

## Relazione — Homework: Sistema di indicizzazione e ricerca testuale

Studente: Douglas Ruffini

Mat.: 482379

Corso: Ingegneria dei Dati — Homework 2 (2025)

Repository: <https://github.com/DouglasRuffini/Ingegneria-dei-Dati/tree/master>

### Testo Homework:

1. scrivere un programma che indicizza i file .txt contenuti in una directory del proprio laptop. In particolare, si devono considerare due campi (e quindi creare due indici): il nome del file, il contenuto del file. Per ciascun campo utilizzare un analyzer appropriato;
2. scrivere un programma che legge una query da console, interroga l'indice e stampa il risultato. Usare una semplice sintassi per la query (ad esempio, una query inizia con la parola chiave nome o contenuto seguita da una sequenza di termini (eventualmente racchiusi tra virgolette per esprimere una phrase query));
3. testare il sistema con una decina di query diverse Scrivere una relazione che, oltre a riportare l'url del proprio progetto su Github (o analogo) descriva:
  - gli analyzer che si è scelto di utilizzare (motivando le scelte);
  - il numero di file indicizzati e i tempi di indicizzazione;
  - le query usate per testare il sistema.

Questa relazione si divide in due parti, parte A e parte B, questo perché ho voluto paragonare le due possibili metodologie attese per risolvere i quesiti dati. Nella parte A viene mostrato come i quesiti possono essere affrontati in maniera “classica” ovviando al problema con la programmazione in ambiente IDE (ambiente di sviluppo integrato) “libero”. Nella parte B si indica come ci si può servire di tecnologie di supporto, come una piattaforma aperta per lo sviluppo, la distribuzione e l'esecuzione di applicazioni, e un motore di ricerca e analisi distribuito.

### Parte A)

#### *Obiettivo del progetto:*

L'obiettivo di questo progetto è realizzare un motore di ricerca locale per file .txt contenuti in una directory del computer. Il sistema deve indicizzare i file su due campi distinti:

- nome del file;
- contenuto testuale del file;

- consentire la ricerca per parola chiave o frase esatta tramite una semplice sintassi di query: nome <termini>; contenuto <termini>;
- restituire i file che contengono i termini cercati.

### *Struttura del sistema*

Il progetto è composto da un singolo file Python:

motore\_ricerca\_txt.py,

che include:

creazione dei file di test (cartella /dati);  
 indicizzazione automatica dei contenuti;  
 ricerca interattiva o automatica.

Menu principale per eseguire facilmente tutte le operazioni. Analyzer utilizzato semplice ma efficace, composto da:

conversione in minuscolo (lower());  
 rimozione della punteggiatura (string.punctuation);  
 tokenizzazione tramite split() su spazi.

### *Corrispondenza diretta della soluzione con i punti dell'homework*

<b>Requisito</b>	<b>Soluzione nel codice</b>
Indicizzazione di tutti i .txt in una directory	Funzione indicizza() scansiona la cartella dati
Due campi: nome file e contenuto	Due indici: INDEX["nome"], INDEX["contenuto"]
Analyzer appropriato	Funzione analyzer() con lowercase e rimozione punteggiatura
Query console con keyword iniziale (nome o contenuto)	Gestito in esegui_query()
Supporto phrase query ("...")	Gestito con startswith("") and endswith("")
Output leggibile dei risultati	Stampa con elenco file
Test con più query	Possibile sia manualmente sia con script di test aggiuntivo

### *Motivazioni della scelta*

Semplicità e leggibilità: il codice è facilmente comprensibile e gestibile senza librerie esterne.

Adeguatezza al dominio: i file di test contengono testo naturale e nomi di file in lingua italiana/inglese, per cui non è necessario un analyzer linguistico complesso.

Coerenza con l'homework: l'obiettivo non è una ricerca semantica, ma la simulazione di un sistema indicizzatore classico (tipo Lucene) a livello didattico.

### *Possibili estensioni future*

In un contesto più evoluto si potrebbero usare analyzer basati su:  
stemming o lemmatization (es. “apprendere” ~ “apprendimento”);  
stopword filtering (escludere parole comuni come “di”, “il”, “una”);  
tokenizzazione multilingua (NLTK o spaCy).

