검색엔진 작동 원리부터 응용까지

만들며 배우는 검색엔진 원리와 응용

김남준

오늘 할 내용

- 검색엔진 (Search Engine) 작동 원리
- 응용 (한글 초성 검색)
- 과제

사전 요구 사항

- Python을 **사용 해 보았다**.

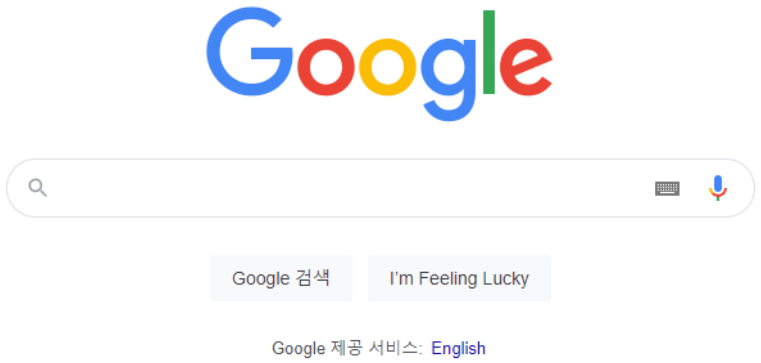
시작 전

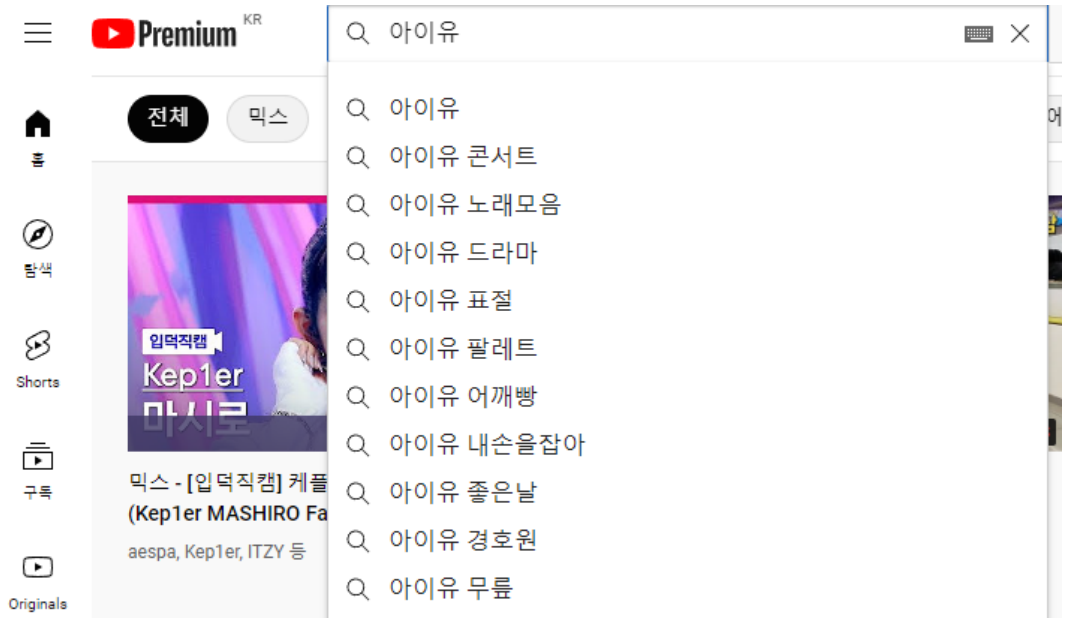
- 검색엔진은 사드세요..

비싼거..

검색 엔진?

