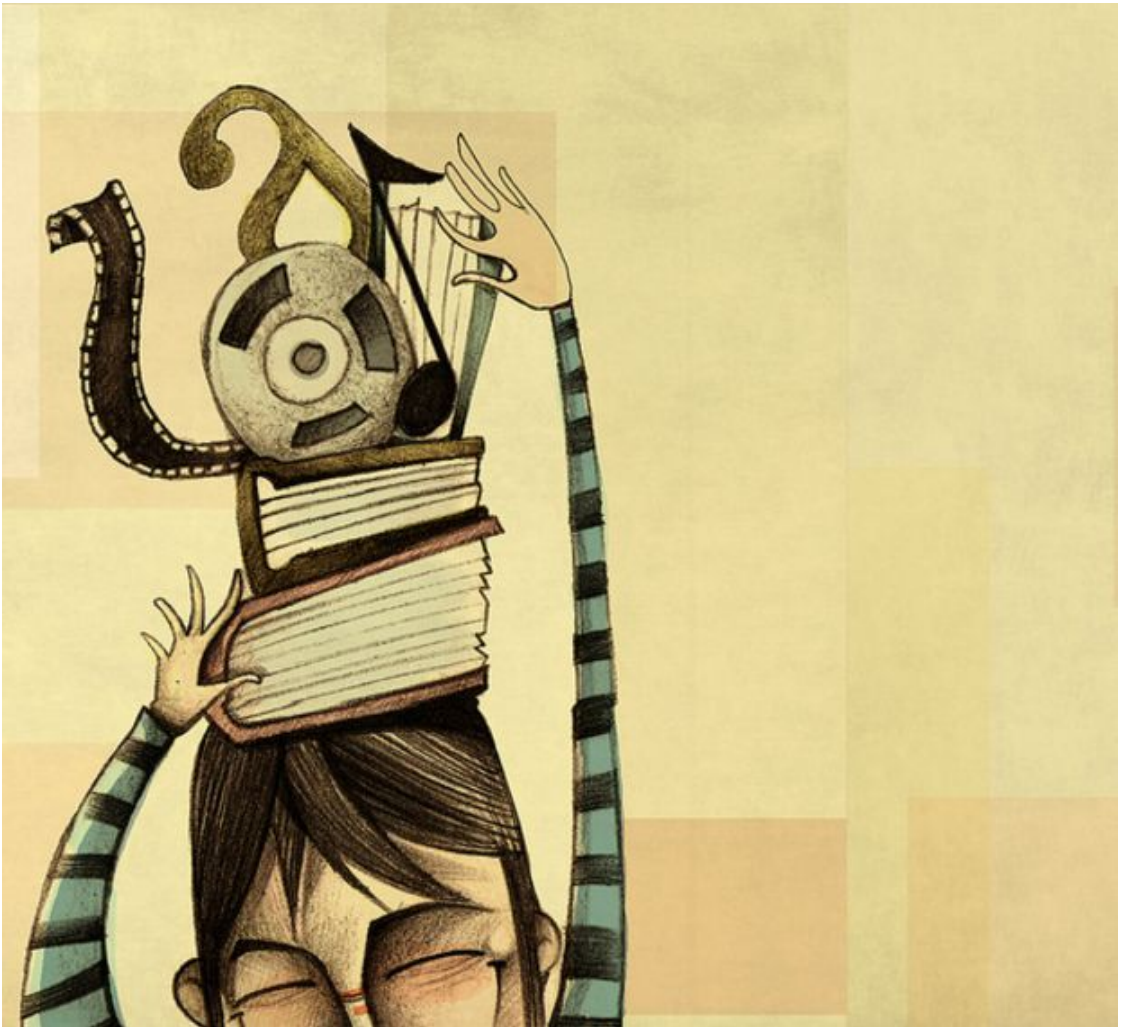
INFORME PROLOG

Recomendador



Juan Jesús Padrón Hernández

Daniel Valle Rodríguez

16/12/2017

INDICE

INTRODUCCIÓN

Ante la propuesta de implementar un **sistema basado en conocimiento**, se nos ocurrió desarrollar un **recomendador de música, películas y libros.**

