정보검색(202201) 과제

컴퓨터학부 2017112351 김태원

1. 전체 프로그램 구성

---

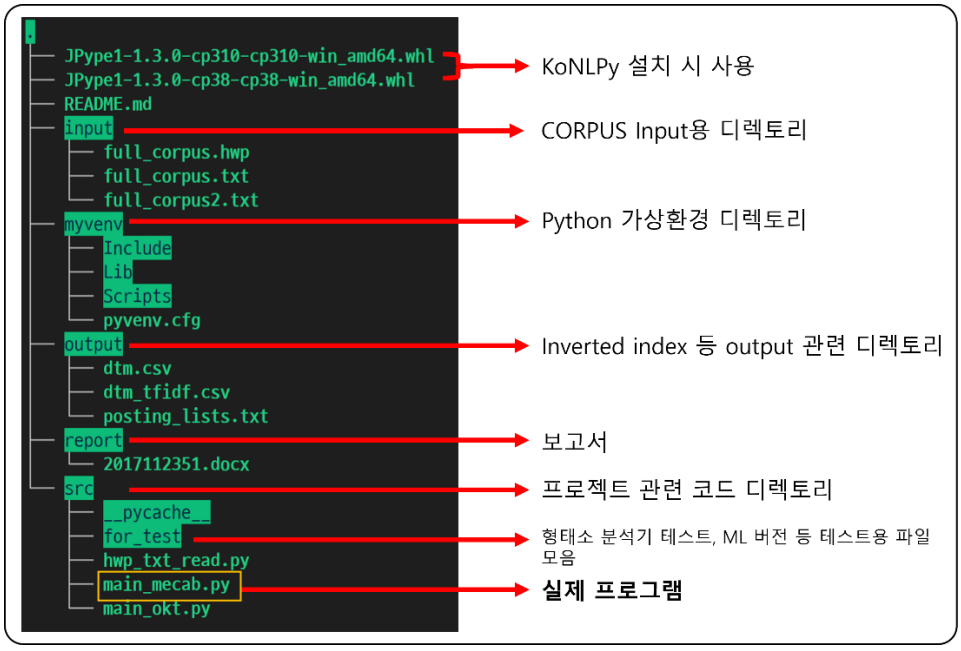
1) Github 링크

https://github.com/Ktaewon/IR_Project1

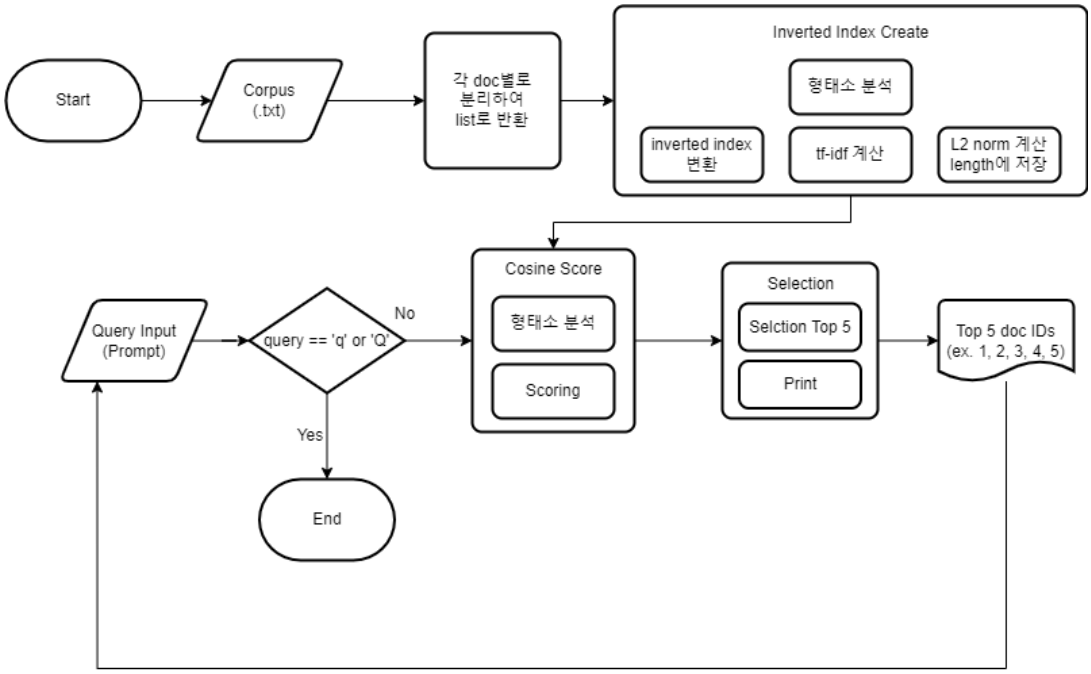
2) 환경

- OS : Window 11
- Langauge : python 3.8.10
- 실행 환경 : python 가상환경(venv)
- Tokenizer : KoNLPy 0.6.0 - mecab, okt
    - JPype1-py3
        - JPype1-1.3.0-cp38-cp38-win_amd64

3) 디렉토리 구조



4) 프로그램 흐름



5) 프로그램 실행법

- Window PC에 KoNLPy 라이브러리가 설치 완료되었다는 전제 하에,

```
git clone https://github.com/Ktaewon/IR_Project1.git
cd IR_Project1
```

- input 디렉토리 생성 및 corpus.txt 삽입

```
mkdir input
# input 디렉토리에 full_corpus.txt 넣어주세요
```

- 실행

```
source myvenv/Scripts/activate
python src/main_mecab.py
```

2. hwp파일(Corpus) 처리

---

1) hwp파일 그대로 읽어보기

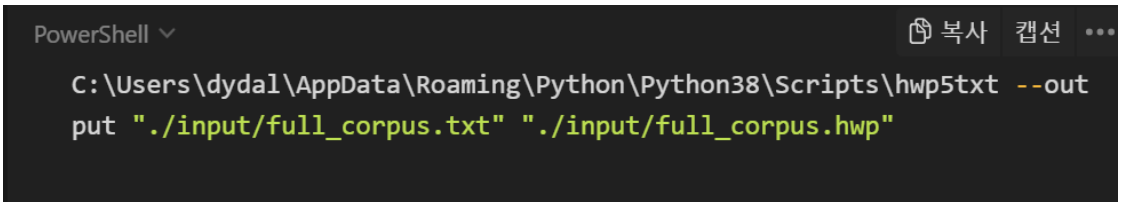