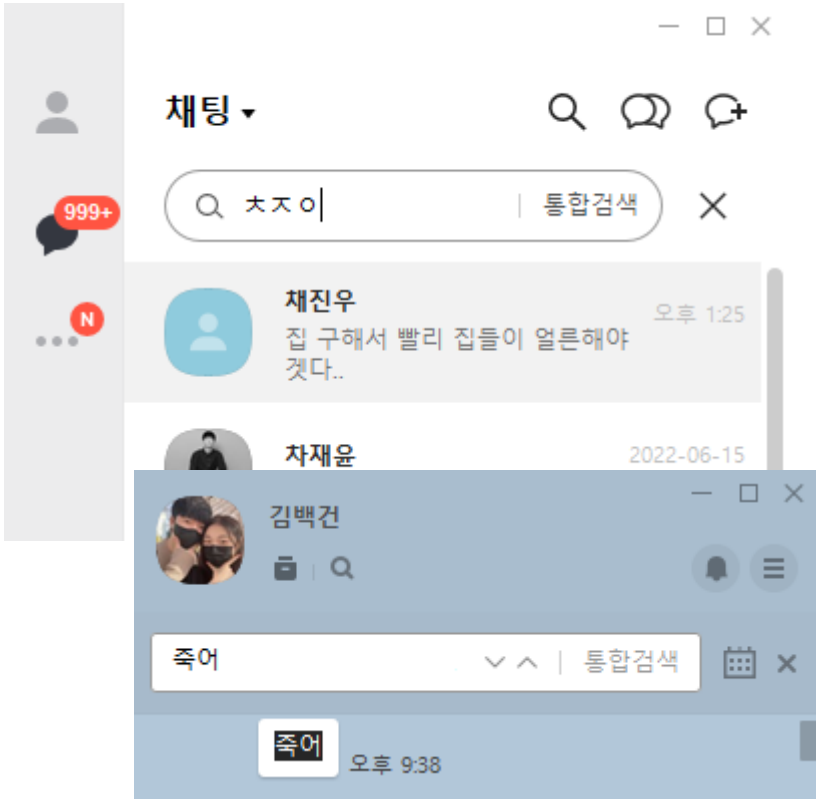
- 우리는 매일 검색 엔진을 사용합니다

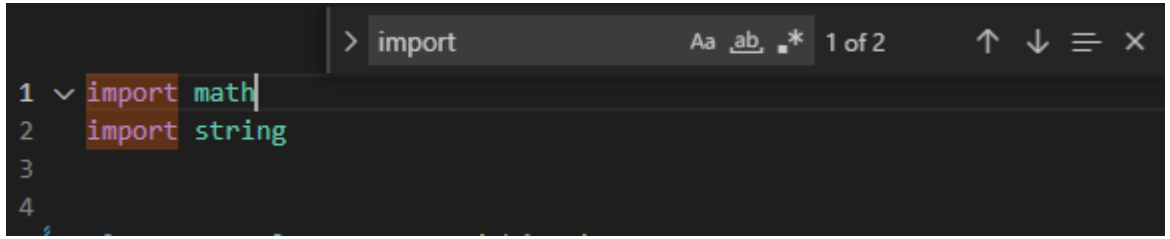


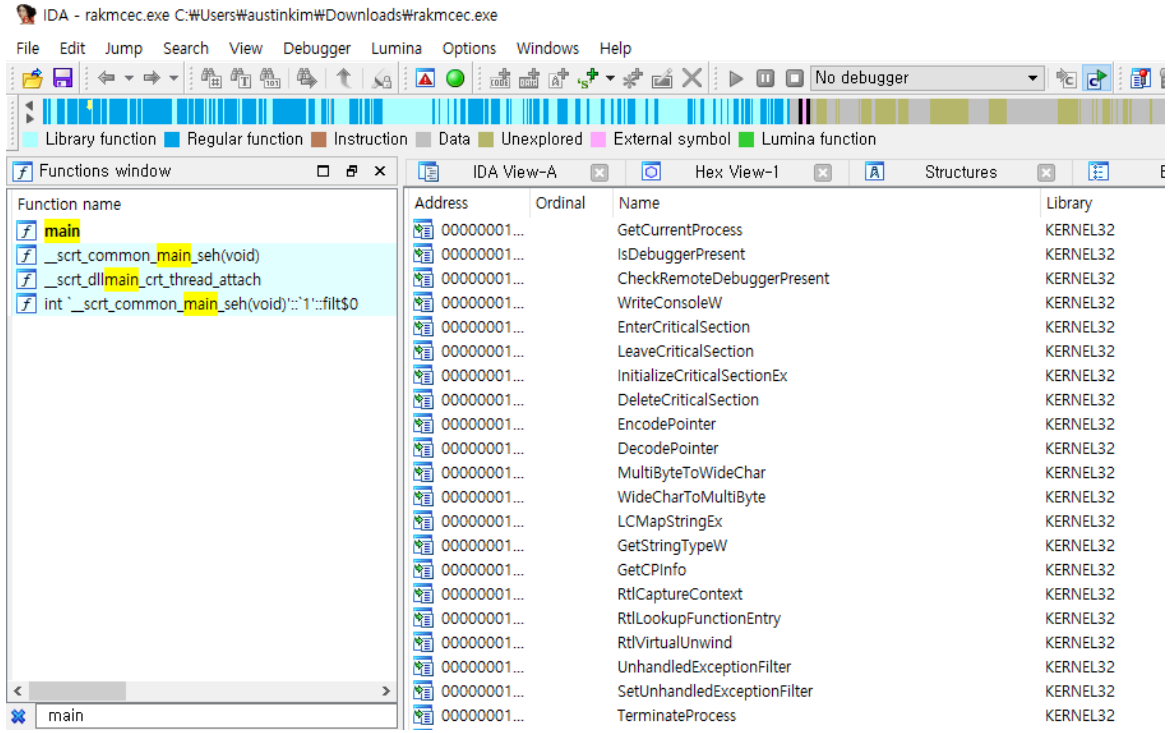


검색 엔진?

- 진짜 매일 사용합니다







왜 검색엔진을 사용해야 하는가?

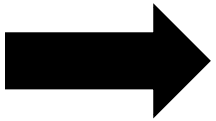
- 검색 없이 내가 원하는 정보를 찾기 **매우** 어려움

처음에는

- 정보를 수집한 뒤 정리해서 저장 해 두기 시작 (데이터의 모음)
- 이걸 우린 **데이터베이스(Database)**라고 부르기로 했어요.

```
SQL Query

SELECT userid, name, address
FROM service_users
WHERE userid = 'austinkim';
```



SQL Query Result

userid	name	address
austinkim	Namjun Kim	605 Skiff Street, North Haven, CT 06473

근데

- 요구사항이 점점 많아지면서 검색에 한계가 발생합니다.

