

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC BÁCH KHOA



**BÀI BÁO CÁO BÀI TẬP LỚN**  
**MÔN HỌC: LẬP TRÌNH NÂNG CAO (CO2039)**

---

**Nhóm 502 Lớp L01**

**Đề tài: Xây dựng web/app tra cứu dữ liệu sao kê của MTTQ VN**

---

GVHD: Lê Đình Thuận  
SV thực hiện: Nguyễn Công Minh – 2312080  
Nguyễn Việt Hùng – 2011315  
Nguyễn Việt Trung – 2014887  
Nguyễn Thế Vinh – 2115302  
Nguyễn Trung Nguyên – 2011710  
Huỳnh Tuấn Anh – 2012587

Tp. Hồ Chí Minh, Tháng 05/2024



## DANH SÁCH THÀNH VIÊN VÀ ĐÁNH GIÁ

Tên	MSSV	Công việc	Ghi chú
Nguyễn Công Minh	2312080	Hệ thống tìm kiếm và viết báo cáo	Hoàn thành
Nguyễn Việt Hùng	2011315	Giao diện và kiểm tra	Hoàn thành
Nguyễn Việt Trung	2014887	Giao diện và kiểm tra	Hoàn thành
Nông Thế Vinh	2115302	Hệ thống tìm kiếm và API	Hoàn thành
Nguyễn Trung Nguyên	2011710	Slide	Hoàn thành
Huỳnh Tuấn Anh	2012587	Hệ thống tìm kiếm và API	Hoàn thành

## I Mô tả bài toán cần giải quyết:

Xây dựng một trang web/app có giao diện để có thể tra cứu dữ liệu sao kê từ mặt trận tổ quốc Việt Nam từ tên, thông tin giao dịch và số tiền đã giao dịch

Tổng quan về dữ liệu (5 dòng đầu tiên của dataset):

date_time	trans_no	credit	debit	detail
01/09/2024_5215.97152	1	3000	0	267515.010924.122904.NGUYEN THI MAO ...
01/09/2024_5219.24714	2	10000	0	018806.010924.213139.chuc mung ngay ...
02/09/2024_5212.22965	3	10000	0	055464.020924.064157.Ung Ho Nha Nuoc ...
02/09/2024_5245.21394	4	10000	0	MBVCB.6924605040.gia dinh Dung Thuy ...
02/09/2024_5078.73943	5	50000	0	MBVCB.6925071164.mung ngay quoc ...

Trong đó:

- date\_time: thời gian xảy ra giao dịch
- trans\_no: thứ tự của giao dịch trong tập dữ liệu
- credit: số tiền giao dịch
- debit: số tiền nợ
- detail: thông tin lời nhắn chi tiết của giao dịch

## II Full-text search:

Sau khi tham khảo nhiều công cụ tìm kiếm văn bản phổ biến hiện nay ở cộng đồng lập trình viên, thì tụi em thấy được rằng elasticsearch rất được ưa chuộng bởi hiệu suất cao cũng như độ chính xác tốt. Dựa vào đó, tụi em đã quyết định mô phỏng lại cấu trúc dữ liệu và giải thuật của công cụ elasticsearch.

Cấu trúc dữ liệu chính tụi em sử dụng là Inverted Index kết hợp với HashTable để có thể truy xuất nhanh với độ phức tạp trung bình là  $O(1)$

Giải thuật có thể chia thành 3 bước chính:

1. Tiền xử lý dữ liệu - preprocess
2. Xây dựng cấu trúc inverted index
3. Trả lời các truy vấn

### 1 Tiền xử lý dữ liệu - preprocess:

#### 1.1 Xử lý dạng chuỗi trong detail:

- Kết hợp tất cả mọi thông tin có trong dataset thành một chuỗi duy nhất cho mỗi giao dịch. Ví dụ như dòng đầu tiên ta sẽ được chuỗi:

"01/09/2024\_5215.97152 1 3000 0 267515.010924.122904.NGUYEN THI MAO Chuyen tien"

- Loại bỏ những dấu câu không có nhiều ý nghĩa trong việc tìm kiếm như dấu ",", ".", "!", ...

