Primer Trabajo de Programación
Facultad de Matemática y Computación de la Universidad de La Habana
Salma Fonseca Curbelo C122

Al inicializar el programa, se ejecuta el método GetDocuments, que se encuentra en la clase Documents. Este método primeramente obtiene la dirección de todos los archivos de textos en la carpeta Content, para lo que se crea el string filepath que se le asigna la dirección de la carpeta a la que entrará y luego con la función GetFiles, de la clase Directory se obtienen las direcciones de todos los documentos de ella y se guardan en un array de string llamado docs.

```
string filePath = @"../Content/";
string[] docs = Directory.GetFiles(filePath, "*.txt");
```

Luego se lleva a cabo un ciclo foreach, que para cada dirección que se encuentra en docs, con la función ReadAllText de la clase File, se obtiene el texto de los documentos, ignorando las mayúsculas con el método ToLower y se guardará en una variable de tipo string llamada document. Después se crea una lista de string con cada palabra del texto, para separarlas se utiliza el método Split de la clase Regex, que cada vez que haya un caracter que sea distinto a una letra (puede tener o no tilde) o un número, hace una división, devolviéndo al final un arreglo de string con cada palabra del texto, el cual se convierte a lista con el método ToList.

Una vez creada la lista se creará un nuevo vector, que se le pasará como parámetro la lista con cada palabra del texto y la dirección.

```
foreach (string doc in docs){
    string document = File.ReadAllText(doc).ToLower();

    List <string> wordsOfDocs = Regex.Split(document, @"[^\wáéíóúÁÉÍÓÚ]+").ToList();
    Vector v = new Vector(wordsOfDocs, doc);
}
```

La construcción del vector se lleva a cabo en la clase Vector, que tiene como propiedad un string path que será la dirección del texto, un entero cantidadPalabras que representará la cantidad de palabras que tiene el vector, una lista de string vectorwords con todas las palabras y un diccionario TF_IDF de cada palabra con su valor de TF. Para construir cada vector:
- a Vectorwords se le asigna la lista de string que recibe como parámetro.
- a path se le asigna el string que recibe como parámetro.
- a cantidadPalabras se le asigna el tamaño de la lista que se recibe, lo cual se determina con la función Count.

Cada vector que se cree se le añadirá a la lista de vectores files de la clase Matrix.

```
public Vector(List<string> words, string path){
        this.Vectorwords = words;
        this.path = path;
        this.cantidadPalabras = words.Count();
        Matrix.files.Add(this);
    }
```

Una vez terminado todo esto, a cada documento le corresponderá un vector.

Luego el método GetDocuments entrará en otro ciclo que, para cada vector de la lista files en la clase Matrix, se llama al método CalculateTFIDF y enviará como parámetro el vector. Este método se encuentra en la clase Matrix, la cual tiene como propiedad la lista de vectores files y un diccionario idf con todas las palabras y su valor de idf.
CalculateTFIDF comienza un ciclo foreach que por cada palabra del vector, si el diccionario TF_IDF no la contiene, entonces se añadirá y se le asigna el valor 1, en caso de sí existir entonces se le aumenta uno al valor de palabra.

```
        foreach (string word in v.Vectorwords){
            if (!v.TF_IDF.ContainsKey(word)){
                v.TF_IDF.Add(word, 1);
                }
                else{
```

```
                    v.TF_IDF[word] ++;
                }
            }
```

Al culminarlo entra en otro foreach que recorrerá cada palabra que se encuentra en las claves del diccionario, si esta no se encuentra en las claves del diccionario idf, entonces añadirá y se le asigna el valor 1, en caso de sí existir entonces se le aumenta uno al valor de palabra.
Y posteriormente se realiza el cálculo del TF de esa palabra, que sería la cantidad de veces que se repite la palabra en el documento (el valor que tiene asignado en el diccionario TF_IDF), divido entre la cantidad de palabras del documento que es una propiedad del vector.

```
foreach (string key in v.TF_IDF.Keys){
                if (! idf.ContainsKey(key)){
                    idf.Add(key, 1);
                }
                else{
                    idf [key] ++;
                }
                v.TF_IDF[key] /= v.cantidadPalabras;
            }
```

