TECNOLOGICO NACIONAL DE MEXICO

INSTITUTO TECNOLOGICO DE MORELIA

"José María Morelos y Pavón"

Programación Lógica Y Funcional

Profesor: Ramos Diaz J. Guadalupe

Reporte Proyecto

Haskell Chat Bot

Carlos Jahir Castro Cázares

17120151

Ingeniería en Sistemas Computacionales

Periodo Sep-Ene 2020

27 de noviembre de 2020

Tabla de contenido

Introducción

Programar un programa en el lenguaje de programación funcional Haskell, que use listas de frases y de respuestas para así dando una cadena este compare esta cadena con las frases a través de similitud de cosenos y de ahí sacar una respuesta a la cadena proporcionada, así creando un chatbot.

Chatbot

"Un chatbot es un programa informático con el que es posible mantener una conversación, tanto si queremos pedirle algún tipo de información o que lleve a cabo una acción." *(Socialmood, s. f.)*

Similitud de cosenos

La similitud de coseno es una medida de similitud entre dos vectores distintos de cero de un espacio de producto interno. Se define como el coseno del ángulo entre ellos, que también es el mismo que el producto interno de los mismos vectores normalizados para que ambos tengan longitud 1. El coseno de 0 ° es 1, y es menor que 1 para cualquier ángulo. en el intervalo (0, π] radianes. *(Similitud de coseno - Cosine similarity - qaz.wiki, s. f.)*

$$\mathrm{soft\_cosine}_1(a, b) = \frac{\sum_{i,j}^{N} s_{ij} a_i b_j}{\sqrt{\sum_{i,j}^{N} s_{ij} a_i a_j} \sqrt{\sum_{i,j}^{N} s_{ij} b_i b_j}},$$

Desarrollo del tema

Para poder realizar este programa, se siguieron los siguientes pasos que posteriormente se explicaran en código fuente.

Pasos

1. Crear una lista de frases y respuestas donde cada respuesta corresponde a una frase.
2. Se pide una cadena llamada ***query*** a través de un método.
3. Se limpian los **caracteres especiales** de la query y la lista de frases, tales como puntos, comas, signos especiales, números, acentos, etc.
4. Separar cada cadena tanto de la query como la de las frases en palabras separadas por espacio haciendo una lista y una lista de listas de cadenas que representan palabras.
5. Crear una lista de palabras llamadas **stop words**, que son palabras que no tienen significado importante para nuestro propósito.
6. Eliminar las palabras en la lista de stop words, de la lista de palabras de la query y de la lista de listas de palabras de las frases.
7. Crear una lista llamada **diccionario,** que se compone de las palabras de la lista de la query y las palabras de las listas de listas de las frases, juntas, pero sin repetirse.
8. Crear los **vectores** tanto de la query y de las frases, que es comparando el diccionario con las listas de las frases y la lista de la query, donde se cuenta cuantas veces aparece una palabra del diccionario en las frases o en la query y la lista resultante es el vector.
9. Posteriormente se crea la lista de **similitud de cosenos** donde se compara el vector de la query con los vectores de cada una de las frases.
10. Tomar el mayor valor de la similitud de cuadrados.
11. Regresar la respuesta de la frase que tenga mas similitud de la query a partir del resultado de la similitud de cosenos.

Explicación del código.

El código fue escrito para el lenguaje de programación funcional Hazel.

```
1   -- ChatBot ---
    import Data.Char ( ord, isAlpha, isPunctuation, toLower )
2   import Data.Char
```

Primero importamos la librería de Data =>Char que nos traerá las funciones de ord, isAlpha, isPunctuation y toLower que las necesitaremos posteriormente.

```
4  frases = ["Hola Buenos dias Que onda",
5            "Que piensas de Avengers o Vengadores",
6            "Cual es tu pelicula favorita de STAR WARS",
7            "Que te parece The Mandalorian",
8            "Que prefies Netflix o Disney plus",
9            "Que piensas de Harry potter",
10           "Tu pelicula favorita de Marvel",
11           "Te gustan las pelicualas mexicanas",
```

```
30 respuestas = ["Hola, Como estas ? ..",
31               "Avengers es de mis peliculas favoritas de super heroes sobre todo la tercera parte",
32               "Mi pelicula favorita es STAR WARS La venganza de los Sith",
33               "The mandalorian es una buena serie, Baby Yoda es adorable",
34               "Yo prefiero ser pirata y descargar las pelicuas de cualquiera",
35               "Harry Potter son buenas peliculas que todos deberian ver",
36               "La mejor pelicula de marvel el Avengers Infinity War con su final estoy como :D",
```

```
56 stopWords = ["de", "la", "que", "el", "en", "y", "a", "los", "del", "se", "las", "por",
57              "mas", "pero", "sus", "le", "ya", "o", "este", "si", "porque", "esta", "entre
58              "hay", "donde", "quien", "desde", "todo", "nos", "durante", "todos", "uno",
59              "e", "esto", "mi", "antes", "algunos", "que", "unos", "yo", "otro", "otras",
60              "nada", "muchos", "cual", "poco", "ella", "estar", "estas", "algunas", "algo"
61              "ellas", "nosotras", "vosotros", "vosotras", "os", "mio", "mia", "mios", "mia
62              "suyas", "nuestro", "nuestra", "nuestros", "nuestras", "vuestro", "vuestra",
63              "esta", "estamos", "estais", "estan", "este", "estes", "estemos", "esteis",
```