- hwp파일을 python으로 읽기 위해선 `olefile`이라는 패키지가 필요합니다. 이를 이용하여 구현한 함수는 `readHWPandParse(filename:str)` [src/hwp_txt_read.py] 입니다. 먼저 `olefile`의 IO함수를 이용해 파일을 읽은 후 스트림 내부의 내용을 꺼내게 됩니다. 스트림 내부의 내용을 꺼내어 이를 디코딩하여 원하는 문서정보를 얻게 됩니다.
- 하지만 해당 패키지로 주어진 corpus를 읽었을 때, 2페이지 정도 밖에 읽지 못하는 문제가 발생했습니다. 따라서 parsing 작업은 일단 멈추고 문제점을 분석하였습니다. 그러던 중 `'PrevText'`라는 스트림이 문제일 것이라 판단하고, 전체 문서를 담은 stream을 적용해보기로 하여 이를 구현해보았습니다.
- 읽은 데이터를 바탕으로 각 document 별로 잘라 딕셔너리로 만들기 위해서 `readHWPandParseAsDict(filename:str)` [src/hwp_txt_read.py] 을 구현했습니다.
- 하지만 고질적인 문제가 하나 발견했습니다. 아래 예시가 첫 번재 doc 을 읽은 모습인데, 문서의 가장 첫 줄을 읽을 때, 이상한 한문이 같이 읽어진다는 문제였습니다. 여러 가지 찾아보았으나 결국 해결책을 찾지 못했습니다.

```
Python                                                    복사  캡션
[1] : 挹獩    汤捯    1. 지미 카터
지미 카터는 민주당 출신 미국 39번째 대통령이다. 지미 카터는 조지아 주  한 마을에서 태어났다. 조지아 공과대학교를 졸업하였다. 그 후 해 군에 들어가 전함·원자력·잠수함의 승무원으로 일하였다.
```

- 따라서 여러가지 복잡한 과정을 거쳐야 했고, 속도 면에서도 불리하였기 때문에, hwp 파일을 그대로 읽는 것이 아닌 txt 파일로 변환 후 읽는 것으로 방향을 선회하였습니다.

