Wikipedia – Search - Engine

Gestione dell'Informazione

2019/2020

Bordini Luca

Parigi Luca

FASE 1

Scelte progettuali.

Spiegazione dei procedimenti attuati.

Primi test eseguiti per verificare un embrionale funzionamento.

Fase 1 - Parser

La prima scelta nella realizzazione del progetto è stata utilizzare SAX come Parser, poiché non ha necessità di mantenere tutto il documento in memoria.

Dovendo analizzare file XML molto grandi si rivela più adatto rispetto a DOM.

Fase 1 – Indexer e Searcher

La seconda scelta è stata utilizzare Whoosh per creare gli indici su cui fare le ricerche.

Scelta fatta per motivi di comodità, volendo programmare con Python e poiché abbiamo trovato molto chiara la documentazione.

Fornisce già varie funzioni di scoring, di analisi del testo ed è possibile inserire query frasali di default.

Fase 1 – Prima ricerca

Il primo passo è stato implementare la classe "WikiXmlHandler" che permette di recuperare unicamente la parte testuale dalle pagine di Wikipedia.

L'abbiamo testato su un piccolo dump (circa 170 MB) scaricato da https://dumps.wikimedia.org/backup-index.html

Fase 1 – Prima ricerca

Il secondo passo è stato creare un semplice Schema su Whoosh, inserendo in modo grezzo ID, titolo e testo per eseguire le prime semplici operazioni di ricerca e verificare un primo funzionamento.

Fase 1 - Preprocessing del testo

Il terzo passo è stato utilizzare la libreria NLTK per fare il preprocessing di titolo e testo (in "Parser.py").

Abbiamo creato tokens per ogni linea di testo, eliminando le stopwords, utilizzando Lemmatizer e Stemmer per ottenere i lessemi.

Fase 1 – Feature Aggiuntive

1. Eliminazione della punteggiatura per ripulire il testo recuperato, al fine di poter controllare meglio i risultati del Parser.

2. Sfruttato la struttura delle pagine Wikipedia per suddividere il testo in:

- External Links : "== external links =="
- Infobox : "{{infobox"
- Category : "[[category"
- Body

Frammento di Parser.py

```python
class WikiXmlHandler(xml.sax.handler.ContentHandler):
    """Content handler for Wiki XML data using SAX"""
    def __init__(self):
        xml.sax.handler.ContentHandler.__init__(self)
        self._buffer = None
        self._values = {}
        self._current_tag = None
        self._pages = []
        self._count = 0
        self._page_count = 0

    def characters(self, content):
        """Characters between opening and closing tags"""
        if self._current_tag:
            self._buffer.append(content)

    def startElement(self, name, attrs):
        """Opening tag of element"""
        if name in ('title', 'text'):
            self._current_tag = name
            self._buffer = []

    def endElement(self, name):
        """Closing tag of element"""
        if name == self._current_tag:
            self._values[name] = ''.join(self._buffer)

        if name == 'page':
            self._page_count += 1
            obj = process_article(**self._values)
            self._tempPage = self._page_count, obj[0], obj[1], obj[2], obj[3], obj[4], obj[5]
            self._pages.append(self._tempPage)
```

Esempio di Output

```
page nr:  0
id:  1
title:  dna
body:  non-techn introduct topic introduct genet us pp-vandal small=y pp-move-indef short descript molecul carri genet inform use dm
category: dna helix biotechnolog nucleic acidsdna helix biotechnolog nucleic acidsdna helix biotechnolog nucleic acidsdna helix biot
infobox:
links:  librari resourc box |onlinebooks=y |by=no |lcheading= dna |label=dna wikiquot wikivers common category|dna spoken wikipedia|
URL:  https://en.wikipedia.org/wiki/DNA
```

Fase 1 – Ranking

Riportata la suddivisione del testo nello Schema ("Indexer.py") e inserita la ricerca su multipli campi con funzione di scoring ("Searcher.py).

```
# Create the Schema
schema = Schema(id=NUMERIC, title=TEXT, body=TEXT, category=TEXT, infobox=TEXT, links=TEXT, URL=ID(stored=True))
```

Abbiamo testato i primi ranking (con altri termini rispetto a quelli richiesti dal progetto) sul dump scaricato inizialmente.