Se declaran las listas de frases, respuestas y stop words.

```haskell
chatbot query = let
                    -- Depuracion de las cadenas --
                    fresesDepuradas = depuracion frases
                    queryDepurada = depurarQuery query

                    -- Partir las cadenas en palabras --
                    palabrasFrases = [ words frase | frase <- fresesDepuradas ]
                    palabrasQuery = words queryDepurada

                    -- Quitar las StopWords --
                    sinSWFrases = [quitarSWFrases frase | frase <- palabrasFrases]
                    sinSWQuery = [palabra | palabra <- palabrasQuery, not (palabra `elem` stopWords)]

                    -- Creacion del diccionario --
                    diccionario = crearDiccionario sinSWQuery sinSWFrases

                    -- Crear Vectores --
                    vectorQuery = crearVector diccionario sinSWQuery
                    vectoresFrases = [ crearVector diccionario frase | frase <- sinSWFrases]

                    -- Calcular similitud de cosenos --
                    similitudes = [ simuitudCosenos vectorQuery vectorFrase  | vectorFrase <- vectoresFrases ]

                    -- Obtner el mayor de la similitudes de coseno --
                    mayorSimilitud = maximum similitudes
                    indiceMayorSimilitud = indiceElemento similitudes mayorSimilitud 0
                in
                    if(mayorSimilitud == 0) then
                       "No tengo palabras"
                    else
                        if (indiceMayorSimilitud == -1) then
                            "Ingresa una query"
                        else
                            respuestas !! indiceMayorSimilitud

```

Se declara la función principal del programa, que recibe la query posteriormente hará los pasos anteriormente mencionados, y al final regresa un resultado dependiendo de que frase tuvo mayor similitud con la query.

```haskell
-- Depuracion de las cadenas --
fresesDepuradas = depuracion frases
queryDepurada = depurarQuery query
```

Primero se depuran las frases y la query con su función respectiva

```haskell
                    -- Convertir a minusculas y quitar caracteres del español --
                    frasesMinusculas = [ depurarCadena frase | frase <- frases ]
```

Ahora en la depuración de frases, primeramente, de hace una lista intencional donde cada elemento (frase) de frases se pasa por la función depurarCadena.

```
132   -- Hacer minusculas los caracteres de una frese y quitar los caracteres del español --
      depurarCadena :: [Char] -> [Char]
133   depurarCadena cad = [ quitarCaracteresEsp  (toLower car) | car <- cad]
134
```

En depurar cadena se hace una lista intencional donde cada elemento de cad (caracter) primeramente pasa por el método toLower que transforma el carácter a su forma de minúscula y posteriormente pasa a la función de quitarCaracteresEsp

```
135   -- Quitar caracteres del español --
      quitarCaracteresEsp :: Char -> Char
136   quitarCaracteresEsp car
137                       | ord car == 225 = 'a' -- Á á --
138                       | ord car == 233 = 'e' -- É é --
139                       | ord car == 237 = 'i' -- Í í --
140                       | ord car == 243 = 'o' -- Ó ó --
141                       | ord car == 250 = 'u' -- Ú ú --
142                       | ord car == 252 = 'u' -- Ü ü --
143                       | ord car == 241 = 'n' -- Ñ ñ --
144                       | ord car == 147 = '?' -- “ --
145                       | ord car == 148 = '?' -- ” --
146                       | ord car == 172 = '?' -- ¬ --
147                       | ord car == 191 = '?' -- ¿ --
148                       | ord car == 161 = '?' -- ¡ --
149                       | ord car == 176 = '?' -- º --
150                       | ord car == 180 = '?' -- ´ --
151                       | otherwise = car
```

En la función quitarCaracteresEsp, se quitan los caracteres del español como lo son los acentos, la ñ, la ü, comillas y caracteres únicos del español, en este se pasa el carácter por la función ord, ¿que nos regresa su representación en decimal y a partir de varias comparaciones cambia las letras acentuadas por las letras normales y los caracteres los cambia por ?, si el carácter no cae en estas opciones se devuelve igual que como ingreso.

```
-- Quitar signos de puntuacion --
fresesPuntuacion = [ puntuacionCadena frase | frase <- frasesMinusculas]
```

Posterior mente regresa a depuración donde la lista resultante pasa por una lista intencional donde cada elemento(frase), pasa por la función de puntuacionCadena.

```
153    -- Quitar signos de puntuacion de una cadena --
       puntuacionCadena :: [Char] -> [Char]
154    puntuacionCadena cad = [ car | car <- cad, not (isPunctuation car)]
```

En la función puntuacionCadena, pasamos cada elemento de cad (carácter), por una lista intencional donde este se agrega a la nueva lista si el carácter no es un signo de puntuación.

```
-- Quitar los no alfanumericos --
fresesAlfa = [ alfaNumericosCadena frase | frase <- fresesPuntuacion]
```

Posteriormente regresa a depuración donde la lista resultante pasa por otra lista intencional donde cada elemento (frase), para por la función de alfaNumericosCadena

```
-- Quitar los caracteres no alfanumericos --
alfaNumericosCadena :: [Char] -> [Char]
alfaNumericosCadena cad = [ car | car <- cad, isAlpha car || car == ' ']
```

En la función alfaNumericosCadena, donde cada elemento de cad (caracter), se agrega a la nueva lista si el carácter el Alpha ósea si es una letra o si es un espacio.