2) hwp → txt

- txt 로의 변환은 여러가지 방법이 있었습니다. 그 중 `pyhwp` 라는 오픈소스 라이브러리를 활용하여 진행하였습니다. 사용법은 매우 간단했습니다. `pyhwp` 설치 후 아래 명령어를 입력하기만 하면 txt 파일이 생성됩니다.
    - 다만 git bash 로 하면 정상적으로 되지 않았고, 저의 경우엔 CMD 나 Powershell 에서 제대로 작업이 되었습니다.



3) txt 파일 읽고 dictionary 로 처리

- txt 파일을 읽어서 각 document 별로 잘라 딕셔너리로 만들기 위해서 `readTXTandParseAsDict(filename:str)` [src/hwp_txt_read.py]을 구현했습니다. 처음에는 XML 문서 형태인 줄 알고, xml parsing 을 했지만 제대로 되지 않아 형식이 다르다는 점을 발견했습니다. 따라서 `<title>`과 `</title>`을 기준으로 파싱을 진행했습니다. corpus 중간에 `</title>`이 제대로 적어지지 않은 문서가 있었지만, 코드 상에서 문제없도록 구현하였습니다. `{ 문서번호 : 문서, …}` 와 같은 형태로 변환하도록 구현했습니다.
- 하지만, dictionary 의 경우 list 에 비해 저장공간 측면에서 조금 불리할 뿐더러, document 번호가 순서 대로였기 때문에, list 를 이용하기로 결정했습니다.

4) txt 파일 읽고 list 로 처리

- document 는 1 번부터 시작하기 때문에, 리스트의 index 와 맞추기 위해서 0 번째 entry 는 빈 문자열을 넣어 처리했습니다.
- 이를 구현한 함수는 `readTXTandParseAsList(filename:str)` [src/hwp_txt_read.py] 입니다. `readTXTandParseAsDict` 함수와 같은 원리로 구현했습니다.

3. 형태소 분석

---

1) 형태소 분석 라이브러리 비교

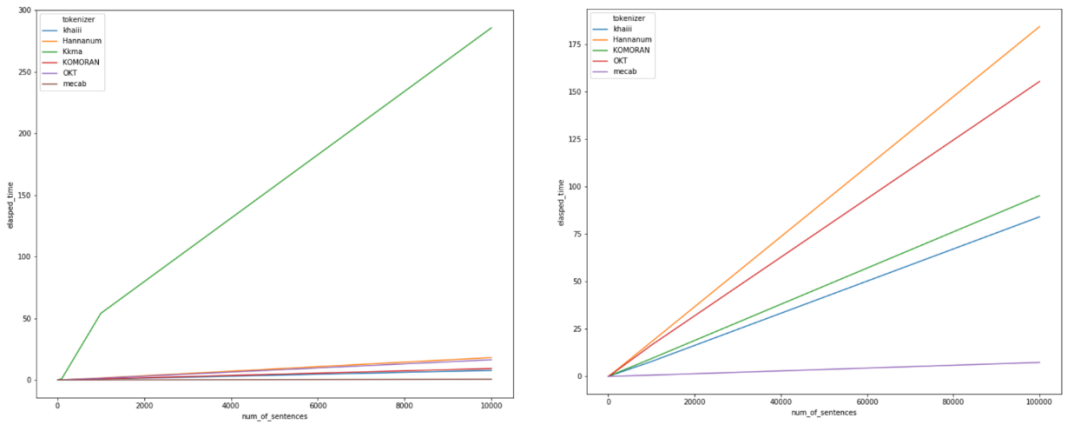
- 형태소 분석을 위해서 우선 오프소스 라이브러리들을 살펴보았습니다. 대표적으로 Mecab, Komoran, Twitter(Okt), khaiii 등이 있었고, 이들의 성능을 비교하기 위해서 직접 코드를 작성해 비교해봤습니다. 이런 라이브러리 설치를 위해서는 KoNLPy 라는 라이브러리에서 통합적으로 불러 사용할 수 있어 사용했습니다.
    - KoNLPy
        - 한국어 형태소 분석기 대부분 지원 (Kkma, Okt, Mecab, Komoran 등등)
        - KoNLPy 의 형태소 분석기는 공통적으로 다음과 같은 메소드를 제공합니다.
            - morphs : 형태소 추출
            - pos: 품사 태깅(Part-of-speech tagging)
            - nouns : 명사 추출
- Mecab 의 경우 윈도우에서는 설치법이 생각보다 복잡하여 설치에 상당한 시간이 들었습니다. 그리고 khaiii 는 윈도우에서는 아예 설치가 불가능하여, 제외했습니다. 따라서 비교한 형태소 분석기는 Mecab, Komoran, Twitter, Okt 인데, Okt 와 Twitter 는 동일한 분석기였습니다.
- example.py 파일에서 각 분석기들을 비교하는 코드를 작성하여 비교했습니다.