Frammento di Searcher.py

```
with index.searcher(weighting=w) as searcher:
    results_list.insert(END, "Results for query '" + text_input + "': \n")
    qp = qparser.MultifieldParser(["id", "title", "body", "category", "infobox", "links"], index.schema)


    query = qp.parse(text_input)
    results = searcher.search(query, terms=True, limit=20)
```

Esempio di Output -->

1 https://en.wikipedia.org/wiki/DNA 542.0
2 https://en.wikipedia.org/wiki/DNA_sequencing 290.0
3 https://en.wikipedia.org/wiki/DNA_replication 277.0
4 https://en.wikipedia.org/wiki/Genealogical_DNA_test 227.0
5 https://en.wikipedia.org/wiki/Molecular_models_of_DNA 185.0
6 https://en.wikipedia.org/wiki/Nucleic_acid_double_helix 178.0
7 https://en.wikipedia.org/wiki/Francis_Crick 178.0
8 https://en.wikipedia.org/wiki/Z-DNA 174.0
9 https://en.wikipedia.org/wiki/Cancer_epigenetics 168.0

FASE 2

Test sulle query.

Grafici a confronto e valutazioni.

Fase 2 – Test sul dump

Dopo aver verificato il funzionamento del codice abbiamo creato un dump contenente i 900 articoli utili per il set di query fornitoci, più un centinaio di articoli di rumore.

Inizialmente abbiamo fissato il limite di recupero a 30 articoli a volta, verificando anche sul nuovo dump il funzionamento del codice.

Abbiamo poi fissato il limite di recupero ai primi 10 articoli, testando 3 funzioni di scoring: BM25F, Frequency e TF-IDF.

Frammenti di risultati raccolti

```
w = scoring.Frequency()

1 https://en.wikipedia.org/wiki/Apple_Inc.  1366.0
2 https://en.wikipedia.org/wiki/Apple_TV  678.0
3 https://en.wikipedia.org/wiki/IPhone  579.0
4 https://en.wikipedia.org/wiki/Apple_Pay  539.0
5 https://en.wikipedia.org/wiki/History_of_Apple_Inc.  475.0
6 https://en.wikipedia.org/wiki/Apple-dG_esigned_processors  474.0
7 https://en.wikipedia.org/wiki/Criticism_of_Apple_Inc.  438.0
8 https://en.wikipedia.org/wiki/Steve_Jobs  436.0
9 https://en.wikipedia.org/wiki/Timeline_of_Apple_Inc._products  347.0
10 https://en.wikipedia.org/wiki/MacOS  346.0
```

R	Rec	Prec
1	0.10	1.00
...		
5	0.20	0.40
...		
7	0.30	0.43
...		
9	0.40	0.44

Ranking #2 : 0.227

DCG #2 : 6 + 4/ln(5) + 5/ln(7) + 1/ln(9) = 11.50
NDCG #2 : 0.507

```
w = scoring.Frequency()

1 https://en.wikipedia.org/wiki/DNA  542.0
2 https://en.wikipedia.org/wiki/DNA_sequencing  290.0
3 https://en.wikipedia.org/wiki/DNA_replication  277.0
4 https://en.wikipedia.org/wiki/Genealogical_DNA_test  227.0
5 https://en.wikipedia.org/wiki/Molecular_models_of_DNA  185.0
6 https://en.wikipedia.org/wiki/Nucleic_acid_double_helix  178.0
7 https://en.wikipedia.org/wiki/Francis_Crick  178.0
8 https://en.wikipedia.org/wiki/Z-DNA  174.0
9 https://en.wikipedia.org/wiki/Cancer_epigenetics  168.0
10 https://en.wikipedia.org/wiki/DNA_computing  164.0
```

R	Rec	Prec
1	0.10	1.00
....		