"01 09 2024 5215 97152 1 3000 0 267515 010924 122904 NGUYEN THI MAO Chuyen tien"

- Chuyển toàn bộ chữ cái sang chữ viết thường để đồng bộ việc lưu trữ và tìm kiếm

"01 09 2024 5215 97152 1 3000 0 267515 010924 122904 nguyen thi mao chuyen tien"

- Tách thành từng từ/ cụm từ. Đa số phần thông tin giao dịch chuyển khoản trong tập dữ liệu cần được tìm kiếm được viết dưới dạng tiếng việt không dấu, nếu tách các từ ra bởi khoảng trắng sẽ mang lại hiệu quả không tốt vì cấu trúc tiếng việt có những cụm từ thay vì những từ đơn như tiếng anh. Dẫn đến quyết định sử dụng thư viện pyvi - ViTokenizer - một thư viện chuyên xử lý ngôn ngữ tiếng việt để chia nhỏ các câu văn thành các từ đơn một cách chính xác, hỗ trợ các bước phân tích tiếp theo như tìm kiếm, phân loại, và khai thác thông tin.
- Sau khi xử lý dữ liệu dạng chuỗi, thu được một danh sách các từ/ cụm từ đã được chuẩn hóa  

```
["01", "09", "2024", "5215", "97152", "1", "3000", "0", "267515", "010924", "122904", "nguyen",  
"thi", "mao", "chuyen", "tien"]
```

## 1.2 Đếm tần số xuất hiện của các từ trong thông tin mỗi giao dịch:

- Xây dựng một tập các từ không lặp lại của toàn bộ các thông tin của giao dịch nhằm mục đích ánh xạ từ "từ" sang "số" để tăng tốc độ truy xuất
- Đếm số lần xuất hiện của mỗi từ và tiến hành ánh xạ các từ trong mỗi thông tin giao dịch đến số lần xuất hiện của từ đó
  - 01: 1
  - 09: 1
  - 2024: 1
  - 5215: 1
  - 97152: 1
  - 3000: 1
  - 0: 1
  - 267515: 1
  - 010924: 1
  - 122904: 1
  - nguyen: 1
  - thi: 1
  - mao: 1
  - chuyen: 1
  - tien: 1
- Đánh số thứ tự cho các từ duy nhất trong tập hợp đã xây dựng ở trên và chuyển đổi ánh xạ từ "từ" sang "số" để tăng tốc độ truy vấn về sau
- Sau khi hoàn thành các bước, ta thu được một HashTable với key là id của mỗi giao dịch để đánh số từ 0 với value là một HashTable khác với key là id của từ trong tập từ duy nhất và value là số lần xuất hiện tương ứng của từ đó trong thông tin giao dịch

## 2 Xây dựng Inverted Index

### 2.1 Sơ lược về Inverted Index:

Inverted Index là một cấu trúc dữ liệu được sử dụng trong các hệ thống truy xuất thông tin để truy xuất hiệu quả các tài liệu hoặc trang web chứa một thuật ngữ hoặc tập hợp các thuật ngữ cụ thể.

Trong một Inverted Index, chỉ mục được tổ chức theo các thuật ngữ (từ), và mỗi thuật ngữ chỉ đến một danh sách các tài liệu hoặc trang web chứa thuật ngữ đó.

Xem ví dụ sau:

1. Document 1: The quick brown fox jumped over the lazy dog.

2. Document 2: The lazy dog slept in the sun.

Inverted Index sẽ có dạng như sau:

- The  $\rightarrow$  {Document 1 : 2}, {Document 2 : 1}
- Quick  $\rightarrow$  {Document 1 : 1}
- Brown  $\rightarrow$  {Document 1 : 1}
- Fox  $\rightarrow$  {Document 1 : 1}
- Jumped  $\rightarrow$  {Document 1 : 1}
- Over  $\rightarrow$  {Document 1 : 1}
- Lazy  $\rightarrow$  {Document 1 : 1}, {Document 2 : 1}
- Dog  $\rightarrow$  {Document 1 : 1}, {Document 2 : 1}
- Slept  $\rightarrow$  {Document 2 : 1}
- In  $\rightarrow$  {Document 2 : 1}
- Sun  $\rightarrow$  {Document 2 : 1}

Có hai loại Inverted Index:

1. Record-Level Inverted Index: Record-Level Inverted Index chứa danh sách các tham chiếu đến các tài liệu cho mỗi từ.
2. Word-Level Inverted Index: Word-Level Inverted Index bổ sung thêm vị trí của mỗi từ trong một tài liệu. Loại sau cung cấp nhiều chức năng hơn nhưng cần nhiều công suất xử lý và không gian lưu trữ hơn để được tạo ra.