La idea es que el sistema disponga una **base de conocimiento** que contenga diferentes artistas musicales, películas y libros, además de sus respectivos géneros. De esta forma, el **usuario** del sistema introducirá **qué** quiere que se le recomiende y los **géneros** que le interesan, generando así una **lista de recomendaciones** ordenadas según el número de géneros en común con los elementos (películas, música o libros) de la base de conocimientos.

Para que un elemento sea recomendado, como mínimo tiene que tener un género en común con los gustos introducidos por el usuario.

BASE DE CONOCIMIENTO

La **base de conocimiento** de nuestro sistema, a parte de las reglas que se explicarán en apartados posteriores, ha de contener **información** de las películas, libros o música que se va a recomendar. Estas sentencias son de la forma:

pelicula(nombreDeLaPelicula).

director(nombreDeLaPelicula,directorDeLaPelicula).

genero(nombreDeLaPelicula,[listaDeGenerosSeparadosPorComas]).

libro(nombreDelLibro).

autor(nombreDelLibro,autorDelLibro).

genero(nombreDelLibro,[listaDeGenerosSeparadosPorComas]).

musica(cantanteOGrupo).

genero(cantanteOGrupo,[listaDeGenerosSeparadosPorComas]).

IMPLEMENTACIÓN

En primer lugar, incluimos la **librería pairs**, la cual nos permitirá manejar de forma adecuada pares de valores.

```
:- use_module(library(pairs)).
```

La sentencia principal del programa es **recomienda/3**, la cual recibe como parámetros una constante que indica qué quieres que te recomiende (musica,libro,pelicula), una lista de constantes que contendrá los géneros que les interesa al usuario y, por último, una variable donde se almacenará la recomendación.

```
recomienda(musica,Gustos,Recomendacion):-
    findall(Musica,musica(Musica),Musicos),
    make_recomendation(Gustos,Musicos,Recomendaciones),
    member(Recomendacion,Recomendaciones).

recomienda(libro,Gustos,Recomendacion):-
    findall(Libro,libro(Libro),Libros),
    make_recomendation(Gustos,Libros,Recomendaciones),
    member(Recomendacion,Recomendaciones).

recomienda(pelicula,Gustos,Recomendacion):-
    findall(Pelicula,pelicula(Pelicula),Peliculas),
    make_recomendation(Gustos,Peliculas,Recomendaciones),
    member(Recomendacion,Recomendaciones).
```

Como podemos observar, tenemos tres sentencias recomienda, una para cada tipo de recomendación.

Esta regla, primero busca todos los elementos que se encuentren en la base de conocimiento y crea una **lista** con ellos.

A continuación, se emplea la regla **make_recomendation/3**, la cual devuelve una lista de recomendaciones hechas de acuerdo con los gustos introducidos por el usuario (más adelante se explica con detalle esta función). Por último, selecciona un elemento de esta para que unifique con la constante **Recomendacion**.

```
make_recomendation(Gustos,[],[]).                                    %Función para recursividad.
make_recomendation(Gustos,Elementos,SortedRecomendaciones):-
    findall(Calidad-Elemento,(member(Elemento,Elementos),calidad(Elemento,Gustos,Calidad),Calidad>0),Pairs),
    %Crea una lista de pares Calidad-Elemento donde Calidad > 0 (al menos tiene que tener un género en común p
    sort(Pairs,AuxPairs),                                            %Ordena la lista de acuerdo con la Calidad pe
    invert(AuxPairs,SortedPairs),                                    %Invierte la lista para que quede ordenado de
    pairs_values(SortedPairs,SortedRecomendaciones).                 %Crea una lista solo con las peliculas, elimi
```