Finalmente regresa a depuración donde regresa esta lista resultante.

```
116  depuracion frases = let
117                          -- Convertir a minusculas y quitar caracteres del español --
118                          frasesMinusculas = [ depurarCadena frase | frase <- frases ]
119                          -- Quitar signos de puntuacion --
120                          fresesPuntuacion = [ puntuacionCadena frase | frase <- frasesMinusculas]
121                          -- Quitar los no alfanumericos --
122                          fresesAlfa = [ alfaNumericosCadena frase | frase <- fresesPuntuacion]
123                      in
124                          fresesAlfa
```

Ahora para depurar la query se llama a la función depurarQuery.

```
depurarQuery query = let
                        queryMinusculas = depurarCadena query
                        queryPuntuacion = puntuacionCadena queryMinusculas
                        queryAlfa = alfaNumericosCadena queryPuntuacion
                    in
                        queryAlfa
```

En la función depurarQuery, primero se pasa la cadena de la query a depurarCadena, la lista resultante pasa por puntuacionCadena y finalmente la resultante pasa por alfaNumericosCadena y la lista final la regresa a la función principal, las funciones anteriores ya fueron explicadas anteriormente.

```
85                      -- Partir las cadenas en palabras --
86                      palabrasFrases = [ words frase | frase <- fresesDepuradas ]
87                      palabrasQuery = words queryDepurada
```

Ahora en la función principal, se pasa la lista de frases depuradas en una lista intencional donde cada elemento (una frase), pasa por la función words que nos convierte la frase en una lista de palabras tomando como separador los espacios en blanco, y de aquí obtenemos una lista de listas de palabras y la query depurada pasa por la función words para separar la cadena en una lista de palabras.

```
89                      -- Quitar las StopWords --
90                      sinSWFrases = [quitarSWFrases frase | frase <- palabrasFrases]
```

Ahora quitamos las stop words, donde pasamos por una lista intencional la lista de palabras de frases, donde cada elemento (lista de palabras), pasa por el método quitarSWFrases

```
159 -- Quitar las stop words de las frases --
    quitarSWFrases :: [[Char]] -> [[Char]]
160 quitarSWFrases frase = [ palabra | palabra <- frase, not (palabra `elem` stopWords)]
```

En el método quitarSWFrases, pasamos por una lista intencional la lista de frases, donde cada elemento (palabra) se une a la nueva lista si esta palabra no es elemento de la lista de stopWords.

```
91          sinSWQuery = [palabra | palabra <- palabrasQuery, not (palabra `elem` stopWords)]
```

Regresamos a la función principal, donde la lista de palabras de la query, pasa por una lista intencional donde cada elemento (palabra), se une a la nueva lista si esta palabra no pertenece a la lista de stopWords.

```
93          -- Creacion del diccionario --
94          diccionario = crearDiccionario sinSWQuery sinSWFrases
```

Ahora regresamos a la función principal, donde creamos el diccionario con las listas sin stop words de las frases y la query, a través de la función crearDiccionario.

```
162 -- Creacion del diccionario --
    crearDiccionario :: Eq a => [a] -> [[a]] -> [a]
163 crearDiccionario listaQuery listaFrases =   let
164                                                 frasesUnidas = unirFrases listaFrases
165                                                 diccionarioRepetido = listaQuery ++ frasesUnidas
166                                                 diccionario = quitarRepetidos diccionarioRepetido
167                                             in
168                                                 diccionario
```

En la funcion de crear diccionario resivimos las listas de frases y la lista de la query, donde primeramente unimos las palabras de las listas de las frases en una sola lista con la función unirFrases

```
164                                         frasesUnidas = unirFrases listaFrases
```

```
170 💡 Unir frases en una sola lista --
    unirFrases :: [[a]] -> [a]
171 unirFrases [] = []
172 unirFrases (x:xs) = x ++ (unirFrases xs)
```

En la función de unir frases donde el caso base es una lista vacía que regresa una lista vacía, y si recibe una lista une el primer elemento (una lista de palabras) con las listas de palabras del resto.

```
165                                     diccionarioRepetido = listaQuery ++ frasesUnidas
```

Posteriormente unimos la lista de palabras de la query, con la lista de frases unidas que obtuvimos anteriormente.

```
166                                    diccionario = quitarRepetidos diccionarioRepetido
```

Por último, quitamos las palabras repetidas de la lista de palabras unidas de frases y la lista de palabras de la query con la función quitarRepetidos

```
174   -- Quitar los elementos repetidos del diccionario --
      quitarRepetidos :: Eq a => [a] -> [a]
175   quitarRepetidos [] = []
176   quitarRepetidos (x:xs) = if (x `elem` xs) then
177                               quitarRepetidos xs
178                            else
179                               x:quitarRepetidos xs
```