## 2.2 Xây dựng Inverted Index cho hệ thống tìm kiếm:

---

```
def __build_inverted_index(self):  
    """  
        Build inverted index for transactions  
        __inverted_index {__term_id : {doc_id: frequency}}  
    """  
    for doc_id, term_freqs in self.__doc_term_freq.items():  
        for t_id, freq in term_freqs.items():  
            self.__inverted_index[t_id][doc_id] = freq;
```

---

- Duyệt lần lượt từng doc\_id và tần số của các từ trong tập doc\_term\_freq đã được xây dựng
- Gán lại đúng cấu trúc của Inverted Index để dễ kiểm tra cũng như truy xuất tần suất xuất hiện của một từ trong một giao dịch

## 3 Tìm kiếm - trả lời truy vấn:

- Chuẩn hóa chuỗi truy vấn của người dùng về đúng chuẩn đã được chuẩn bị trước
- Dùng Inverted Index để lấy ra những truy vấn có từ liên quan đến lời truy vấn
- Sử dụng công thức tính điểm để chọn ra được truy vấn có độ liên quan cao nhất

### 3.1 DF - IDF:

**Document Frequency (DF)** của một từ là số lượng tài liệu trong tập hợp tài liệu chứa từ đó.

$$DF(t) = \text{Số lượng tài liệu chứa từ } t$$

**Inverse Document Frequency (IDF)** - Tần Suất Ngược Tài Liệu

Inverse Document Frequency (IDF) đo lường tầm quan trọng của từ trong toàn bộ tập hợp tài liệu. Từ càng hiếm trong tập tài liệu, IDF của nó càng cao.

$$IDF(t) = \log\left(\frac{N}{DF(t)}\right)$$

**Term Frequency-Inverse Document Frequency (TF-IDF)**

TF-IDF kết hợp giữa Term Frequency (TF) (tần suất từ trong tài liệu) và IDF để đánh giá tầm quan trọng của từ trong tài liệu đó so với toàn bộ tập hợp tài liệu.

$$DF - IDF(t, d) = DF(t, d) \times IDF(t)$$

**Nhược điểm:** Tuy TF-IDF là một công cụ mạnh mẽ và đơn giản để đánh giá tầm quan trọng của từ, nó cũng có một số nhược điểm

1. Không xử lý ngữ cảnh và ngữ nghĩa: DF - IDF tính toán dựa trên tần số xuất hiện của mỗi từ nên đã bỏ qua ngữ cảnh và ngữ nghĩa của câu
2. Bỏ qua thứ tự từ và cấu trúc ngữ pháp
3. Rơi vào vấn đề thường xuyên (Common Words): nếu một từ trong giao dịch lặp lại nhiều lần và vô tình trùng với từ trong truy vấn sẽ dẫn đến giao dịch đó được đánh giá cao hơn thay vì xem xét toàn bộ từ và độ dài chuỗi

### 3.2 BM25 - Best Matching 25:

**BM25 - Best Matching 25** được phát triển dựa trên TF - IDF để xếp hạng tài liệu dựa trên sự phù hợp giữa truy vấn và tài liệu với các cải tiến để tăng cường hiệu quả trong các nhiệm vụ tìm kiếm văn bản và cũng là thước đo được sử dụng trong hệ thống tìm kiếm

**BM25 - Best Matching 25** khắc phục điểm yếu của TF - IDF bằng cách chuẩn hóa độ dài của tài liệu và giới hạn sự ảnh hưởng của TF đến thang đo tổng thể

**Chuẩn hóa độ dài tài liệu (Normalization):**

$$BM25 = \sum_{t \in q} IDF(t) \times \frac{TF(t, d) \times (k_1) + 1}{TF(t, d) + k_1 \times (1 - b + b \times \frac{|d|}{avgdl})}$$

Trong đó:

- $|d|$  là độ dài của tài liệu  $d$
- $avgdl$  là độ dài trung bình của toàn bộ tài liệu
- $k_1, b$  là các tham số điều chỉnh

**Sự bão hòa tần suất từ**

**Nhược Điểm TF-IDF:** TF-IDF không giới hạn ảnh hưởng của TF, nghĩa là sự tăng lên của TF vẫn dẫn đến sự tăng điểm số tuyến tính, không phản ánh thực tế rằng sau một mức độ nào đó, việc tăng TF không còn làm tăng độ phù hợp của tài liệu.