2) 형태소 분석기 선택 - Mecab, Okt

- 테스트 후 나온 결과와 한국어 형태소 분석기 성능 비교 (tistory.com) (출처: [Almost Baseball, CSE and Diary:티스토리]) 에서 실제 시간과 정확도를 비교한 것을 바탕으로 Mecab 과 Okt 를 우선 선정했습니다.
- 로딩 시간 (출처 : [Almost Baseball, CSE and Diary:티스토리])

분석기	로딩 시간
Komoran	0.9542
Okt	0.0001
Mecab	`0.0004

- 형태소 분석 시간 (출처 : [Almost Baseball, CSE and Diary:티스토리])
    - 왼쪽 그림은 10, 100, 1000, 10000 문장에 대해 시간을 비교한 것이고,
    - 오른쪽 그림은 10 만까지 문장을 늘렸을 때의 시간을 비교한 것입니다.
        - 보라색 Mecab 이 속도 면에서 매우 우세함을 볼 수 있습니다.



- 형태소 분석 속도 면에서는 비교가 불가능할 정도로 Mecab 이 우세할 뿐더러 분석 품질도 상위권이였습니다. 하지만, Okt 의 stemming 의 정규화가 매우 매력적으로 다가왔기 때문에 두 분석기 모두 사용해 구현해보고 더 적당한 분석기를 선택하기로 했습니다.

4. tf, df, idf, tf-idf 계산

---

1) tf (Term Frequency): 해당 문서에서 특정 단어의 빈도 수

- python 의 `count` 내장함수를 활용해서 input 으로 들어온 리스트에서 해당 term 이 몇 개인지 반환합니다.
- 하지만 실제로 사용하는 것은 여기에 **로그를 취한 값**을 사용합니다.
    - $1 + \log_{10} \mathrm{tf}_{t,d} \quad (\mathrm{tf}_{t,d} > 0)$
    - $0 \quad (otherwise)$

2) df (Document Frequency): 특정 단어가 등장한 문서의 수

- 특정 단어가 등장한 문서의 수를 반환하는 함수로, `count` 라는 변수를 사용해 해당 term 이 해당 document 에 있을 때 1 씩 증가시켰습니다.

3) idf (inverse document frequency): 특정 단어가 등장한 문서의 수를 뒤집은 값 + log()

- df 가 클 수록 가치가 낮게 df 가 작을 수록 가치가 높으므로, 이를 뒤집은 값에 전체 document 개수를 곱하여 사용합니다. 하지만 그 효과가 비례하지 않기 때문에, 그 효과를 줄이기 위해 log 를 붙여 사용합니다. 식은 아래와 같습니다.
- $\mathrm{idf}_t = \log_{10}(\mathrm{N}/\mathrm{df}_t)$

4) tf-idf

01. tf-idf

- 교과서의 정의를 보면, tf 에 로그를 취한 값이 아닌 기본 tf 에 idf 를 곱하여 계산하는 방식으로, 이에 맞게 구현해봤습니다.

02. $\mathbf{W_{t.d} = \left(1 + \log_{10} tf_{t,d}\right) \times \log_{10}(N/df_t)}$

- 교과서의 후반부와 슬라이드와 강의에서 weight 를 계산할 때 위와 같은 식을 이용합니다. 즉, tf 에 로그를 취한 값을 대신 사용하는 것입니다. 제가 작성한 프로그램에서도 위 식을 이용하여 진행했습니다. tf 에 로그를 취한 값을 얻는 함수는 w_tf(t, d) 와 idf(t, D)를 곱하여 tf-idf 를 즉, weight 를 구하게 됩니다.

3) 불용어 제거

- 변환 결과를 확인해 본 결과, "에서", "에", "하다" 와 같은 불용어들이 모두 포함되어 데이터 검색 결과에 영향을 끼칠 수 있다고 판단했습니다. 따라서 `STOPWORDS` 라는 전역변수를 통해 `def delete_stopwords(morphs, stopwords):` 를 inverted index 생성 시 거치게 되면서 불용어를 제거하도록 했습니다.