예를 들자면?

- "경기도 안양시" 와 "경기도 광주시"에 사는 사람을 검색하고 싶어요. (LIKE 문 해결)
  만약 중국에서 이 서비스를 한다면? (중국 인구 14.02억, 2020년 기준)
- 사람 이름을 초성으로만 검색할 수 있으면 좋겠습니다.
- 주소를 입력하면 그 주소와 가장 비슷한 곳에 사는 사람을 찾고 싶어요 (랭킹)
- 기타등등...

그래서 오늘 배워볼 것은

- 이러한 문제를 해결하기 위해 사람들이 만들어 둔 좋은 기술들을 배워보는 것.
- 그리고 그 기술을 내 손으로 직접 만들어 보는 과정

- 다시 한번 말하지만, 실제로 검색 엔진을 써야 한다면 제발 사드세요.

검색 엔진의 3요소

- 수집 (Crawling)
- 색인 (Indexing)
- 질의 (Searching)

수집

- Crawling (크롤링)
- 색인하고 질의할 정보를 가져오는 과정.
- 우리가 크롤러(ex. 웹 크롤러)라고 부르는 것들이 여기에 속함.
- 외부 서비스 API를 사용하거나, 자체 정보를 통해 수집할 수 있음.

색인

- Indexing (인덱싱)
- 검색을 더욱 빠르게 하기 위해서 데이터의 특성이나 정보를 저장하는 행위

예시

- 우리는 이미 인덱싱을 실생활에서 많이 보고 있습니다.
- 책이나 논문, 단어집 등등

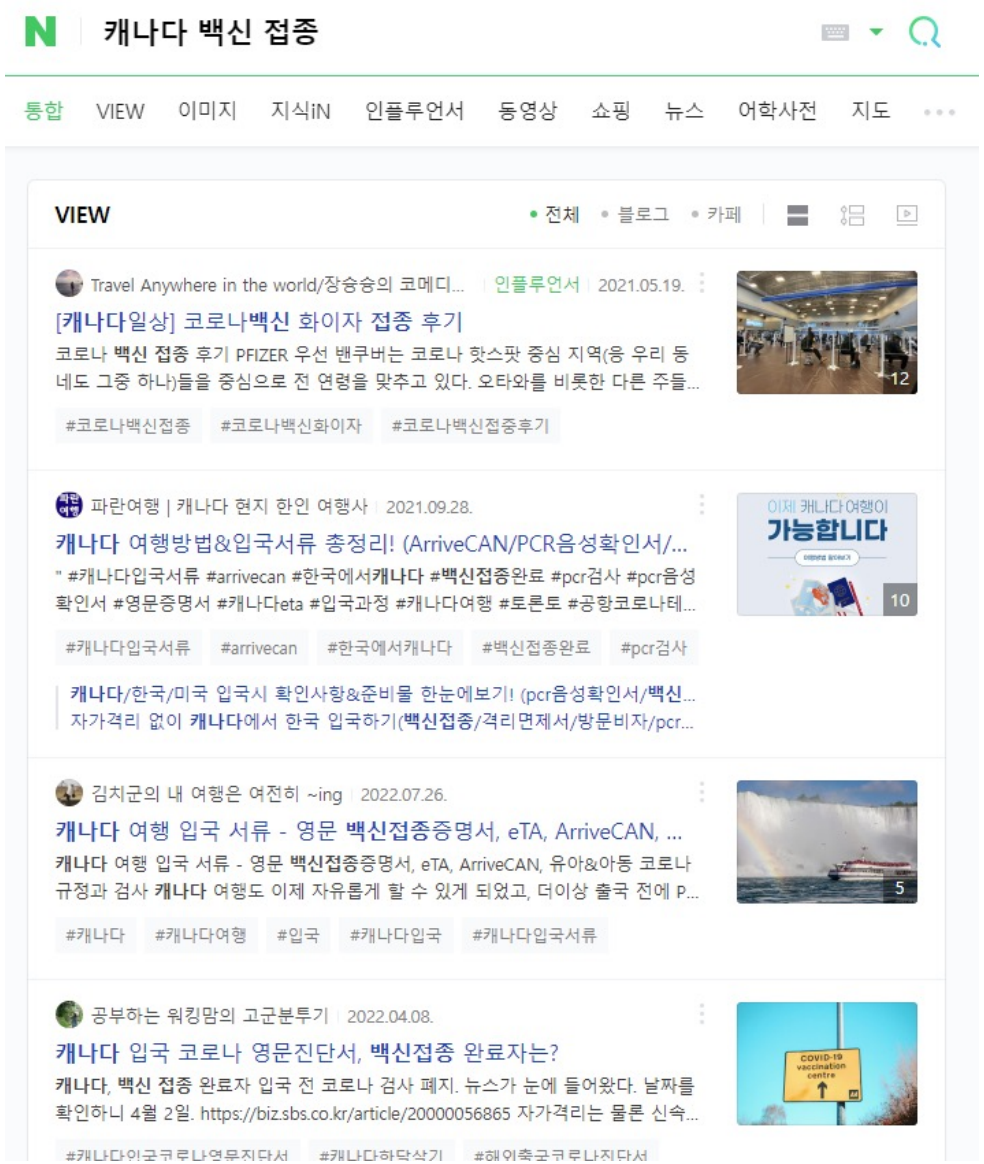
질의

- Searching (or Querying)
- 사용자가 질의(Text, SQL)을 통해 원하는 정보를 얻는 행위
- 질의 결과를 사용자에게 잘 보여주기 위해 스코어링(랭킹 시스템 등)을 사용하기도 함