**Bão Hòa Tần Suất Từ:** BM25 giới hạn ảnh hưởng của TF thông qua tham số  $k_1$ . Khi TF tăng lên, ảnh hưởng của nó vào điểm số sẽ giảm dần, phản ánh rằng sau một mức độ nhất định, việc xuất hiện

thêm của từ không làm tăng sự phù hợp một cách đáng kể

### Xử lý các từ phổ biến và hiếm

Mặc dù TF-IDF giảm giá trị của các từ phổ biến thông qua IDF, nó không thể xử lý tốt các từ cực kỳ phổ biến hoặc cực kỳ hiếm một cách linh hoạt

IDF Tinh Chỉnh: BM25 sử dụng một công thức IDF được tinh chỉnh để cân bằng hơn trong việc xử lý các từ phổ biến và hiếm, giúp cải thiện chất lượng xếp hạng.

$$IDF(t) = \log\left(\frac{N - n(t) + 0.5}{n(t) + 0.5} + 1\right)$$

Trong đó:

- N là tổng số tài liệu
- n(t) là số tài liệu chứa từ t

### 3.3 Xây dựng thước đo lường cho hệ thống tìm kiếm:

Tại em sử dụng BM250kapi từ thư viện rank\_bm25 để có được hàm tính điểm BM25 đã trình bày ở trên

Khởi tạo BM25:

---

```
def __build_bm25(self):
    self.__corpus = []
    for doc_id in sorted(self.__documents.keys()):
        terms = [self.__id_term[tid] for tid in self.__doc_term_freq[doc_id].keys()]
        self.__corpus.append(terms)

    self.bm25 = BM250kapi(self.__corpus)
```

---

Hàm search:

---

```
def search(self, query, num = 10) -> list:
    """
    @ Params: query - string: query need to find
    Return 5 document most relevant
    """

    query_terms = self.__preprocess_text(query)

    candidate_docs = set()

    for term in query_terms:
        term_id = self.__term_id.get(term)
        if term_id is not None:
            docs = self.__inverted_index[term_id].keys()
            candidate_docs.update(docs)

    if not candidate_docs:
        return []

    candidate_docs = list(candidate_docs)
    candidate_indices = [doc_id for doc_id in candidate_docs]
    scores = self.bm25.get_batch_scores(query_terms, candidate_indices)

    doc_scores = list(zip(candidate_docs, scores))

    ranked_docs = sorted(doc_scores, key=lambda x: x[1], reverse=True)[:num]
```