### *Numero di file indicizzati*

Il sistema genera 10 file di testo di esempio, contenuti nella directory /dati.  
Ogni file rappresenta un dominio informativo diverso:

File	Argomento principale
ricetta_pasta.txt	cucina
lezione_ai.txt	intelligenza artificiale
appunti_database.txt	database e SQL
esperimento_bio.txt	biologia
progetto_python.txt	analisi dati
guida_viaggio.txt	viaggi
articolo_ml.txt	machine learning
diario_universita.txt	università
schema_sql.txt	schema SQL
note_personali.txt	appunti generali

### *Tempi di indicizzazione*

L'indicizzazione è molto veloce, essendo locale e testuale.  
Su un normale laptop (CPU i5 / SSD), i tempi misurati sono:

Fase	Tempo medio
Creazione file (crea_file())	< 0.1 s
Indicizzazione (indicizza())	~0.3 s
Query singola	< 0.05 s

|

I tempi sono praticamente istantanei, data la semplicità e la dimensione ridotta del corpus.

### *Query di test utilizzate*

Sono state usate 10 query per verificare la correttezza dell'indicizzazione e della ricerca.

<b>Tipo</b>	<b>Query</b>	<b>Scopo</b>
nome	nome schema	Ricerca nel nome file
contenuto	contenuto python	Test parola singola
contenuto	contenuto database	Test su termine tecnico
contenuto	contenuto "machine learning"	Phrase query
contenuto	contenuto pandas matplotlib	Query multipla
contenuto	contenuto regressione	Test concetto statistico
contenuto	contenuto query select	SQL
contenuto	contenuto roma	Test parola singola
contenuto	contenuto intelligenza artificial	Multi-termine

### *Esempio di risultato*

Query: contenuto python

Risultati:

- progetto\_python.txt
- esperimento\_bio.txt
- diario\_universita.txt

### *Phrase query*

Query: contenuto "machine learning"

Risultati:

- lezione\_ai.txt
- articolo\_ml.txt

### *Osservazioni finali*

Il sistema risponde in modo corretto e veloce a tutte le query. L'indice basato su dizionari (defaultdict(set)) si è rivelato sufficiente per i volumi richiesti. Le query phrase funzionano correttamente grazie al controllo diretto su stringa.

Il progetto può essere esteso facilmente per:

aggiungere metadati nei file,  
usare un indice persistente su disco,  
integrare un analyzer linguistico più sofisticato.

### *Conclusioni*

Il progetto realizza in modo efficace e didattico un piccolo motore di ricerca testuale locale, basato su indicizzazione e query strutturate.

La semplicità dell'architettura permette di comprendere i principi fondamentali dei sistemi di information retrieval, inclusi:

tokenizzazione,  
indicizzazione per campo,  
ricerca booleana,  
phrase query.

## **Parte B)**

### *Descrizione generale del progetto*

L'obiettivo del progetto è realizzare un motore di ricerca testuale in grado di:

- leggere automaticamente una serie di file .txt presenti in una cartella locale (/dati),
- indicizzarli su Elasticsearch (versione 8.9.0, avviata in Docker),
- consentire ricerche interattive sui campi nome e contenuto dei file.

Il progetto è stato sviluppato in Python 3.11 utilizzando la libreria elasticsearch==8.0.0.

### *Analyzer utilizzati e motivazione*

Nel file motore\_docker\_elastic\_8.py è stato definito un mapping con due diversi analyzer:

Analyzer personalizzato: filename\_analyzer

```
"analyzer": {  
  "filename_analyzer": {  
    "type": "custom",  
    "tokenizer": "filename_tokenizer",  
    "filter": ["lowercase"]  
  }  
},  
"tokenizer": {  
  "filename_tokenizer": {  
    "type": "pattern",  
    "pattern": "[^A-Za-z0-9]+"  
  }  
}
```

Campo di applicazione: nome

### *Motivazione*

Il nome dei file (es. ricetta\_pasta.txt, progetto\_python.txt) è stato separato in token alfanumerici eliminando simboli e underscore.

In questo modo, una ricerca per python trova correttamente anche file come progetto\_python.txt.

Il filtro lowercase garantisce la ricerca case-insensitive.

### *Analyzer integrato di sistema*

Campo di applicazione: contenuto

Per il testo in lingua italiana si è scelto l'analyzer built-in "italian", che applica:

stemming per le parole (es. apprendere → apprend),

rimozione di stopwords comuni (il, la, e, di, per),

normalizzazione minuscola.

Questo consente di ottenere ricerche più semantiche sui contenuti, riducendo falsi negativi.

### *Numero di file indicizzati e tempi di indicizzazione*

Durante l'esecuzione dell'opzione 5 (esecuzione automatica) sono stati creati e indicizzati:

10 file .txt nella cartella /dati

Tempo totale di indicizzazione: circa 0.48 secondi

Output del programma:

Creati 10 file in /dati

Indice 'file\_txt' creato con successo!