질의

스코어링 / 랭킹

- 내 질의 내용이랑 제일 유사한 순으로 정렬
- 내가 제일 원하는 정보를 가장 위로



만들기 전에 다시 한번 봅시다

검색 엔진의 3요소는

- 수집 (Crawling)
- 색인 (Indexing)
- 질의 (Searching)

오늘 구현해 볼 내용은

- ~~수집 (Crawling)~~
- **색인 (Indexing) + 전처리 (Pre-processing)**
- **질의 (Searching)**

전처리

색인 전에 전처리(Pre-processing) 과정이 필요합니다

- Cleaning + Normalization (정제 + 정규화)
- Tokenization (토큰화)
- Removing Stopwords (불용어 제거)
- Stemming (어간 추출)
- Lemmatization (표제어 추출)

예제

이해하기 쉽도록 예시 문장을 가져와서 변환 해 보도록 하겠습니다. (영문)

Bitcoin transactions are verified by network nodes through cryptography and recorded in a public distributed ledger called a blockchain.

Cleaning and Normalization

- 토큰화(Tokenization) 하기 전에 데이터를 **정제**하고 **정규화** 하는 작업.

정제(Cleaning)

- 대소문자 통합, 특수문자 등 노이즈 데이터 제거, 짧은 길이의 단어 제거(n=3)

원문) Bitcoin transactions are verified by network nodes through cryptography and recorded in a public distributed ledger called a blockchain.

정제 후) bitcoin transactions verified network nodes through cryptography recorded public distributed ledger called blockchain

Cleaning and Normalization

정규화(Normalization)

- 표현 방법이 다른 단어를 하나로 통합시켜 동일하게 만드는 과정.

원문) bitcoin transactions verified network nodes through cryptography recorded public distributed ledger called blockchain

정제 후) bitcoin **transaction** **verify** network **node** through cryptography **record** public **distribute** ledger **call** blockchain

Tokenizing

- 코퍼스(말뭉치, Corpus)에서 Token으로 단위를 바꾸는 과정.
- 의미가 있는 단위로 토큰화 작업을 진행

- 예시) 띄어쓰기를 기준으로 토큰화를 진행

원본) bitcoin transaction verify network node through cryptography record public distribute ledger call blockchain

토큰화 후) List[“bitcoin”, “transaction”, “verify”, “network”, “node”, “through”, “cryptography”, “record”, “public”, “distribute”, “ledger”, “call”, “blockchain”]

Removing Stopwords

- 분석 시 의미가 없는 단어들을 제거하는 과정 (불용어, 사용하지 않는 단어)
- 여기서는 주로 불용어 사전을 이용하여 제거하는 방식

```
>>> from nltk.corpus import stopwords
>>> import nltk
>>>
>>> nltk.download('stopwords')
[nltk_data] Downloading package stopwords to
[nltk_data]     C:₩Users₩austinkim₩AppData₩Roaming₩nltk_data...
[nltk_data]   Unzipping corpora₩stopwords.zip.
True
>>> db = stopwords.words('english')
>>> len(db)
179
>>> tokens = ["bitcoin", "transaction", "verify", "network", "node", "through", "cryptography", "record", "public", "dis
tribute", "ledger", "call", "blockchain"]
>>> tokens = [v for v in tokens if v not in db]
>>> tokens
['bitcoin', 'transaction', 'verify', 'network', 'node', 'cryptography', 'record', 'public', 'distribute', 'ledger', 'cal
l', 'blockchain']
```

Stemming

- 단어에서 어간(단어의 핵심 부분)을 추출하는 작업 (affix를 제거)
- ex) handles, handle, handler, handling -> handl
- Porter Stemming Algorithm: 5개의 단계로 접미사를 제거하는 알고리즘