La regla **make_recomendation/3** se encarga de crear con la sentencia **findall/3** una lista de **Calidad-Elemento**, donde se ha de cumplir que los elementos en ella tienen que ser miembros de la lista creada en **recomienda/3**, la **Calidad** se calcula contando cuántos géneros tienen en común y, además, han de tener **Calidad > 0**, es decir, al menos tienen que tener un género en común para que aparezca en la lista de recomendaciones. Esta lista, denominada **Pairs**, es ordenada con **sort/2** y, luego, invertida con **invert/2**, porque

sort ordena en el orden contrario del que deseamos. Por último, **pairs_values/2** crea una lista con los valores de los pares en la lista ya mencionada, siendo esta la que llega a **recomienda/3**.

```
    findall(Libro,libro(Libro),Libros),
    make_recomendation(Gustos,Libros,Recom
    member(Recomendacion,Recomendaciones).

recomienda(pelicula,Gustos,Recomendacion):
```

La última regla que compone el núcleo de nuestro sistema es **calidad/3**, la cual recibe un **Elemento** y obtiene la lista de **Genero**s de este. Esta lista es intersectada con la de lista de **Gustos** del usuario y a continuación, se calcula el tamaño de la intersección con **tam/2**, almacenando este en la variable **Calidad**.

OTRAS REGLAS EMPLEADAS

En el código del sistema se han implementado diferentes **reglas** que usamos como **herramientas** para las funcionalidades principales del mismo. Algunas de estas ya las hemos visto en el apartado anterior. A continuación se hará un pequeño resumen de la funcionalidad de cada una:

my_intersect/3: devuelve la intersección de dos listas (elementos comunes).
invert/2: invierte el orden de la lista.
addend/3: añade elemento al final de una lista.
tam/2: devuelve el tamaño de una lista.

DIFICULTADES

Las mayores dificultades que se han tenido durante el desarrollo de este proyecto han sido por nuestra inexperiencia, no solo con **Prolog** y las peculiaridades que tiene este lenguaje, sino con la **programación declarativa** en sí.

En primer lugar, nos costó elegir una idea que desarrollar. Nos costaba entender los ejemplos que encontrábamos o, si los entendíamos, tenían poco margen de adaptación. Por ello optamos a iniciar el proyecto desde cero.

En segundo lugar, mientras desarrollábamos el código, nos costaba adaptarnos a la nomenclatura del lenguaje, teniendo que estar consultando continuamente diferentes páginas, manuales, tutoriales,etc...

CONCLUSIÓN

Como **conclusión** a este proyecto, queremos mencionar un par de conceptos que creemos importantes, uno referente a la **sistema desarrollado** en sí y otro respecto al **desarrollo en general del proyecto**.

Respecto al **sistema desarrollado**, se ha de notar que, siendo un **recomendador**, necesita de una **amplia base de datos** para un mejor funcionamiento, es decir, no podemos recomendar algo de lo que no tenemos información. A pesar de esto, creemos que este sistema si que puede tener alguna **utilidad a pequeña escala**, almacenando en su base de conocimiento las películas, libros y música que personalmente nos gustan, y, por tanto, convirtiéndolo en un **recomendador personal** que, en base las cosas que nos han gustado, nos sirva para recomendar a otras personas de acuerdo con sus gustos.

Respecto al **desarrollo del proyecto**, el hecho de trabajar con un lenguaje tan diferente de lo que acostumbramos, nos ha abierto bastante la mente, ayudándonos a salir de la "zona de confort" de los lenguajes imperativos.

ORGANIZACIÓN DEL PROYECTO

Juan Jesús Padrón Hernández: Código funcional,base de datos de libros y mitad del informe.

Daniel Valle Rodríguez: Base de datos sobre películas y música y la otra mitad del informe.

BIBLIOGRAFÍA

1. *Programming in Prolog*, W.F.Clocksin, C.S.Mellish
2. www.swi-prolog.org
3. www.lawebdelprogramador.com
4. www.stackoverflow.com