Ranking #2: 0.10

DCG#2: 6.00
NDCG#2: 0.264

```
w = scoring.Frequency()

1 https://en.wikipedia.org/wiki/Eye_of_Ra  152.0
2 https://en.wikipedia.org/wiki/Horus  147.0
3 https://en.wikipedia.org/wiki/Talk:Eye_of_Horus  136.0
4 https://en.wikipedia.org/wiki/Osiris_myth  132.0
5 https://en.wikipedia.org/wiki/Eye_of_Providence  98.0
6 https://en.wikipedia.org/wiki/Eye_of_Horus  96.0
7 https://en.wikipedia.org/wiki/Hathor  67.0
8 https://en.wikipedia.org/wiki/Set_(deity)  52.0
9 https://en.wikipedia.org/wiki/Isis  52.0
10 https://en.wikipedia.org/wiki/Four_sons_of_Horus  50.0
```

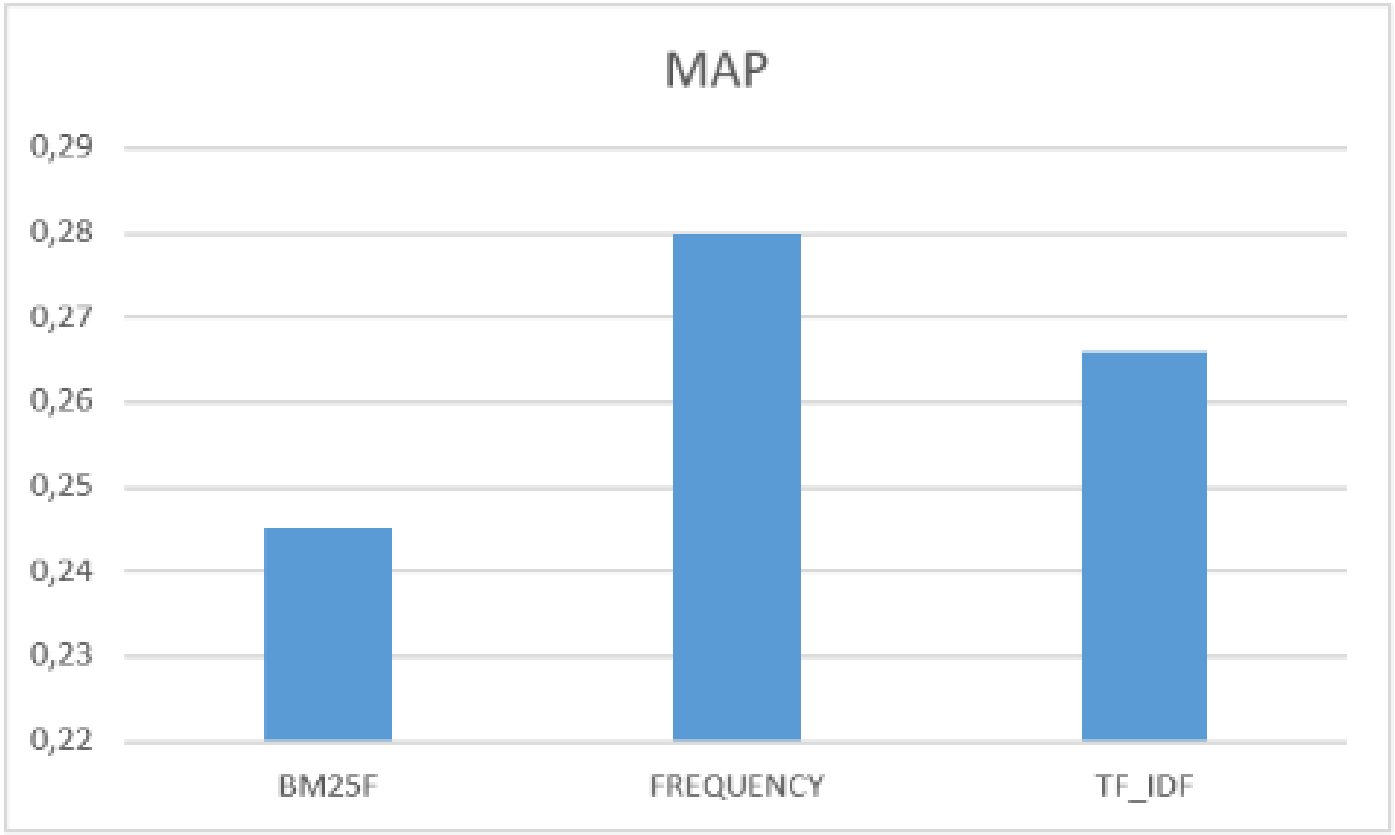
R	Rec	Prec
1	0.10	1.00
2	0.20	1.00
3	0.30	1.00
...		
6	0.40	0.67
7	0.50	0.71
8	0.60	0.75
9	0.70	0.00
10	0.80	0.00
...		