En la función quitarRepetidos tenemos como caso base una lista vacía que regresa una lista vacía y de lo contrario si es una lista se analiza el primer elemento; Si el primer elemento se encuentra en el resto de la lista, seguimos analizando el resto de la lista y si no se agrega el primer elemento y se sigue analizando el resto.

```
167                                                in
168                                                   diccionario
```

Esta lista se regresa a la función de crearDiccionario y la misma regresa a la lista de la función principal.

```
96    -- Crear Vectores --
97    vectorQuery = crearVector diccionario sinSWQuery
```

Ahora regresando a la función principal para crear el vector de la query, donde usamos la función de crearVector y pasamos el diccionario y la lista de palabras de la query.

```
181   -- Crear un vector --
      crearVector :: (Num a, Eq t) => [t] -> [t] -> [a]
182   crearVector diccionario frase = [ contarPalabraFrase frase palabra | palabra <- diccionario]
```

En esta función residimos el diccionario y una lista de palabras de la frase, en esta pasamos el diccionario por una lista intencional donde cada palabra del diccionario pasa por la función de contarPalabrasFrase y crea una nueva lista.

```haskell
-- Contar cuantas veces aparece una palabra del diccionario en una frase --
contarPalabraFrase :: (Num p, Eq t) => [t] -> t -> p
contarPalabraFrase [] palabra = 0
contarPalabraFrase (x:xs) palabra = if(x == palabra) then
                                        1 + (contarPalabraFrase xs palabra)
                                    else
                                        contarPalabraFrase xs palabra
```

En la función contarPalabraFrase en el caso base recibimos una lista vacía y la palabra a analizar y si es así regresamos un 0, si no es así si residimos una lista y una palabra analizamos si el primer elemento de la lista es igual a la palabra a analizar si es así sumamos 1 y seguimos analizando el resto de la lista y si no solo seguimos analizando la lista.

Y regresamos esta lista vector.

```haskell
                vectoresFrases = [ crearVector diccionario frase | frase <- sinSWFrases]
```

Posteriormente regresamos a la función principal donde pasmos la lista de listas de palabras por una lista intencional donde cada lista pasa por la función crearVector que se explicó anteriormente.

Y así obtenemos un vector de la query y una lista de vectores de las frases.

```haskell
            -- Calcular similitud de cosenos --
            similitudes = [ simuitudCosenos vectorQuery vectorFrase  | vectorFrase <- vectoresFrases ]
```

Ahora en la función principal calculamos las similitudes de cuadrados, donde creamos una lista intencional con cada elemento de la lista de vectores donde cada vector pasa por la función similitudCosenos con el vector de la query.

```haskell
-- Similitud de cosenos --
simuitudCosenos :: (Eq p, Floating p, Integral a) => [a] -> [a] -> p
simuitudCosenos vector1 vector2 = let
                                    divisor = sumatoriaMultiplicacion vector1 vector2
                                    raizV1 = sqrt (fromIntegral(sumatoriaCuadrados vector1))
                                    raizV2 = sqrt (fromIntegral(sumatoriaCuadrados vector2))
                                    dividendo = raizV1 * raizV2
                                  in
                                    if(dividendo == 0) then
                                        0.0
                                    else
                                        fromIntegral divisor / dividendo
```

En la función similitudCosenos residimos dos vectores, primeramente, obtenemos la sumatoria de la multiplicación de los elementos con la función sumatoriaMultiplicacion y lo guardamos en divisor.

```
193                                   divisor = sumatoriaMultiplicacion vector1 vector2
```

```
203   -- Sumatoria de la multiclicacion de A*B
      sumatoriaMultiplicacion :: Num p => [p] -> [p] -> p
204   sumatoriaMultiplicacion [] [] = 0
205   sumatoriaMultiplicacion (x:xs) (y:ys) =  (x*y) + (sumatoriaMultiplicacion xs ys)
```

En la función sumatoriaMultiplicacion tenemos como caso base 2 listas vacías que nos regresa un 0, y si recibe 2 listas multiplica los 2 primeros elementos, y los sumas con la sumatoria del resto de la lista.

```
194                                   raizV1 = sqrt (fromIntegral(sumatoriaCuadrados vector1))
195                                   raizV2 = sqrt (fromIntegral(sumatoriaCuadrados vector2))
```

Posteriormente regresamos a la función similitudCosenos, donde se hace la sumatoria de los cuadrados de los elementos de un vector, para esto lo pasamos por la función sumatoriaCuadrados y el resultado de esta le sacamos la raíz cuadrada, esto se hace por cada vector.

```
207   -- Sumatoria de los cuadrados de un vector --
      sumatoriaCuadrados :: Num p => [p] -> p
208   sumatoriaCuadrados [] = 0
209   sumatoriaCuadrados (x:xs) = (x*x) + (sumatoriaCuadrados xs)
```

En la función de sumatoriaCuadrados, obtenemos como caso base una lista vacía regresamos 0, si no es así multiplicamos el primer elemento por si mismo y lo sumamos a la sumatoria del resto de la lista. Este numero lo regresa a la función similitudCosenos.

```
196                                   dividendo = raizV1 * raizV2
197                                 in
198                                   if(dividendo == 0) then
199                                     0.0
200                                   else
201                                     fromIntegral divisor / dividendo
```

En la función similitudCosenos multiplicamos las 2 raíces y regresamos dependiendo si el dividendo es 0 regresamos 0 y si no es así dividimos los valores del divisor y el dividendo y lo regresamos en la función principal con una lista de las similitudes de cosenos entre las frases y la query.

```
103                 -- Obtner el mayor de la similitudes de coseno --
104                 mayorSimilitud = maximum similitudes
105                 indiceMayorSimilitud = indiceElemento similitudes mayorSimilitud 0
```

Ahora en la función principal obtenemos el valor mayor de la lista de similitudes de coseno y lo guardamos en mayorSimilitud.