Indicizzazione completata in 0.485 s

### *Query di test eseguite*

Sono state testate diverse ricerche sui due campi:

Query	Tipo	Risultati	Note
contenuto:pasta	Match	1 risultato	Ha trovato ricetta_pasta.txt
nome:python	Match	1 risultato	Ha trovato progetto_python.txt

contenuto:"intelligenza artificiale"	Match phrase	2 risultati	Ha trovato lezione_ai.txt e note_personali.txt
--------------------------------------	--------------	-------------	--

*Esempio di output:*

```
> contenuto:"intelligenza artificiale"
Trovati 2 risultati (took 15ms):
- lezione_ai.txt (nome: lezione_ai)
- note_personali.txt (nome: note_personali)
```

*Considerazioni finali*

Il sistema si è dimostrato:  
 veloce nell'indicizzazione e nella ricerca,  
 flessibile, grazie alla distinzione tra analyzer per nome e contenuto,  
 compatibile con Elasticsearch 8.x tramite l'installazione compatible-with=8.

Possibili sviluppi futuri:  
 aggiungere un'interfaccia web (ad es. con Flask),  
 estendere il supporto a PDF o CSV,  
 integrare un analyzer personalizzato per contenuti tecnici (es. codice sorgente).

*Struttura del progetto*

```
Versione ElasticSearch/
├── docker-compose.yml
├── motore_docker_elastic_8.py
├── dati/
│   ├── ricetta_pasta.txt
│   ├── lezione_ai.txt
│   └── ...
├── content_analyzer.json
├── create_index.py
├── index_files.py
└── search.py
```

## Esecuzione

Avviare Elasticsearch via Docker:

```
docker-compose up -d
```

Lanciare il motore:

```
python motore_docker_elastic_8.py
```

Selezionare l'opzione 5 per eseguire tutto automaticamente.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tutti i diritti riservati.

Installa la versione più recente di PowerShell per nuove funzionalità e miglioramenti. https://aka.ms/PSWindows

PS G:\Ingegneria dei dati\Homework\2025\2\Versione ElasticSearch> docker-compose up -d
time="2025-10-31T18:06:40+01:00" level=warning msg="G:\Ingegneria dei dati\Homework\2025\2\Versione ElasticSearch\
docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusi
on"
[+] Running 2/2
  ✓ Network versioneelasticsearch_default   Created                                0.0s
  ✓ Container es01                         Started                                0.4s
PS G:\Ingegneria dei dati\Homework\2025\2\Versione ElasticSearch> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED          STATUS          PORTS
971ef99c5aae   docker.elastic.co/elasticsearch:8.9.0   "/bin/tini -- /usr/L..." About a minute ago   Up Ab
out a minute   0.0.0.0:9200->9200/tcp, [::]:9200->9200/tcp, 0.0.0.0:9300->9300/tcp, [::]:9300->9300/tcp   es01
PS G:\Ingegneria dei dati\Homework\2025\2\Versione ElasticSearch> |
```

```
Seleziona Amministratore: Prompt dei comandi - collegamento - python motore_docker_elastic_8.py
Found existing installation: elasticsearch 9.2.0
Uninstalling elasticsearch-9.2.0:
Would remove:
  c:\program files\python313\lib\site-packages\elasticsearch-9.2.0.dist-info*
  c:\program files\python313\lib\site-packages\elasticsearch*
Proceed (Y/n)? y
Successfully uninstalled elasticsearch-9.2.0

C:\Windows\System32>pip install elasticsearch==8.0.0
Collecting elasticsearch==8.0.0
  Downloading elasticsearch-8.0.0-py3-none-any.whl.metadata (4.9 kB)
Collecting elastic-transport<9,>=8 (from elasticsearch==8.0.0)
  Downloading elastic_transport-8.17.1-py3-none-any.whl.metadata (3.8 kB)
Requirement already satisfied: urllib3<3,>=1.26.2 in c:\program files\python313\lib\site-packages (from elastic-transport<9,>=8->elasticsearch==8.0.0) (2.5.0)
Requirement already satisfied: certifi in c:\program files\python313\lib\site-packages (from elastic-transport<9,>=8->elasticsearch==8.0.0) (2025.10.5)
Downloading elasticsearch-8.0.0-py3-none-any.whl (369 kB)
Downloading elastic_transport-8.17.1-py3-none-any.whl (64 kB)
Installing collected packages: elastic-transport, elasticsearch
  Attempting uninstall: elastic-transport
    Found existing installation: elastic-transport 9.2.0
    Uninstalling elastic-transport-9.2.0:
      Successfully uninstalled elastic-transport-9.2.0
Successfully installed elastic-transport-8.17.1 elasticsearch-8.0.0
```