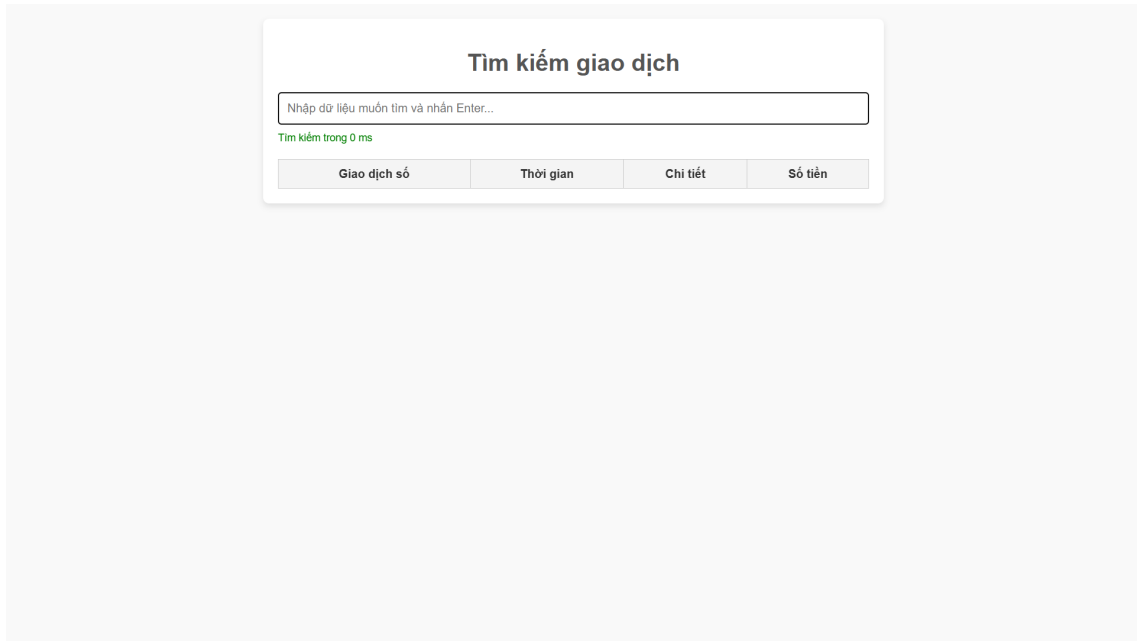
---



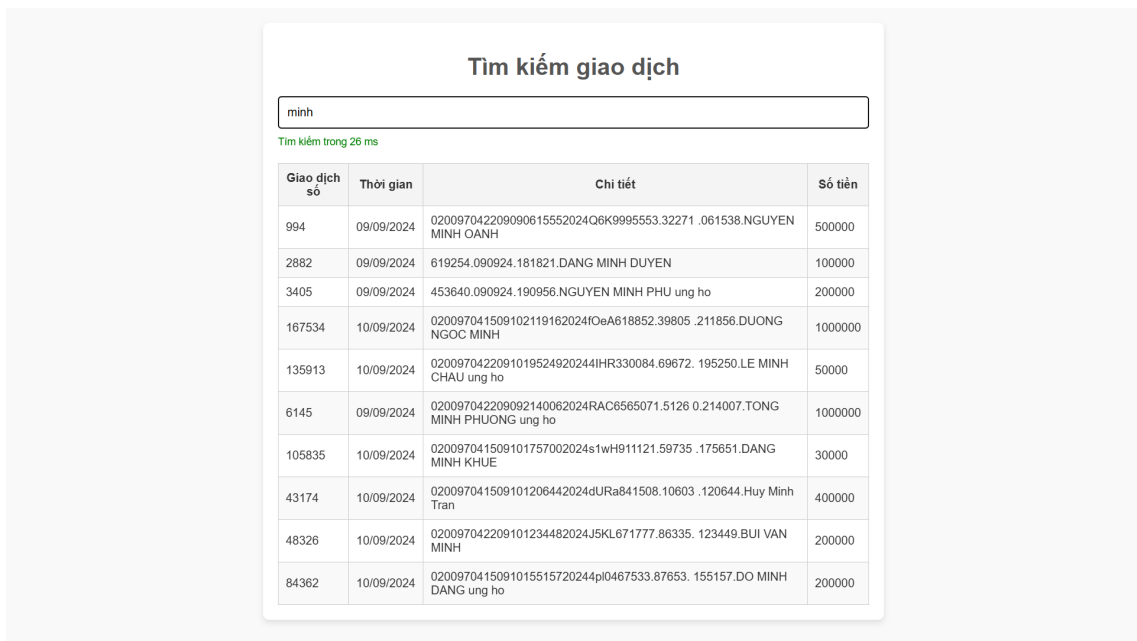
```
return ranked_docs
```

### III Vận hành trang web:

#### 1 Giao diện - frontend:



Giao diện của chúng em khá đơn giản, tập trung vào mục đích tìm kiếm là chính, chỉ cần chạy trang web ở máy cá nhân, sau đó nhập thông tin của giao dịch cần tìm kiếm trang web sẽ trả về 10 giao dịch có độ liên quan cao nhất





## 2 Backend

Tụi em dùng ngôn ngữ lập trình python để viết nên hệ thống tìm kiếm cũng như api nên phải đảm bảo máy tính cá nhân đã tải python phiên bản mới nhất từ trang web <https://www.python.org/downloads/>

Không những vậy phải đảm bảo đã cài đặt các thư viện sau:

**Thư viện hỗ trợ tính toán:**

- csv: pip install csv
- collection: pip install collection
- pyvi: pip install pyvi
- rank\_bm25: pip install rank\_bm25

**Thư viện hỗ trợ API:**

- os: pip install os
- flask: pip install flask
- pyvi: pip install pyvi
- flask\_cors: pip install flask\_cors