Y sacamos a que índice de la lista de frases pertenece el mayor valor de las similitudes de cosenos, ya que un valor de similitud pertenece a una frase que pertenece a una respuesta, esto lo hacemos por la función de indiceElemento en el cual enviamos la lista de similitudes, el mayor valor y el índice 0 para que inicie el ciclo y su respuesta la guardamos en índiceMayorSimilitud.

```
211   -- Obtener el indice de un elemento --
      indiceElemento :: (Num t1, Eq t2) => [t2] -> t2 -> t1 -> t1
212   indiceElemento [] elemento indice = (-1)
213 v indiceElemento (x:xs) elemento indice = if (x == elemento) then indice
214                                           else indiceElemento xs elemento (indice+1)
```

En la función, indiceElemento tenemos como caso base una lista vacía, el elemento a buscar y un índice, si no es así comparamos el primer elemento de la lista con el elemento de la lista que estamos buscando si es así regresa el índice, si no es así seguimos analizando el resto de la lista e incrementamos el índice.

```
106             in
107                 if(mayorSimilitud == 0) then
108                     "No tengo palabras"
109                 else
110                     if (indiceMayorSimilitud == -1) then
111                         "Ingresa una query"
112                     else
113                         respuestas !! indiceMayorSimilitud
```

Finalmente comparamos primeramente si el mayor en similitud es cero (si es así no encontró ninguna similitud) y si es así regresamos una cadena, si no ahora comparamos si el indiceMayor es -1 (No se ingresó ninguna query o query vacía) y regresamos una cadena, si no regresamos un elemento de la lista de respuestas en el índice indicado por el índice de la mayor similitud y muestra la respuesta del chatbot.

Código íntegro

El programa fue escrito en el editor VSCode y usando WinHugs.

Código Fuente para descargar: https://drive.google.com/file/d/15Z6Fwr-AiEgS3zI8cT4KoZvqTaIz-kPn/view

```
-- ChatBot ---

import Data.Char

-- Lista de Fraces --

frases = ["Hola Buenos dias Que onda",

        "Que piensas de Avengers o Vengadores",

        "Cual es tu pelicula favorita de STAR WARS",

        "Que te parece The Mandalorian",

        "Que prefies Netflix o Disney plus",

        "Que piensas de Harry potter",

        "Tu pelicula favorita de Marvel",

        "Te gustan las pelicualas mexicanas",

        "Cual fue la ultima pelicuala que te gusto",

        "Ciencia Ficcion o Fantasia",

        "Que pelicula me recomiendas o recomendarias ver",

        "Que serie me recomiendas ver",

        "Que serie de animacion me recomiendas ver",

        "Cual fue la ultima serie que viste",

        "Que pelicula de animacion me recomiendas",

        "Que opinas de las pelicualas del señor de los anillos",

        "Que opinas de las peliculas de terror",

        "En que pelicua te dormiste",

        "Que pelicula te sorprendio",

        "Que pelicua siempre has querido ver pero no puedes",
```

"Que genero te gusta mas",

"Que pelicula esperas a que salga",

"Pelicula en la que has llorado o te hiso llorar",

"Serie que esperas ver proximamente",

"Cual es la mejor pelicula"]

-- Lista de Respuestas --

respuestas = ["Hola, Como estas ? ..",

"Avengers es de mis peliculas favoritas de super heroes sobre todo la tercera parte",

"Mi pelicula favorita es STAR WARS La venganza de los Sith",

"The mandalorian es una buena serie, Baby Yoda es adorable",

"Yo prefiero ser pirata y descargar las pelicuas de cualquiera",

"Harry Potter son buenas peliculas que todos deberian ver",

"La mejor pelicula de marvel el Avengers Infinity War con su final estoy como :D",

"No me gustan las pelicualas de mexicanas de comedia pero algunas son muy buenas",

"La ultima pelicula que vi y que me gusto fue la de Nuevo Orden",

"Me gusta mas la ciencia ficccion pero no le ago feo a la fantacia",

"Te recomiendo ver la pelicula de los Hijos del Hombre es una buena pelicula",

"Te recomiendo ver la serie de Game of Thrones y sus libros",

"Te recomiendo la serie de Bojack Horseman en Netflix una buena sere de comediay te hace reflexionar",