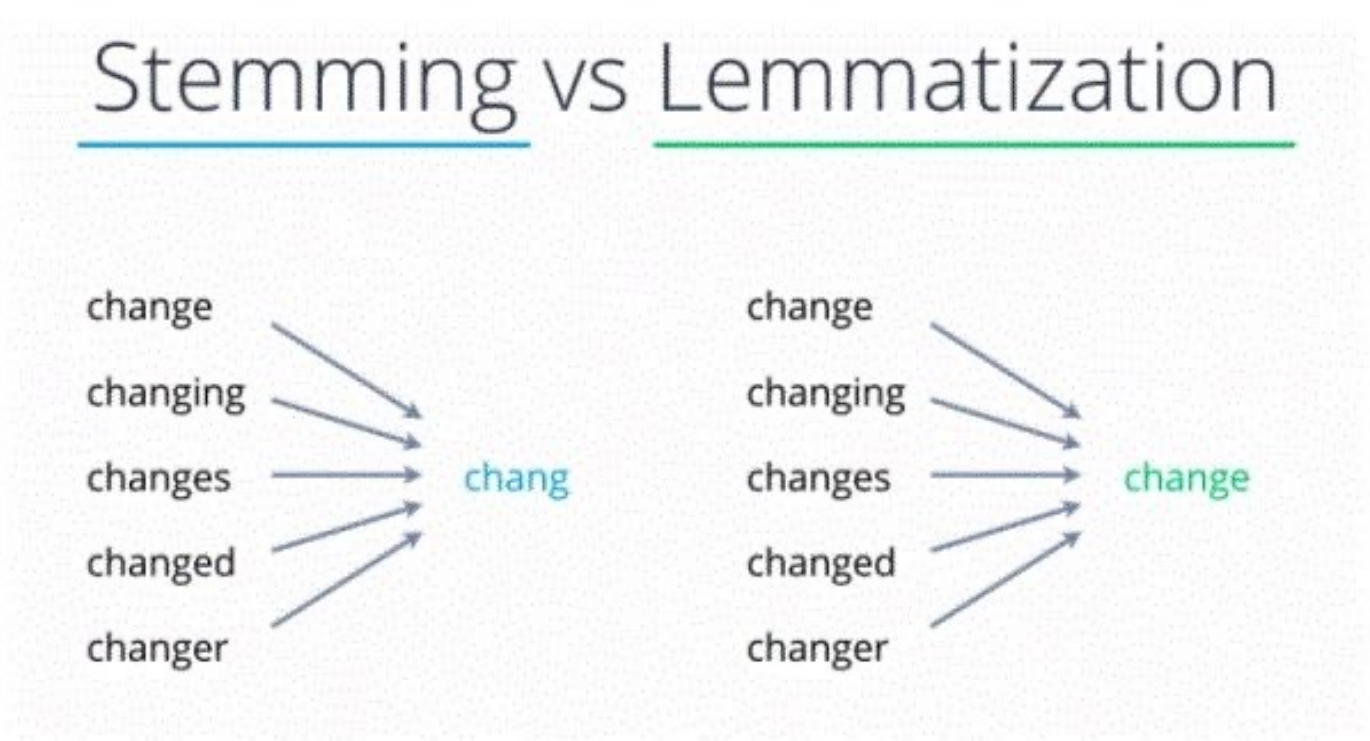
Rule	**Example**
SSES → SS	caresses → caress
IES → I	ponies → poni
SS → SS	caress → caress
S →	cats → cat

Lemmatization

- 단어에서 표제어(사전에서 검색 가능한 단어)를 추출하는 과정
- ex) handles, handle, handler, handling -> handle

Stemming 와 Lemmatization의 차이?

- 둘 다 Corpus 내 단어의 수를 줄이는데 사용

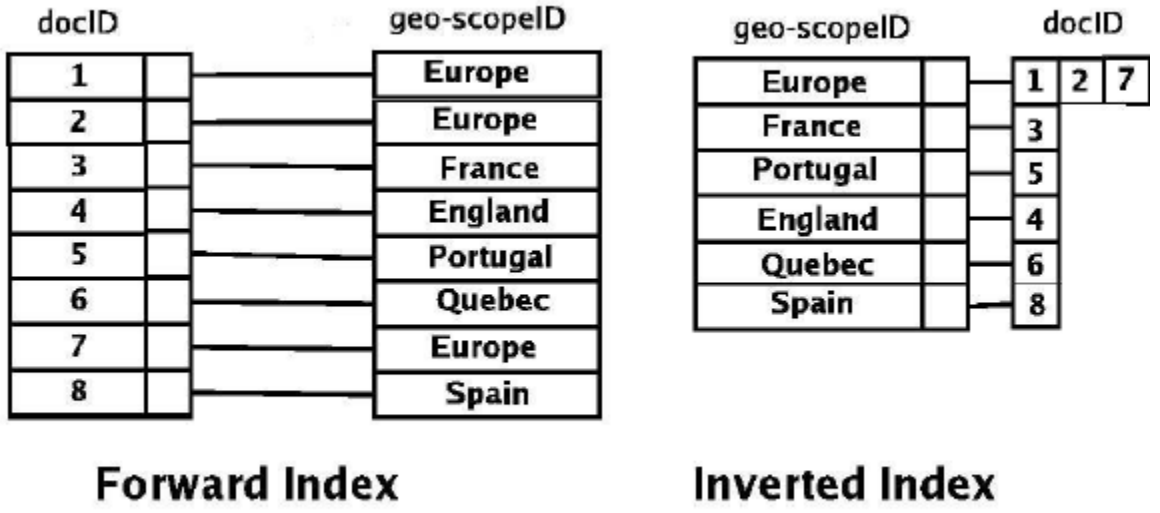


Indexing

- 위에서 고생하여 뽑은 Token들을 색인에 저장하는 과정
- 일반적으로 저장이라고 하지 않고 인덱싱(색인)한다고 한다.
- **Inverted Index**

Inverted Index

- 역인덱싱(Inverted Index)
- 우리가 일반적으로 RDBMS에서 사용했던 방식은 Forward Index라고 부릅니다.