5. Dictionary, Postings list 생성

- 이번 과제에서는 Inverted Index 를 직접 구현해보기로 했습니다. 이를 위해서는 클래스가 2 가지 필요합니다. 또한 doc 들을 dictionary 와 postings list 로 바꾸어 주는 함수가 필요합니다.

1) Term, Posting 클래스

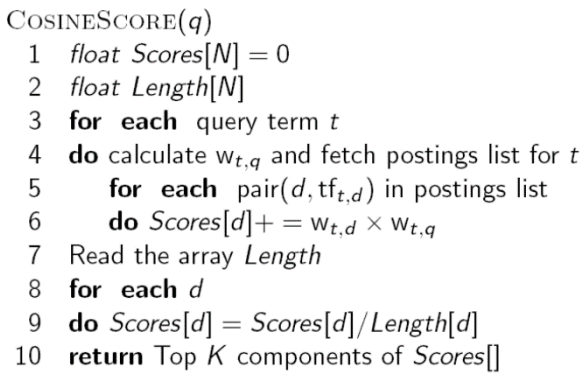
- 딕셔너리에 들어갈 단어 하나 하나를 가지기 위한 클래스인 Term 과 해당 단어에 포함된 document 의 번호와 tf-idf 값 등 정보를 가지는 Posting 클래스를 구현했습니다.

2) doc 리스트 → dictionary, postings lists 변환

- 변환 과정에서 바로 tf-idf 또한 계산하도록 구현했습니다.
- 그리고 쿼리와 score 계산 시에 L2 normalization 적용을 위해 length 라는 리스트 또한 생성합니다.
- 해당 코드는 `def makePostingList(D):` 에 있습니다.

6. Scoring

- 슬라이드와 교과서의 6.14 그림을 참고하여 점수를 부여했습니다. *(옆의 그림 참조)*
- 초반에는 length 배열의 역할이 무엇인지 잘 이해가 가지 않아 헷갈렸지만, L2 정규화를 위한 것임을 깨닫고 Inverted Index 생성 시에 length 배열을 미리 만들어 놨습니다.



1) 쿼리의 tf-idf 계산

- 처음에는 `w_tq = w_tfidf(t, q, [q])` 처럼 쿼리로 입력 받은 term 들에 대해서도 모두 tf-idf 를 계산하여 진행하여 보았습니다. 하지만, `w_tfidf` 를 계산할 때, idf 또한 계산하게 되는데, 쿼리로 입력 받은 문장은 1 개이므로 idf 계산시 $\log_{10}(1/1)$이 되어 0 이 되는 문제가 발생합니다. 이는 모든 term 의 weight 를 0 으로 만들어버리게 되므로, `w_tq = w_tf(t, q)`로 변경하였습니다. 모든 doc 에서 해당 term 이 나타나므로, idf 는 쿼리 텀 계산 시에는 의미가 없어진다고 판단하여 제거했습니다.
- 하지만, 수업에서 배웠듯이 쿼리로 들어오는 문장에서 거의 대부분의 의미 있는 단어들은 중복 되지 않기 때문에, 모두 1 을 부여하는 식으로 바꾸어 보았습니다. 결과 상 큰 차이는 없었고, 계산 효율성을 위해 1 을 부여하는 것으로 결정한 후 진행했습니다.

2) L2 norm

- inverted indexing 때 미리 구한 length 배열을 이 때 이용합니다. 사진에서도 보이는 것 처럼 각 doc 마다 최종 점수를 length 의 L2 norm 으로 나누어 최종 결과값을 만들어 냅니다.
- 하지만 실제로 수업 때 배운 L2 Norm 은 다음과 같습니다.
- $\cos(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\cdot\frac{\vec{d}}{|\vec{d}|} = \frac{\sum^{|V|}_{i=1} q_i d_i}{\sqrt{\sigma^{|V|}_{i=1} q_i^2}\sqrt{\sigma^{|V|}_{i=1} d_i^2}}$
- 하지만 $\sqrt{\sigma^{|V|}_{i=1} q_i^2}$ 이 부분은 항상 같기 때문에 굳이 계산을 더 할 필요는 없다고 생각되어 생략하는 방법을 택했습니다. 즉 doc 에 대해서만 L2 norm 을 적용했습니다.
- 따라서 정리해보면, 최종적으로 적용한 식은 다음과 같습니다.
- $\frac{\sum^{|V|}_{i=1} q_i d_i}{\sqrt{\sigma^{|V|}_{i=1} d_i^2}}$