"La ultima serie que vi fue Aggretsuko me mori de la risa Ja ja ja",

"Te recomiendover la pelicual de Your Name, buen final",

"Son buenas pelicualas que nunca e podido ver seguidas",

"Las pelicuas de terror no dan miedo solo son de suspenso",

"Me dormi en la pelicual de Gretel y Hansel no recuerdo la mitad",

"Godzilla King of Moster me sorprendio con las peleas de mounstros",

"Siempre e querido ver la primera pelicula de Godzilla pero no se encuentra en buena calidad",

"Me gusta el genero de ciencia ficcion",

"Espero que salga la pelicula de Kong vs Godzilla",

"EL ultimo tren a subusa, llore con el final :'(",

"La serie que espero es The mandalorian temporada 2",

"La mejor pelicula no existe cada quien tiene sus gustos, pero si me preguntas seria la saga de STAR WARS"]

-- Lista de StopWords --

stopWords = ["de", "la", "que", "el", "en", "y", "a", "los", "del", "se", "las", "por", "un", "para", "con", "no", "una", "su", "al", "lo", "como",

"mas", "pero", "sus", "le", "ya", "o", "este", "si", "porque", "esta", "entre", "cuando", "muy", "sin", "sobre", "tambien", "me", "hasta",

"hay", "donde", "quien", "desde", "todo", "nos", "durante", "todos", "uno", "les", "ni", "contra", "otros", "ese", "eso", "ante", "ellos",

"e", "esto", "mi", "antes", "algunos", "que", "unos", "yo", "otro", "otras", "otra", "el", "tanto", "esa", "estos", "mucho", "quienes",

"nada", "muchos", "cual", "poco", "ella", "estar", "estas", "algunas", "algo", "nosotros", "mi", "mis", "tu", "te", "ti", "tu", "tus",

"ellas", "nosotras", "vosotros", "vosotras", "os", "mio", "mia", "mios", "mias", "tuyo", "tuya", "tuyos", "tuyas", "suyo", "suya", "suyos",

"suyas", "nuestro", "nuestra", "nuestros", "nuestras", "vuestro", "vuestra", "vuestros", "vuestras", "esos", "esas", "estoy", "estas",

"esta", "estamos", "estais", "estan", "este", "estes", "estemos", "esteis", "esten", "estare", "estaras", "estara", "estaremos", "estareis",

"estaran", "estaria", "estarias", "estariamos", "estariais", "estarian", "estaba", "estabas", "estabamos", "estabais", "estaban", "estuve",

"estuviste", "estuvo", "estuvimos", "estuvisteis", "estuvieron", "estuviera", "estuvieras", "estuvieramos", "estuvierais", "estuvieran",

"estuviese", "estuvieses", "estuviesemos", "estuvieseis", "estuviesen", "estando", "estado", "estada", "estados", "estadas", "estad", "he",

"has", "ha", "hemos", "habeis", "han", "haya", "hayas", "hayamos", "hayais", "hayan", "habre", "habras", "habra", "habremos", "habreis",

"habran", "habria", "habrias", "habriamos", "habriais", "habrian", "habia", "habias", "habiamos", "habiais", "habian", "hube", "hubiste",

"hubo", "hubimos", "hubisteis", "hubieron", "hubiera", "hubieras", "hubieramos", "hubierais", "hubieran", "hubiese", "hubieses", "hubiesemos",

"hubieseis", "hubiesen", "habiendo", "habido", "habida", "habidos", "habidas", "soy", "eres", "es", "somos", "sois", "son", "sea", "seas",

"seamos", "seais", "sean", "sere", "seras", "sera", "seremos", "sereis", "seran", "seria", "serias", "seriamos", "seriais", "serian", "era",

"eras", "eramos", "erais", "eran", "fui", "fuiste", "fue", "fuimos", "fuisteis", "fueron", "fuera", "fueras", "fueramos", "fuerais", "fueran",

"fuese", "fueses", "fuesemos", "fueseis", "fuesen", "sintiendo", "sentido", "sentida", "sentidos", "sentidas", "siente", "sentid", "tengo",

"tienes", "tiene", "tenemos", "teneis", "tienen", "tenga", "tengas", "tengamos", "tengais", "tengan", "tendre", "tendras", "tendra",

"tendremos", "tendreis", "tendran", "tendria", "tendrias", "tendriamos", "tendriais", "tendrian", "tenia", "tenias", "teniamos", "teniais",

"tenian", "tuve", "tuviste", "tuvo", "tuvimos", "tuvisteis", "tuvieron", "tuviera", "tuvieras", "tuvieramos", "tuvierais", "tuvieran",

