectos de manera más eficiente. La experiencia ayudó a entender cómo la recursividad y el uso de listas hacen posible resolver problemas reales de rutas y caminos, reforzando la lógica de búsqueda y el pensamiento estructurado en Prolog.

ProgIIIIG101-Act05-

Santiago Henao

-Johany Ballesteros

-Juan Guillermo Galindo

<https://github.com/johany-ballesteros/ProgIIIIG101-Act01-Santiago-Henao-Johany-Ballesteros-Juan-Guillermo-Galindo>