---

```
from flask import Flask, request, jsonify
from fullTextSearch import fullTextSearch
import os
from flask_cors import CORS

app = Flask(__name__)
csv_filepath = os.path.join(os.path.dirname(__file__), 'chuyen_khoan.csv')
fts = fullTextSearch(filename=csv_filepath)
fts.search("chuyen tien 200000 09/09/2024")

CORS(app)

@app.route('/', methods=['GET'])
def home():
    return jsonify({"message": "Welcome to the Transaction Full Text Search API"}), 200

@app.route('/search', methods=['POST'])
def search_transactions():
    data = request.get_json()
    if not data or 'query' not in data:
        return jsonify({"error": "Missing query"}), 400

    query = data['query'].strip()
    if not query:
        return jsonify({"error": "Do not let query empty"}), 400

    ranked_docs = fts.search(query)

    results = []
    for doc_id, score in ranked_docs:
        transaction_info = fts.get_transaction_info(doc_id)
        if transaction_info:
            results.append({
                **transaction_info
            })

    return jsonify({
```

```
"query": query,
"results": results
}), 200

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000, debug=False)
```

Ở bước tạo đối tượng fts từ class fulltextSearch, tui em có thực hiện 1 bước tìm kiếm giả để làm "âm" hệ thống tìm kiếm, do một số hàm từ thư viện được khai báo thêm có cơ chế "lazy propagation" mà tui em không kiểm soát được hoàn toàn. Nên giải pháp được đưa ra là sẽ giả tìm kiếm lần đầu để khởi động các phương thức/ hàm, tránh ảnh hưởng đến trải nghiệm của người dùng web

## IV Thảo luận và mở rộng:

So sánh Inverted Index với B-tree:

	B-tree	Inverted Index + BM25
<b>Cấu trúc dữ liệu</b>	Sử dụng cấu trúc cây cân bằng (B-tree) để lưu trữ và truy xuất dữ liệu có thứ tự.	Sử dụng cấu trúc <b>inverted index</b> để lưu trữ danh sách tài liệu chứa mỗi từ khóa, kết hợp với thuật toán BM25 để xếp hạng.
<b>Phương pháp tìm kiếm</b>	Tìm kiếm tuần tự hoặc theo thứ tự cây, có thể sử dụng các phép toán như range queries.	Tìm kiếm thông qua danh sách ngược (inverted list) của từ khóa và xếp hạng tài liệu dựa trên BM25.
<b>Khả năng xếp hạng kết quả</b>	Thường không có cơ chế xếp hạng kết quả; trả về các tài liệu chứa từ khóa một cách không thứ tự.	Có khả năng xếp hạng kết quả dựa trên mức độ phù hợp với truy vấn thông qua BM25.
<b>Hiệu suất</b>	Tìm kiếm có thể chậm khi dữ liệu lớn do phải duyệt cây B-tree toàn bộ hoặc phần lớn cây.	Tìm kiếm nhanh hơn nhờ inverted index, đặc biệt là với các truy vấn phức tạp và tập dữ liệu lớn.
<b>Tốc độ xây dựng index</b>	Tốc độ xây dựng index tương đối nhanh đối với các dữ liệu có thứ tự và nhỏ đến vừa.	Xây dựng inverted index có thể tốn thời gian hơn, đặc biệt với tập dữ liệu rất lớn, nhưng thường được tối ưu hóa cho việc đọc.
<b>Khả năng mở rộng</b>	Khó mở rộng với dữ liệu rất lớn do cấu trúc cây có thể trở nên phức tạp và tốn tài nguyên.	Dễ dàng mở rộng với dữ liệu lớn, đặc biệt khi sử dụng các hệ thống phân tán như Elasticsearch hoặc Solr.
<b>Bộ nhớ</b>	Yêu cầu bộ nhớ thấp hơn vì chỉ cần lưu trữ cấu trúc cây.	Yêu cầu bộ nhớ cao hơn do phải lưu trữ inverted index và các tham số BM25 cho mỗi từ khóa và tài liệu.
<b>Triển khai</b>	Đơn giản và trực quan để triển khai, đặc biệt trong các hệ thống cơ sở dữ liệu truyền thống.	Yêu cầu kiến thức về xây dựng inverted index và hiểu biết về thuật toán BM25; phức tạp hơn trong triển khai ban đầu.