Al haber hecho esto con todos los vectores se tendrá calculado el TF de cada palabra por documento y también se tendrá en cuantos documentos existe cada palabra.
El método GetDocuments entre en otro ciclo foreach que para cada palabra clave del diccionario idf, realizará el cálculo del IDF, que sería el logaritmo de la cantidad de documentos de la base de datos, que se obtiene con la cantidad de vectores que contiene la lista file, utilizando la funcion Count, dividido entre la cantidad de documentos que contienen la palabra (es el valor que tiene en el diccionario idf)

```
foreach (string key in Matrix.idf.Keys){
    Matrix.idf [key] = Math.Log(Matrix.files.Count/Matrix.idf [key]);
```

Al finalizar todo esto se abrirá el programa con todos los documentos cargados, y los TF e IDF calculados.

Al usuario introducir lo que desea buscar, se ejecuta la función Search dentro de la clase Moogle, que recibirá como parámetro el query que se introdujo. Comienza convirtiendo dicho query a una lista de string query, con el método Split igual que se utilizó anteriormente y llevándolo a una lista con la función ToList. También se crea una lista de SearchItem.
Comienza un ciclo que por cada vector de la lista de vectores files, crea una variable tipo double llamada score que se le asigna el valor 0 para empezar. Entre en otro ciclo que para cada palabra de la lista query, si esta se encuentra en las claves del diccionario TF_IDF y su valor en el diccionario idf es mayor que 0.5 (si fuera menor significa que la palabra se encuentra en la mayoría de los documentos por tanto no tendría importancia), se multiplica el valor del TF por el del IDF lo cual se le asigna a la variable double tfidf.
Luego al valor del score se le suma tfidf multiplicado por el valor del idf de esta palabra, multiplicado por el TF de la palabra en el query. Para saber la cantidad de veces que se encuentra la palabra en el query se utiliza la funcion Count.

```
foreach(Vector doc in Matrix.files){
            double maxTFIDF = 0;
            string maxword = "";
            double score = 0;
            foreach(string word in query){
                if (doc.TF_IDF.ContainsKey (word) && Matrix.idf [word] > 0.5){
                    double tfidf = doc.tf[word] * Matrix.idf[word];
                    score += tfidf * (Matrix.idf[word] * (double) query.Count(x => x == word) /
query.Count);
}
```

Al terminar esto con todas las palabras del query, si el score es mayor que 0, se le añade a la lista de SearchItem, llamada result, el nombre del documento, que se obtiene a través de la función GetName, el Snnipet, que se obtiene mediante la función Snippet y el score.
Al terminar esto con todos los vectores, se ordenan de manera descendente teniendo en cuenta el score y se convierta la lista a un array. Lo cual retorna la función y así se puede dar el resultado de la

busqueda.

return result.OrderByDescending(x => x.Score).ToArray();

La función Snippet, de la misma clase Moogle recibe un string con la dirección del documento, utilizando la función ReadAllText como se hizo anteriormente se obtiene el texto del documento y se crea un substring:
Si el tamaño del string recibido es menor de 150 caracteres entonces se devuelve todo el string con la función Substring en 0; en caso de ser mayor entonces se hace el substring de los primeros 150 caracteres, igualmente con la función substring desde 0 hasta 149.

```
static string Snippet (string path, string word){
        string text = File.ReadAllText(path);
        if (text.Length < 150){
           text =  text.Substring(0);
        }
        else{
         text = text.Substring(0, 149);
        }
        return text;
    }
```

La función GetName con la dirección del documento, crea un substring que va desde el último '/ '(con la función IndexOfLast + 1) hasta donde empieza el txt, que se resta el IndexOf de donde está txt menos el IdexOfLast de '/' menos 1. IndexOf devuelve la posición del caracter u string que se pasa como parámetro.

```
 public string Getname (){
    return path.Substring(path.LastIndexOf('/') + 1, path.IndexOf(".txt") - path.LastIndexOf('/') -
1);
}
```