차이를 구분하는 법

- Forward Index는 책 앞에 있는 차례 페이지
- Inverted Index는 책 맨 뒤에 있는 색인 페이지

그래서 그게 뭔데요?

- Term(키워드)을 가지고 있는 Document ID를 저장하는 방식.
- 더 많은 저장공간을 차지하지만, 데이터가 늘어나도 빠른 속도로 검색 가능

이 방식을 왜 사용하나요?

- 예를 들어 Forward Index Scan일 경우 (query = “코로나”)

Doc ID	Text	Result
1	비트코인, 금리인상 공포에 털석... 2만 1천달러대 턱걸이	일치 텍스트 없음
2	4.4조원 비트코인 곧 매물로.. 시장은 초긴장	일치 텍스트 없음
3	방학 끝나면 학교 현장 코로나 확산 우려	일치 텍스트 있음
4	경기도 코로나19 신규확진 3만1339명, 사망 19명	일치 텍스트 있음
5	브렉시트의 저주인가? 영국 소비자 물가 상승률 10% 돌파	일치 텍스트 없음

이 방식을 왜 사용하나요?

- 그러나 Inverted Index 방식일 경우 (query = “코로나”)

Term	**Document ID**
비트코인	1, 2
금리인상	1
방학	3
학교	4
코로나	3, 4
신규확진	4
브렉시트	5
영국	5
소비자	5

질의

- 인덱싱을 했으니, 질의를 해야 한다.
- 근데 뭐가 가장 정확한 검색 결과인가요?
- Document 중 내가 가장 찾고 싶은 결과를 보고 싶어요.

Term	Document ID
비트코인	1, 2
금리인상	1
방학	3
학교	4
코로나	3, 4
신규확진	4
브렉시트	5
영국	5
소비자	5

TF-IDF

- 문서의 집합에서 특정 단어가 얼마나 중요한지 알려주는 수치적으로 나타내는 방법

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = log\frac{N}{1+df}$$

$$TF-IDF(t,d) = TF(t,d) * IDF(t)$$

이렇게 생겼습니다.

공식 뜯어보기

- TF (Term Frequency) → tf(d, t)
  특정 문서 d에서의 특정 단어 t의 등장 횟수를 가져옴

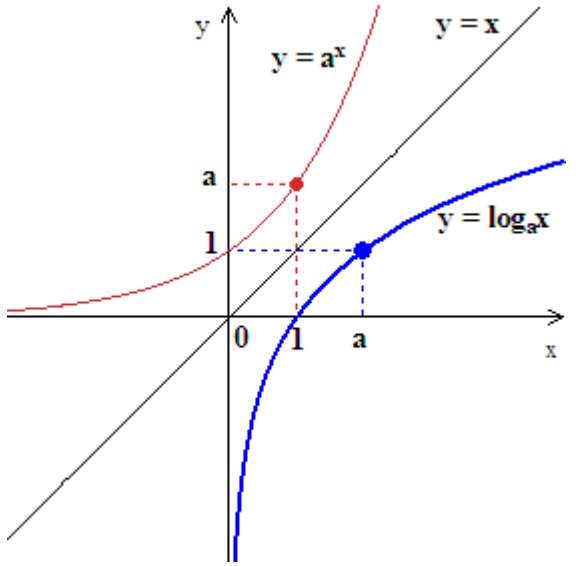
- DF (Document Frequency) → df(t)
  특정 단어 t가 등장한 문서의 수

- IDF (Inverse Document Frequency)
  df(t)의 반비례

$$idf(d,t) = log(\frac{n}{1+df(t)})$$

IDF

- 왜 IDF는 DF의 역수가 아닐까ξ $idf(d,t) = log(\frac{n}{1+df(t)})$
- 단순히 IDF가 DF의 역수를 취해준다면, n이 커질수록(문서의 수가 많아질수록) IDF가 기하급수적으로 커지기에 로그를 취해 이를 방지한다.
- 1을 더해주는 이유는 df(t)가 0일 경우를 방지하기 위해서



TF-IDF

- 다시 돌아와서, TF-IDF는 IDF(문서 전체에서 Term의 중요도) * TF(문서 내의 Term 빈도)다.
- 이걸 이용해서 내가 질의한 Term과 제일 관련도가 높은 Document 대로 가져올 수 있다.
- 근데 이런 완벽해 보이는 TF-IDF에도 문제가 있으니..

TF-IDF의 한계점

- TF-IDF는 단어의 반복성(문서에서 자주 언급)과 대중성(여러 문서에서 사용)이 점수에 영향을 끼침.
- 신조어의 경우 반복성은 충족 하더라도, 대중성은 충족하기 어려움. → 검색이 잘 안됨
- 불용어가 검색 점수에 영향을 줌 (필터링이 되면 좋겠지만 완벽하지 않으므로)
- 문서 길이가 길 경우 점수에 오염을 줄 수 있음 (여러 곳에서 언급되진 않았으나 특정 문서에서만 언급이 많이 된 경우 TF-IDF에서는 점수가 높게 나올 수 있음)

Okapi BM25

- 이를 해결하기 위해 Okapi BM25를 현재 사용 중 (Elasticsearch 등)