3) 구현

- 1)과 2)를 토대로 위 그림을 python 코드로 적절히 변형을 거쳐가며 구현한 것은 `def consineScore():`와 같습니다. 4 와 5 의 과정을 거쳐 만들어진 tf-idf 가 포함된 inverted index 를 적절하게 이용하여 score 를 계산하게 됨

니다. 계산된 score 를 L2 norm 을 이용하여 나누어 최종 score 계산되어 반환됩니다.

7. Selection

- document 가 100 개 밖에 되지 않아 초기에는 sorting 을 사용하여 구현했으나, 수업에서 배운 대로 Heap 을 이용한 selection 으로 바꾸기로 결정했습니다. Time Complexity 를 따져봤을 때, Max Heap 을 이용한 selection 이 더 빠르기 때문이기도 했습니다.
- 즉 총 문서 개수가 J 개라고 할 때,
    - Sorting 의 경우, avg: $O(JlogJ)$, worst: $O(J^2)$
    - Max Heap 의 경우, Top K 개를 뽑을 때, $O(J) + O(klogJ)$

  가 소요됩니다. 즉, $O(J) + O(klogJ) < O(JlogJ)$ 이기 때문에, Max Heap 을 최종적으로 사용했습니다.

- 초기에 작성했던 sorting 을 이용한 코드와 Max Heap 을 이용한 코드 모두 작성하였고, sorting 코드는 주석처리 하였습니다. Max Heap 을 만들고 pop 할 때는 Python 의 표준 라이브러리인 `heapq` 를 이용했습니다.

8. 진행하며 발생한 문제점(이슈 & 해결)

---

1) Mecab 사전 우선순위 문제

- Mecab 에서 형태소를 분석할 때, "조지아 주 한 마을"에서 "조지아"를 "조지", "아"로 분석하는 것을 발견했습니다. 하지만 단독으로 "조지아"를 input 으로 줬을 때는 형태소 분석 결과가 "조지아"로 정상적으로 된 것으로 보아, 뒤에 붙는 단어와 자르는 단어와의 우선순위가 문제라고 생각하여, 직접 사용자 사전에 등록 후 우선순위를 높여주는 작업을 진행했습니다.
- 그 결과, 이제 "조지아 주 한 마을"을 형태소 분석 시 정상적으로 "조지아"로 분석하는 것을 발견하고 해당 이슈는 마무리하였습니다.

2) Okt 사전 문제

- Okt 역시 사전에서 위의 "조지아" 이슈처럼 "일제 강점기"를 넣었을 때, "일제", "강점", "기"로 분석하는 현상이 나타났습니다. KoNLPy 에서 직접 사전을 추가하는 방법은 매우 복잡하고, custom 하여 다른 라이브러리를 사용해야 이런 문제를 해결할 수 있었습니다.
- 하지만, Mecab 분석기에서는 이러한 문제가 전혀 발생하지 않음을 발견하고, Okt 를 수정하기 보다 성능이 우수하고 속도 또한 빠른 Mecab 을 주 분석기로 사용하기로 결정했습니다.

9. 결론

---

1) 프로그램 동작

- 실행

```
$ python src/main_mecab.py
검색어를 입력해주세요(종료하려면 q나 Q를 입력해주세요):
```

- 출력

```
검색어를 입력해주세요(종료하려면 q나 Q를 입력해주세요): 대한민국 음악가, 그리고 생물학자
[30, 37, 15, 70, 19]
검색어를 입력해주세요(종료하려면 q나 Q를 입력해주세요): 미국 이론 물리학자는 누구?
[39, 36, 13, 23, 14]
검색어를 입력해주세요(종료하려면 q나 Q를 입력해주세요): Q
프로그램을 종료합니다.
(myvenv)
```

- 수업 시간에 배운 내용을 토대로 하나하나 따라가면서 구현하다 보니 결국 프로그램을 완성했습니다. 모든 과정이 어려웠지만 이렇게 하나의 간단한 검색엔진을 성공적으로 만들 수 있게 되어 매우 좋은 기회였습니다.