Ranking #2: 0.513

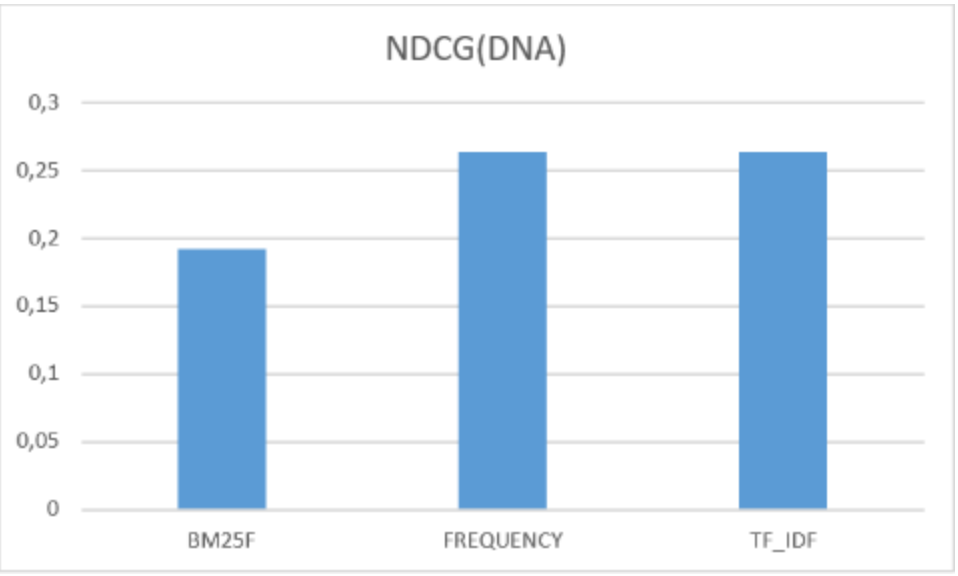
DCG#2: 3 + 5/ln(2) + 4/ln(3) + 6/ln(6) + 1/ln(7) + 1/ln(8) = 18.19
NDCG#2: 0.801

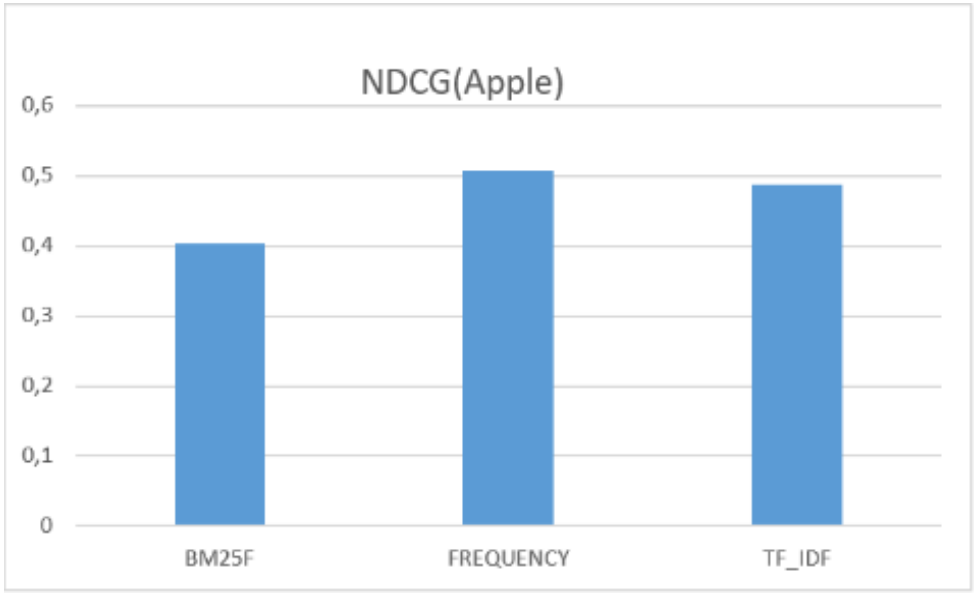
DCG di base = 6 + 5/ln(2) + 4/ln(3) + 3/ln(4) + 2/ln(5) + 1/ln(6) + 1/ln(7) + 1/ln(8) + 1/ln(9) + 1/ln(10) = 22.70