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

해설

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

- 이 공식에는 사용자가 설정할 수 있는 변수가 있습니다. (k = 1.2, b = 0.75)
- f(qi, D) → 문서 D에 있는 qi의 빈도
- |D| → D의 길이
- avgdl → 문서의 평균 길이

$$\text{IDF}(q_i) = \ln\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right)$$

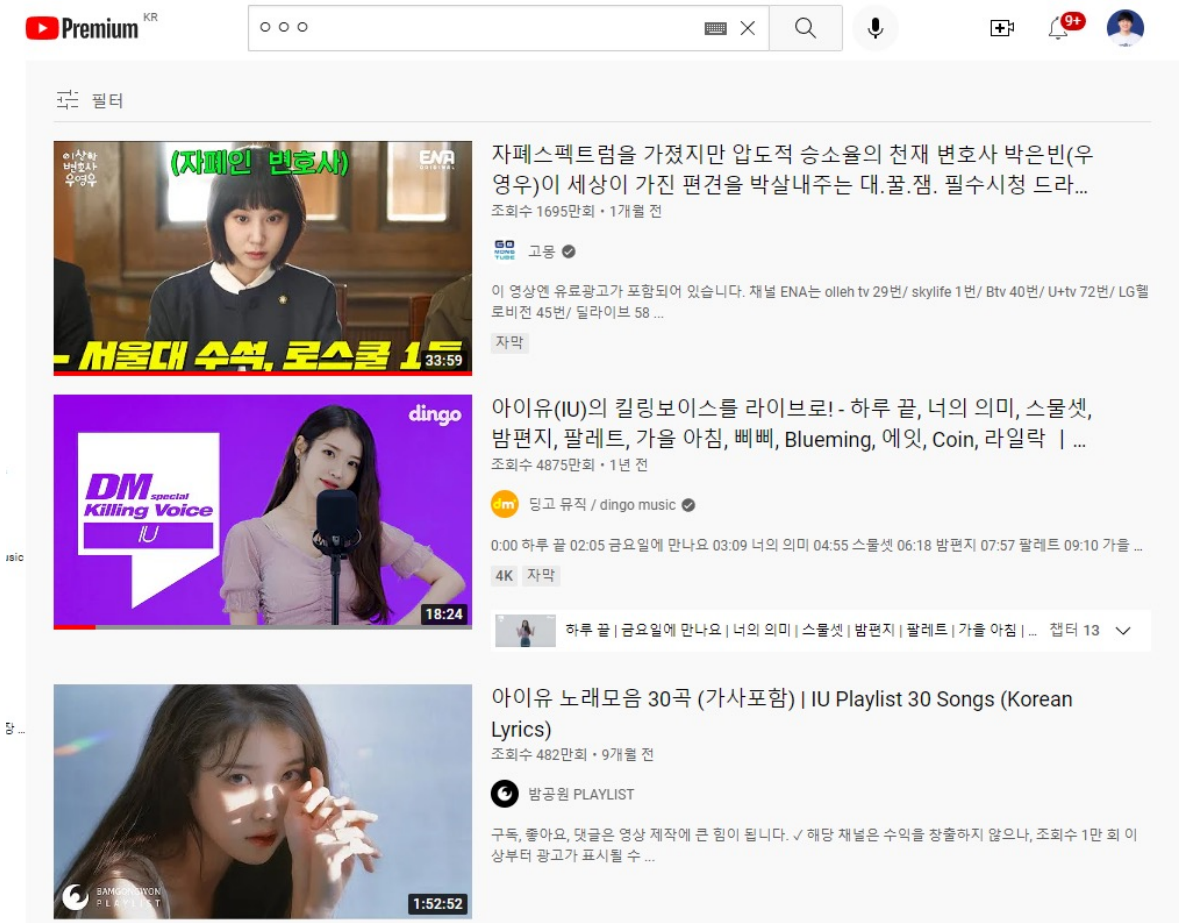
- IDF(qi) → qi에 대한 IDF weight, n(qi) → qi를 포함하는 문서의 수, N → 전체 문서의 수

응용

- 이렇게 만든 검색 엔진을 응용해서 다양한 검색에 사용할 수 있습니다.
- 검색 품질을 늘리기 위한 다양한 기술 등이 있지만, 여기서는 **“한글 초성 검색”** 을 예시로 사용합니다.

초성 검색

- 검색 키워드: “ㅇㅇㅇ”
- 검색 결과: 우영우, 아이유

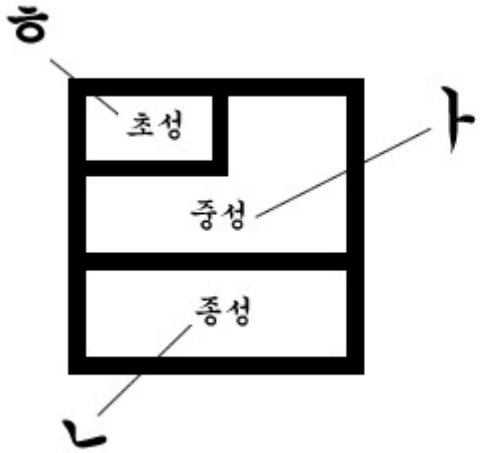


그럼 어느 과정에서 처리해야 할까요?