"tuviese", "tuvieses", "tuviesemos", "tuvieseis", "tuviesen", "teniendo", "tenido", "tenida", "tenidos", "tenidas", "tened"]

```
-- Metodo principal del Chatbot --
chatbot query = let
        -- Depuracion de las cadenas --
        fresesDepuradas = depuracion frases
        queryDepurada = depurarQuery query

        -- Partir las cadenas en palabras --
        palabrasFrases = [ words frase | frase <- fresesDepuradas ]
        palabrasQuery = words queryDepurada

        -- Quitar las StopWords --
        sinSWFrases = [quitarSWFrases frase | frase <- palabrasFrases]
        sinSWQuery = [palabra | palabra <- palabrasQuery, not (palabra `elem` stopWords)]

        -- Creacion del diccionario --
        diccionario = crearDiccionario sinSWQuery sinSWFrases
```

```
            -- Crear Vectores --
            vectorQuery = crearVector diccionario sinSWQuery
            vectoresFrases = [ crearVector diccionario frase | frase <- sinSWFrases]

            -- Calcular similitud de cosenos --
            similitudes = [ simuitudCosenos vectorQuery vectorFrase  | vectorFrase <-
vectoresFrases ]

            -- Obtner el mayor de la similitudes de coseno --
            mayorSimilitud = maximum similitudes
            indiceMayorSimilitud = indiceElemento similitudes mayorSimilitud 0
        in
            if(mayorSimilitud == 0) then
               "No tengo palabras"
            else
               if (indiceMayorSimilitud == -1) then
                  "Ingresa una query"
               else
                  respuestas !! indiceMayorSimilitud

-- Depuracion de la Query y Fraces --
depuracion frases = let
            -- Convertir a minusculas y quitar caracteres del español --
            frasesMinusculas = [ depurarCadena frase | frase <- frases ]
            -- Quitar signos de puntuacion --
            fresesPuntuacion = [ puntuacionCadena frase | frase <- frasesMinusculas]
            -- Quitar los no alfanumericos --
            fresesAlfa = [ alfaNumericosCadena frase | frase <- fresesPuntuacion]
```