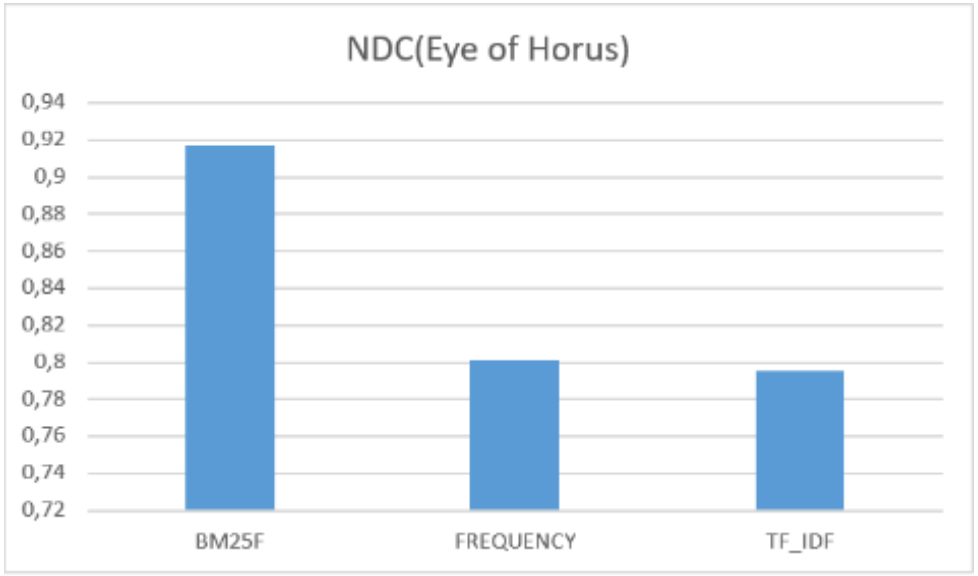
Grafici



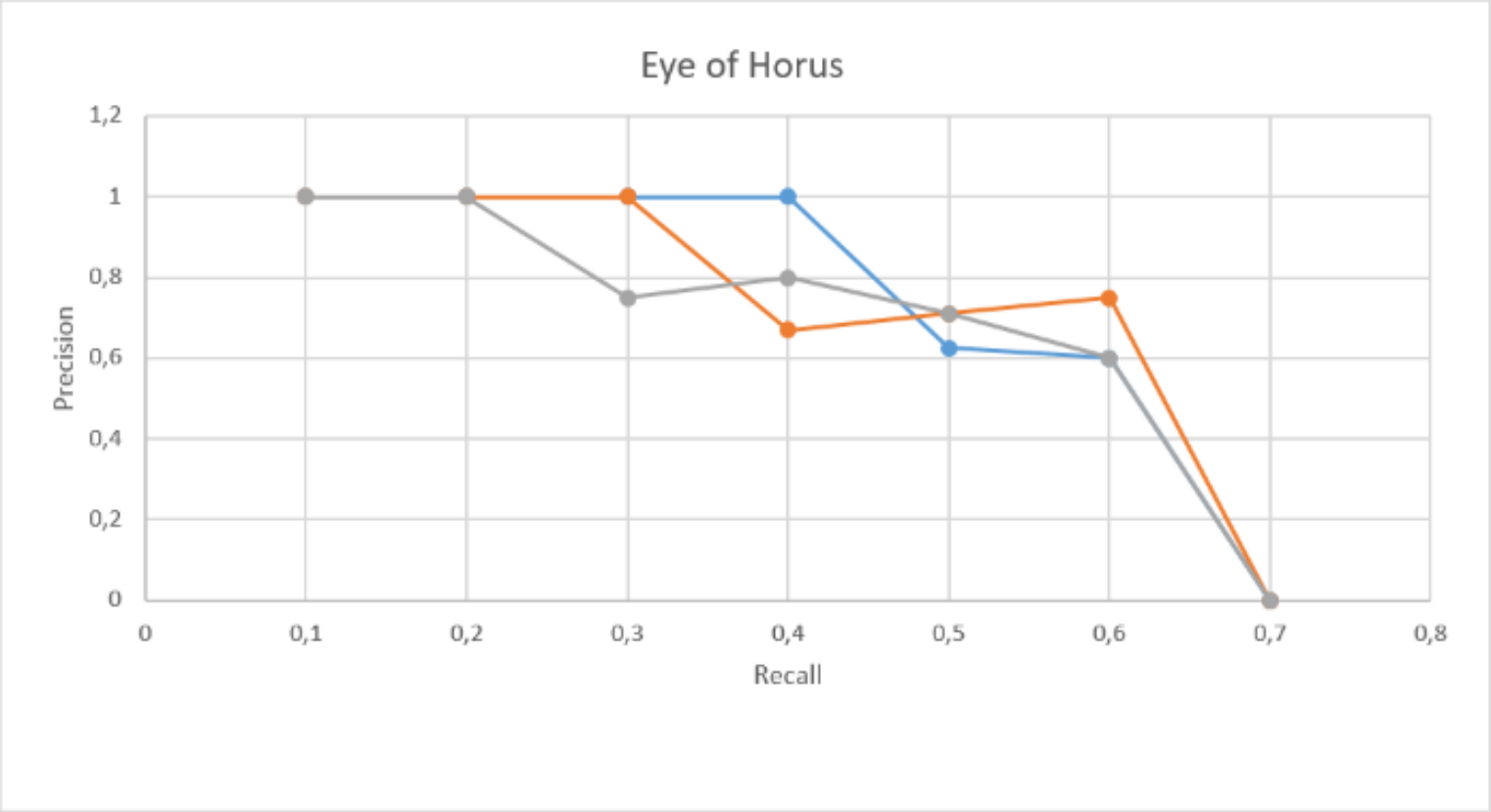
Grafici


NDCG(DNA)
0,3
0,25
0,2
0,15
0,1
0,05
0
BM25F
FREQUENCY
TF_IDF


NDCG(Apple)
0,6
0,5
0,4
0,3
0,2
0,1
0
BM25F
FREQUENCY
TF_IDF


NDC(Eye of Horus)
0,94
0,92
0,9
0,88
0,86
0,84
0,82
0,8
0,78
0,76
0,74
0,72
BM25F
FREQUENCY
TF_IDF

Grafici



BM25F – Azzurro
Frequency – Arancione
TF-IDF - Grigio

Fase 3

Test ulteriori.

Considerazioni.

Fase 3 – Altri test

Abbiamo fatto anche altri test con la funzione BM25F inserendo valori diversi in base alla posizione del termine all'interno della struttura del documento.

Ha prodotto un risultato interessante ponendo più elevato il valore del campo Category rispetto agli altri, ma soltanto per alcune query.

Abbiamo provato la funzione MultiWeighting per utilizzare più funzioni di scoring associandole a campi diversi, ma non ha fornito risultati interessanti.

Fase 3 – Scelta della funzione di Scoring

In seguito ai test eseguiti sulle query, abbiamo scelto Frequency come funzione di Scoring.