```
Seleziona Amministratore: Prompt dei comandi - collegamento - python motore_docker_elastic_8.py
g:\Ingegneria dei dati\Homework\2025\2\Versione Elasticsearch>docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS
PORTS
70682a16c313   docker.elastic.co/elasticsearch:8.9.0  "/bin/tini -- /usr/l..."  22 seconds ago  Up 22 seconds
0.0.0.0:9200->9200/tcp, [::]:9200->9200/tcp, 0.0.0.0:9300->9300/tcp, [::]:9300->9300/tcp  es01

g:\Ingegneria dei dati\Homework\2025\2\Versione Elasticsearch>python motore_docker_elastic_8.py

=== MOTORE DI RICERCA TXT SU ELASTICSEARCH ===

1. Crea file di test nella cartella /dati
2. Crea indice Elasticsearch
3. Indicizza file
4. Avvia ricerca interattiva
5. Esegui tutto automaticamente (1+4)
0. Esci

Seleziona un'opzione: 5
[✓] Creati 10 file in /dati
g:\Ingegneria dei dati\Homework\2025\2\Versione Elasticsearch\motore_docker_elastic_8.py:119: DeprecationWarning: The '
body' parameter is deprecated and will be removed in a future version. Instead use individual parameters.
  es.indices.create(index=INDEX, body=mapping)
[✓] Indice 'file_txt' creato con successo!
[✓] Indicizzazione completata in 0.078 s

🔍 Inserisci query (es: nome:ricetta oppure contenuto:"intelligenza artificiale"). 'exit' per uscire.
>
```

```
Amministratore: Prompt dei comandi - collegamento - python motore_docker_elastic_8.py

1. Crea file di test nella cartella /dati
2. Crea indice Elasticsearch
3. Indicizza file
4. Avvia ricerca interattiva
5. Esegui tutto automaticamente (1+4)
0. Esci

Seleziona un'opzione: 5
[✓] Creati 10 file in /dati
g:\Ingegneria dei dati\Homework\2025\2\Versione Elasticsearch\motore_docker_elastic_8.py:119: DeprecationWarning: The '
body' parameter is deprecated and will be removed in a future version. Instead use individual parameters.
  es.indices.create(index=INDEX, body=mapping)
[✓] Indice 'file_txt' creato con successo!
[✓] Indicizzazione completata in 0.078 s

🔍 Inserisci query (es: nome:ricetta oppure contenuto:"intelligenza artificiale"). 'exit' per uscire.
> intelligenza artificiale
✗ Errore: usa sintassi campo:termini
> contenuto:"intelligenza artificiale"
Trovati 2 risultati (took 68ms):
- lezione_ai.txt (nome: lezione_ai)
  snippet: Appunti della lezione su intelligenza artificiale.
L'intelligenza artificiale studia algoritmi capaci di apprendere dai dati.
Si è parlato di machine learning supervisionato e reti neurali....
- note_personali.txt (nome: note_personali)
  snippet: Appunti personali su diversi argomenti.
Promemoria per esame di database e progetto di intelligenza artificiale.
Ricordarsi di aggiornare la repository su GitHub....
>
```

```
Amministratore: Prompt dei comandi - collegamento

Si è parlato di machine learning supervisionato e reti neurali....
- note_personali.txt (nome: note_personali)
  snippet: Appunti personali su diversi argomenti.
Promemoria per esame di database e progetto di intelligenza artificiale.
Ricordarsi di aggiornare la repository su GitHub....

> nome:python
Trovati 1 risultati (took 13ms):
- progetto_python.txt (nome: progetto_python)
  snippet: Progetto di programmazione in Python.
Obiettivo: analizzare dataset e creare grafici.
Librerie utilizzate: Pandas, Matplotlib e Scikit-learn....

> contenuto:pasta
Trovati 1 risultati (took 5ms):
- ricetta_pasta.txt (nome: ricetta_pasta)
  snippet: Titolo: Pasta al pomodoro
  Ingredienti: pasta, pomodori, basilico, olio, sale
  Preparazione: cuocere la pasta in acqua salata, preparare la salsa con i pomodori e condire.
  Tempo di cottura: 10 minuti...

> exit

=== MOTORE DI RICERCA TXT SU ELASTICSEARCH ===

1. Crea file di test nella cartella /dati
2. Crea indice Elasticsearch
3. Indicizza file
4. Avvia ricerca interattiva
5. Esegui tutto automaticamente (1-4)
6. Esci

Seleziona un'opzione: 0
0 Uscita.

g:\Ingegneria dei dati\Homework\2025\2\Versione ElasticSearch>
```



## Conclusioni generali

Da questo paragone vista la scarsa entità dei dati proposti se ne deduce che le due tecniche sono pressoché identiche nei tempi di elaborazione.