- Tokenization 후 추출된 초성 값을 Indexing 하면 끝!

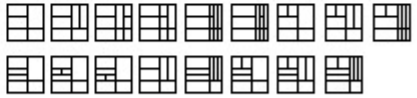
한글 처리 방법

- 한글은 초성, 중성, 종성으로 구성되어 있습니다.



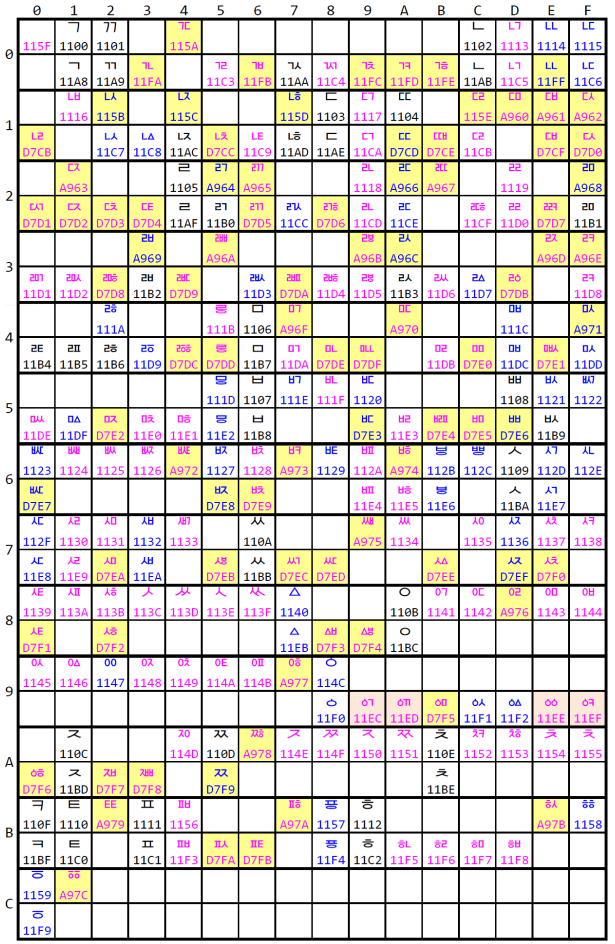
Exam.

Module Form.



초성 추출

- 유니코드 한글 시작 44032
- 유니코드 한글 끝 55199
- 초성 시작 단어 4352
- 한글의 경우 자모 결합 가지수가 각 초성당 588개



공식

[(내가 원하는 글자) – 초성 시작 단어(44032) / 결합 가짓수(588)]

+ 초성 시작 단어(4352)

구현된 것을 보도록 할까요

- https://github.com/bunseokbot/simple-python-search-engine

```
D:\dev\simple_search_engine>python engine.py
Term: militia
A well regulated Militia, being necessary to the security of a free State, the right of the people to keep and bear Arms, shall not be infrin
ged.
TF-IDF Score: 1.2039728043259361
BM25 Score: 1.477104722757996
No person shall be held to answer for a capital, or otherwise infamous crime, unless on a presentment or indictment of a Grand Jury, except i
n cases arising in the land or naval forces, or in the Militia, when in actual service in time of War or public danger; nor shall any person
be subject for the same offence to be twice put in jeopardy of life or limb; nor shall be compelled in any criminal case to be a witness agai
nst himself, nor be deprived of life, liberty, or property, without due process of law; nor shall private property be taken for public use, w
ithout just compensation.
TF-IDF Score: 1.2039728043259361
BM25 Score: 0.8018220859122895


Term: ㄷㅎㅁㄱ
대한민국은 민주공화국이다.
TF-IDF Score: 0.8266785731844679
BM25 Score: 1.1425736584735233
대한민국의 주권은 국민에게 있고, 모든 권력은 국민으로부터 나온다.
TF-IDF Score: 0.8266785731844679
BM25 Score: 0.878147415942302
대한민국의 국민이 되는 요건은 법률로 정한다.
TF-IDF Score: 0.8266785731844679
BM25 Score: 0.9987417682230217
대한민국의 영토는 한반도와 그 부속도서로 한다.
TF-IDF Score: 0.8266785731844679
BM25 Score: 0.987441476353444
대한민국은 통일을 지향하며, 자유민주적 기본질서에 입각한 평화적 통일정책을 수립하고 이를 추진한다.
TF-IDF Score: 0.8266785731844679
BM25 Score: 0.7434874087174042
대한민국은 국제평화의 유지에 노력하고 침략적 전쟁을 부인한다.
TF-IDF Score: 0.8266785731844679
BM25 Score: 0.905480774519078
```

한장으로 정리해 봐요

- 검색 엔진의 3요소는 수집(크롤링), 색인(인덱싱), 그리고 질의(쿼리)이다.
- 색인 과정에서 전처리와 Tokenizing 을 포함한 여러 단계를 거쳐 색인된다.
- 색인 된 이후에는 질의를 통해 데이터를 검색할 수 있다.
- 내 질의 내용에 대해 우선순위별로 스코어링 할 수 있는 TF-IDF, Okapi BM25가 있다.
- 실제로 써야 한다면 만들지 말고 사드세요..

End of Document

@austinkim