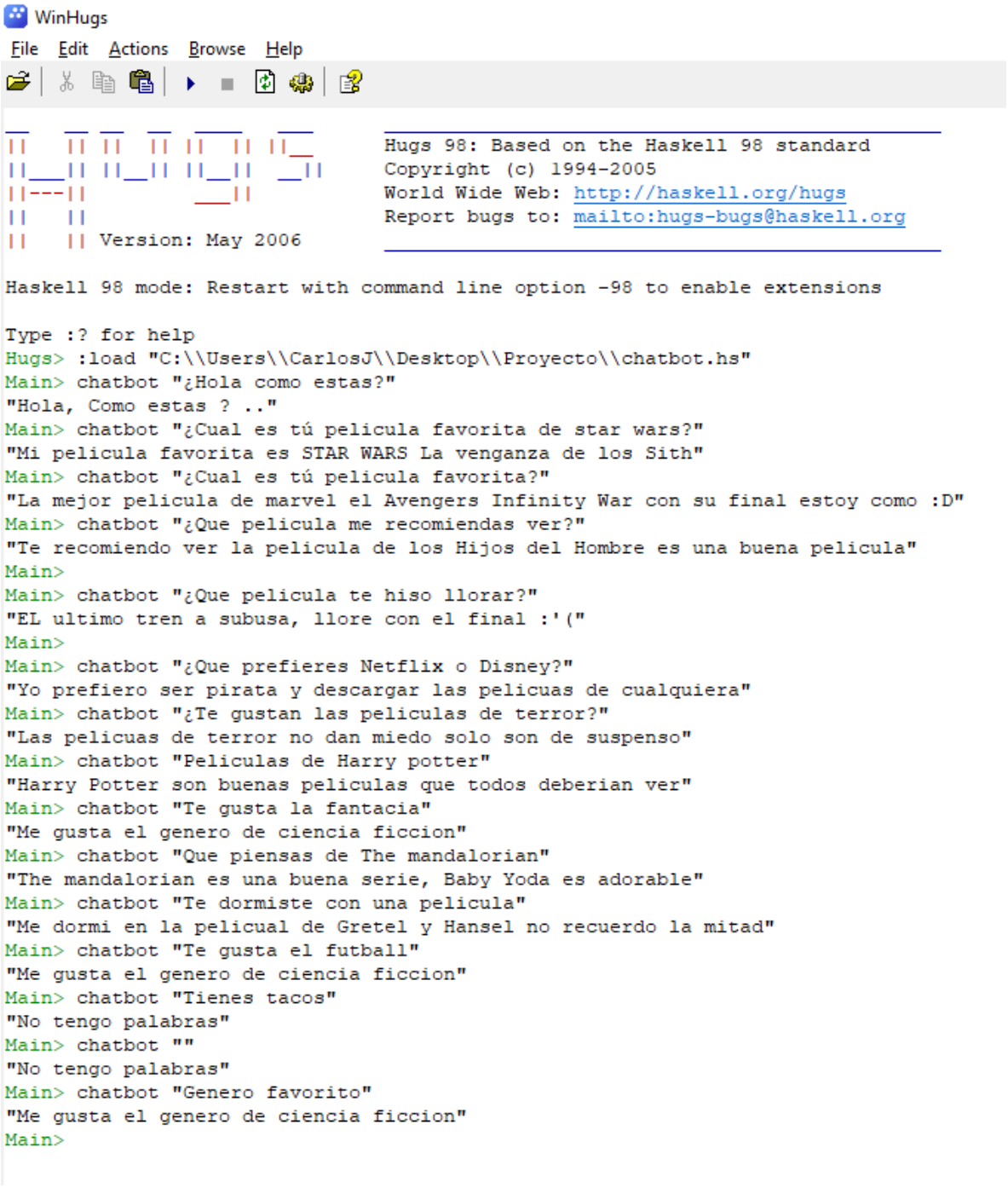
```
                in
                    fresesAlfa
depurarQuery query = let
                    queryMinusculas = depurarCadena query
                    queryPuntuacion = puntuacionCadena queryMinusculas
                    queryAlfa = alfaNumericosCadena queryPuntuacion
                in
                    queryAlfa

-- Hacer minusculas los caracteres de una frese y quitar los caracteres del español --
depurarCadena cad = [ quitarCaracteresEsp  (toLower car) | car <- cad]

-- Quitar caracteres del español --
quitarCaracteresEsp car
            | ord car == 225 = 'a' -- Á á --
            | ord car == 233 = 'e' -- É é --
            | ord car == 237 = 'i' -- Í í --
            | ord car == 243 = 'o' -- Ó ó --
            | ord car == 250 = 'u' -- Ú ú --
            | ord car == 252 = 'u' -- Ü ü --
            | ord car == 241 = 'n' -- Ñ ñ --
            | ord car == 147 = '?' -- “ --
            | ord car == 148 = '?' -- ” --
            | ord car == 172 = '?' -- ¬ --
            | ord car == 191 = '?' -- ¿ --
            | ord car == 161 = '?' -- ¡ --
            | ord car == 176 = '?' -- ° --
            | ord car == 180 = '?' -- ´ --
            | otherwise = car
```

```
-- Quitar signos de puntuacion de una cadena --
puntuacionCadena cad = [ car | car <- cad, not (isPunctuation car)]

-- Quitar los caracteres no alfanumericos --
alfaNumericosCadena cad = [ car | car <- cad, isAlpha car || car == ' ']

-- Quitar las stop words de las frases --
quitarSWFrases frase = [ palabra | palabra <- frase, not (palabra `elem` stopWords)]

-- Creacion del diccionario --
crearDiccionario listaQuery listaFrases =   let
                                              frasesUnidas = unirFrases listaFrases
                                              diccionarioRepetido = listaQuery ++ frasesUnidas
                                              diccionario = quitarRepetidos diccionarioRepetido
                                            in
                                              diccionario

-- Unir frases en una sola lista --
unirFrases [] = []
unirFrases (x:xs) = x ++ (unirFrases xs)

-- Quitar los elementos repetidos del diccionario --
quitarRepetidos [] = []
quitarRepetidos (x:xs) = if (x `elem` xs) then
                           quitarRepetidos xs
                         else
                           x:quitarRepetidos xs
```

```
-- Crear un vector --
crearVector diccionario frase = [ contarPalabraFrase frase palabra | palabra <- diccionario]

-- Contar cuantas veces aparece una palabra del diccionario en una frase --
contarPalabraFrase [] palabra = 0
contarPalabraFrase (x:xs) palabra = if(x == palabra) then
                                      1 + (contarPalabraFrase xs palabra)
                                    else
                                      contarPalabraFrase xs palabra

-- Similitud de cosenos --
simuitudCosenos vector1 vector2 = let
                                    divisor = sumatoriaMultiplicacion vector1 vector2
                                    raizV1 = sqrt (fromIntegral(sumatoriaCuadrados vector1))
                                    raizV2 = sqrt (fromIntegral(sumatoriaCuadrados vector2))
                                    dividendo = raizV1 * raizV2
                                  in
                                    if(dividendo == 0) then
                                      0.0
                                    else
                                      fromIntegral divisor / dividendo

-- Sumatoria de la multiclicacion de A*B
sumatoriaMultiplicacion [] [] = 0
sumatoriaMultiplicacion (x:xs) (y:ys) =  (x*y) + (sumatoriaMultiplicacion xs ys)

-- Sumatoria de los cuadrados de un vector --
sumatoriaCuadrados [] = 0
sumatoriaCuadrados (x:xs) = (x*x) + (sumatoriaCuadrados xs)
```

```
-- Obtener el indice de un elemento --
indiceElemento [] elemento indice = (-1)
indiceElemento (x:xs) elemento indice = if (x == elemento) then indice
                                        else indiceElemento xs elemento (indice+1)
```

Pruebas



Código Fuente para descargar: https://drive.google.com/file/d/15Z6Fwr-AiEgS3zI8cT4KoZvqTaIz-kPn/view

Conclusiones

En conclusión, podemos decir que el programar un chatbot ah sido un reto interesante, principalmente armarlo con un lenguaje funcional como haskell donde se necesita tanto de la recursión, y es nuevo para nosotros, podemos ver como en este ejercicio se usaron mayormente operaciones sobre listas y usando funciones de orden superior. Es un buen ejercicio para fortalecer nuestro conocimiento sobre este lenguaje y el uso de listas, además que funciones como las listas intencionales las podemos usar en otros lenguajes y el extraer la lógica de recursión nos ayuda tanto en este lenguaje como en otros lenguajes para futuros proyectos.

Referencias

Socialmood. (s. f.). ¿Qué es un chatbot? 40deFiebre. Recuperado 27 de noviembre de 2020, de https://www.40defiebre.com/que-es/chatbot

Similitud de coseno - Cosine similarity - qaz.wiki. (s. f.). QAZ. Recuperado 27 de noviembre de 2020, de https://es.qaz.wiki/wiki/Cosine_similarity