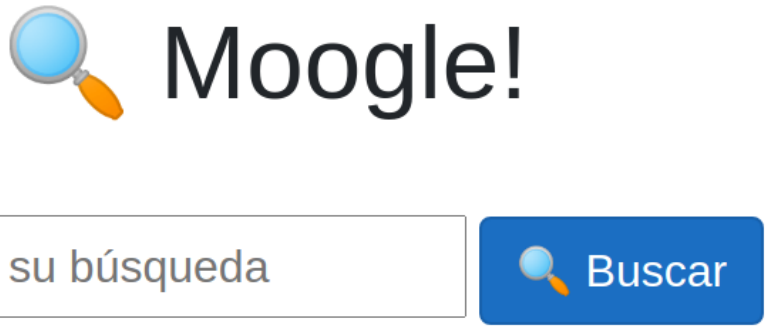
Moogle



> 1er Proyecto de Programación Curso: 2023-2024 Autor: Diego A. Martínez Jiménez Grupo: C121

Simple motor de búsqueda con simple interfaz gráfica.

Instrucciones para correr el proyecto

- **Abrir una terminal en la carpeta del proyecto.**

Linux:

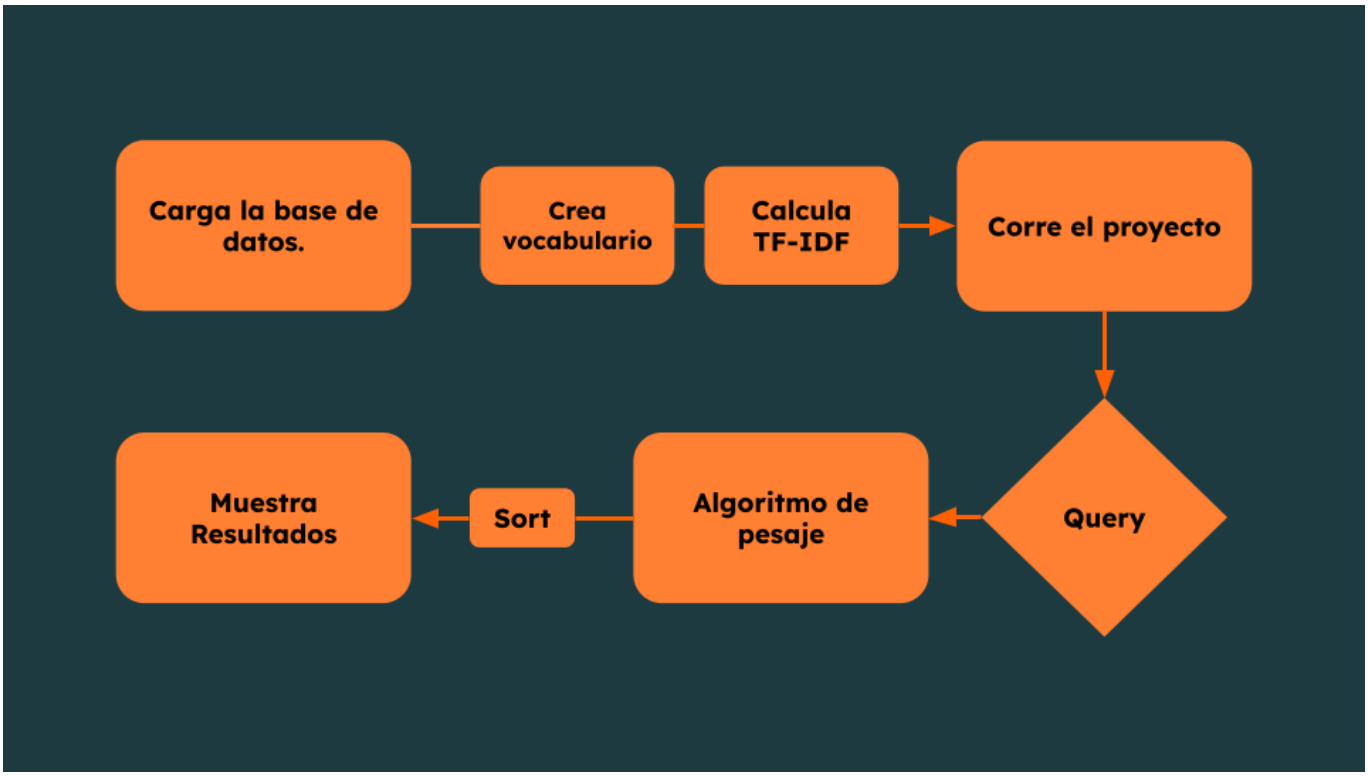
- `make dev`

Windows:

- `dotnet run --project`

Arquitectura del proyecto

Aceptando la misión que se me fue otorgada, ayude en la implementación de **Moogle**!. Para ello tuve en cuenta la información que me pudieron proporcionar acerca de "**TF-IDF**" y "**Álgebra lineal**". También me fue útil este link https://en.wikipedia.org/wiki/Tf–idf



> Orden de los procesos del proyecto.