Fase 3 – Preprocessing della query

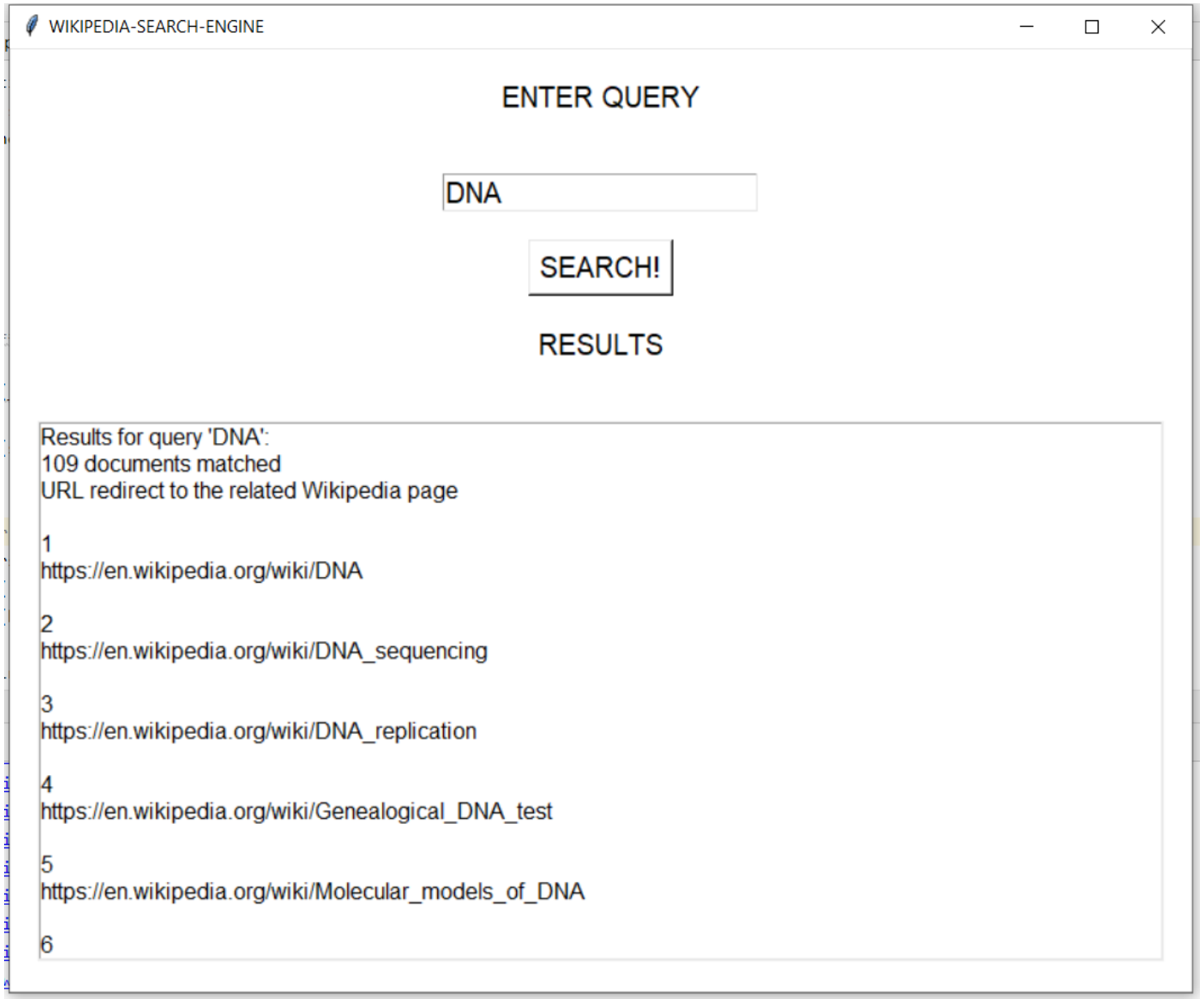
Abbiamo inserito le operazioni di preprocessing del testo anche sulla query.

Riscontro positivo: abbiamo recuperato articoli anche per la query "Do geese see god" che prima non forniva risultati.

Fase 3 – Interfaccia Grafica

Abbiamo utilizzato la libreria tkinter per creare un'interfaccia grafica minimale.

Abbiamo inserito il valore dell'URL nell'indice, per permettere all'utente di essere reindirizzato alle pagine di Wikipedia recuperate.


WIKIPEDIA-SEARCH-ENGINE
ENTER QUERY
DNA
SEARCH!
RESULTS
Results for query 'DNA':
109 documents matched
URL redirect to the related Wikipedia page
1
https://en.wikipedia.org/wiki/DNA
2
https://en.wikipedia.org/wiki/DNA_sequencing
3
https://en.wikipedia.org/wiki/DNA_replication
4
https://en.wikipedia.org/wiki/Genealogical_DNA_test
5
https://en.wikipedia.org/wiki/Molecular_models_of_DNA
6