Cargando los documentos

Lo primero que implemente fue una clase que nombre `Documents` esta contiene varios métodos relacionados con operaciones que se le pueden a hacer a documentos, por ejemplo el método `Documents.ReadText()` el cual retorna como string todo el texto de uno .txt. Lo más importante de esta clase es su constructor:

```
    public Documents(string path){


            this.path = path;
            int documents = 0;

            this.directory = GetDocuments(this.path);
            this.Vocabulary = GetVocabulary();


            foreach( string file in this.directory)documents++;
            this.documents = documents;

            this.TF = new Matrix(this.documents,this.words);
            this.IDF = new Vector(new double[words]);
            _IDF = new Vector(new double[words]);


            this.ComputeDocuments();


            _TFIDF = this.TF;
```

```
            _Vocabulary = this.Vocabulary;
            Doc = this.directory;
    }
```

Este recibe como parámetro `path` que deberá ser un string con la dirección de una carpeta donde estén almacenados documentos .txt, *(de no ser así no garantizo su correcto funcionamiento)*. Al crear una instancia de `Documents` esta asigna un número a cada término encontrado en el corpus, (el método encargado de este proceso es `Documents.GetVocabulary`) luego el método `ComputeDocuments` calcula el TF-IDF de cada documento, creando una matriz donde `TFIDF[i,j]` tiene guardado el TF-IDF de el término `j` en el documento `i`. Toda la información útil es almacenada en variables tipo `static` para su uso posterior.

En las clases `Algebra.Vector` y `Algebra.Matrix` están implementados en métodos las operaciones relacionadas con estos conceptos provenientes del **Álgebra Lineal**. Estas son fundamentales para el funcionamiento de `MoogleEngine.Documents`.

Respondiendo la query

Luego de implementar estas clases, arregle la clase `Moogle` la cual en su momento no era muy útil. El objetivo principal de esta clase es responder a la query a través del método `Moogle.Query`. La idea para este método es modelar un vector en el que cada componente de este, sea el TF-IDF de cada término que pertenezca al corpus de documentos. Luego hallar el coseno entre este vector y cada uno de los vectores creados a partir de los documentos.

Primero guardo en variables el TF-IDF, el IDF y el vocabulario previamente calculados al cargar los documentos.

```
    Matrix TFIDF = Documents._TFIDF;
    Vector idf = Documents._IDF;
    Dictionary<string,int> vocabulary = Documents._Vocabulary;
```

Luego calcula el TF-IDF de cada término en la query, en caso de un término de la query no encontrarse en `vocabulary` será ignorado:

```
    tfidf = Documents.CalculateTF(query,vocabulary);

    for(int i = 0; i < idf.Count; i++){
        tfidf[i] *= idf[i];
    }
```

Se almacena luego en `tfidf` que es una variable tipo `Algebra.Vector` para luego calcular el Producto Punto entre `tfidf` y cada uno de los vectores construidos a partir de la matriz `TFIDF` en esta línea:

```
    Vector currentDocTFIDF = new Vector(TFIDF,i);
```

El Producto Punto se calcula con el método `Algebra.Vector.DotProduct` que hace, pues exactamente lo que su nombre indica. Luego el resultado del cálculo será el `score` de su respectivo documento. Luego los documentos son ordenados con el método `Array.Sort` dependiendo de su respectivo `score`. En caso de que el `score` de un documento sea `0` es ignorado, pues no tiene relevancia alguna con la query.

Luego se construye un `SearchResult` a partir de esta información guardada en `items`.

```
    return new SearchResult(items, suggestion);
```

Después puede ver el resultado en su navegador.

Las Sugerencias

Para las sugerencias usé el algoritmo de Distancia de Levenshtein. El cual calcula de forma dinámica el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. El método para esto es `Documents.EditDistance`.

Al recibir una query la sugerencia se calcula dentro del método `Utils.Suggestion` que por cada término guardado en `vocabulary` calcula su respectiva Distancia de Levenshtein con respecto a la query.

En caso de que no se encuentre ningún término relacionado con query, `Moogle.Query` retornara los documentos relacionados con la sugerencia.