Advanced Database Project#1

Information Retrieval System

2018-26161 한지승

## **개발 환경**

* 하드웨어 스펙
  + MacOS 10.14.4
  + Intel Core i7 2.6GHz
  + RAM DDR3 16GB
* Python 3.7.2
* MariaDB 10.3
* 라이브러리
  + NLTLK 3.4
  + Numpy 1.16.2
  + PyMysql 0.9.3

## **구현 기능 소개**

우선 패키지 구조는 다음과 같습니다.

1. python-pagerank
   1. data
      1. doc\_info.sql
      2. inverted\_sql.sql
      3. link.sql
      4. wiki.sql
   2. datasource.py
   3. generator.py
   4. main.py
   5. pagerank.py
   6. search\_engine.py
   7. requirements.txt

**1.1 data**는 테이블 스키마 및 제공된 기본 데이터 sql 파일이 위치해 있으며, 해당 파일을 통해 테이블 스키마와 데이터를 추가할 수 있습니다. **1.2 datasource.py**는 데이터베이스를 접근할 수 있는 유일한 모듈이며, 모든 모듈들을 이를 통해 데이터베이스에 읽고 쓸 수 있습니다. 현재 주어진 데이터로는 모두 딕셔너리로 넣어 계산할 수 있으나, 테스트셋이나 검색엔진으로서 수 많은 데이터를 가정해야한다 판단되어 db cursor를 이용한 generator 방식으로 데이터를 빼와 작업을 할 수 있도록 구성되었습니다. **1.3 generator.py**는 검색엔진을 위한 색인 정보들을 만들어내는 모듈로 inverted index와 pagerank를 생성하는 주된 역할을 합니다. Inverted table은 대소문자를 구분하여 만들어지며, 계산할 때 term의 갯수와 도큐먼트의 전체 term 갯수를 저장합니다. **1.4 main.py**는 기본적인 검색엔진을 시작하는 모듈로 처음 빌드 여부를 묻고 ‘y’를 입력할 경우 index table을 만들며, ‘y’를 제외한 다른 것을 입력할 경우 무시하고 넘어가며, 모두 만들어지는데 제공된 파일 기준 4.5초 가량 소모가 됩니다. 그 후 search engine 모드로 들어가면서 검색할 수 있는 환경이 시작됩니다. ‘/quit’를 입력할 경우 프로그램을 종료합니다. **1.5 page\_rank.py**는 link 테이블의 존재하는 모든 도큐먼트를 대상으로 페이지랭크를 계산하며, power method를 통해 forward link 테이블을 jump probability를 고려하여 만들어지며, forward link가 없은 dangling node는 논문에서 언급한 대로 1 / n (document total size)로 초기화하여 만들어지며, eplsilon보다 작아질 때 까지 score vector를 계산합니다. **1.6 search\_engine**은 만들어진 index table을 기반으로 해당하는 도큐먼트를 불러와 tf-idf를 계산하고, document별로 tf-idf를 더한 뒤 페이지랭크를 곱하는 방식으로 스코어링하여 결과를 상위 10개를 반환합니다. **1.7 requirements.txt**은 필요한 라이브러리를 명시하여 설치가 가능토록 합니다.

## **PageRank 구현 코드 설명**

PageRank 구현 코드는 **1.5 page\_rank.py**에 구현되어 있습니다.

Source code [line number]: page\_rank.py [16-17]

우선 처음에는 링크 테이블을 통해 페이지랭크 계산을 하기 위해 필요한 모든 도큐먼트를 구하는 쿼리 (self.db.get\_all\_link\_docs())를 통해 계산에 필요한 모든 도큐먼트 id를 구해오며, 함께 계산 편의를 위해 도큐먼트 id를 리스트 인덱스로 변환이 가능한 id2idx를 함께 만들어 둡니다.

Source code [line number]: page\_rank.py [20-26]

이후, link 테이블에서 forward link 정보를 모두 불러와 도큐먼트마다 갈 수 있는 도큐먼트 정보를 딕셔너리로 만들어 둡니다. 그리고, link matrix 정보를 만들기 위해 우선 다른 곳으로 넘어갈 확률로 forward link 확률을 초기화하고, score 계산을 위한 벡터도 1 / n (all document size)로 초기화합니다.

Source code [line number]: page\_rank.py [27-34]

모든 도큐먼트를 순회하며 forward link가 있는 경우 (1 – jump probability) \* 1 / C (forward link 갯수)를 더해주면서 도큐먼트 마다 forward link에 대한 probability를 계산해 줍니다. 단, forward link가 없는 경우 즉, dangling node의 경우 처리를 안해 줄 경우 중요한 문서로부터 back link를 가지고 있어도 랭크가 반영되지 않는 것을 보완하기 위해 적용하는 어떤 문서로든 갈 수 있는 값인 1 (1 – jump probability) \* 1 / n (all document size)로 초기화를 해줌으로써 모든 문서로 갈 수 있는 확률이 1 / n이 되도록 더해줍니다.

Source code [line number]: page\_rank.py [38-42]

만들어진 matrix에 score vector를 주어진 epsilon 보다 변화가 적을 때까지 matmul연산을 반복하면서 page rank를 계산합니다.

## **주요 SQL문 설명**

SQL 구현 코드는 **1.2 datasource.py**에 구현되어 있습니다.

모든 SQL 코드들은 fetchall()을 통해 모두 바로 불러오는 것이 아닌 cursor를 줌으로써 많은 데이터양에도 대응되도록 만들었습니다.

Source code [line number]: datasource.py [46-53]

Wiki 테이블의 모든 텍스트 파일을 불러오며 inverted\_index를 만들 때 사용됩니다.

Source code [line number]: datasource.py [55-65]

Link 테이블에 존재하는 모든 도큐먼트 id를 가져옵니다. 이는 페이지랭크 계산을 위해 필요한 도큐먼트 대상을 준비하기 위해 필요합니다. Back link, forward link에 있는 모든 도큐먼트 id를 중복 없이 가져오기 위해 UNION과 DISTINCT를 사용하여 가져옵니다.

Source code [line number]: datasource.py [67-75, 77-85]

Link 테이블에서 도큐먼트 별로 back link나 forward link를 가져오기 위해 사용되는 쿼리입니다. 두 쿼리는 작동 방식이 같으며 back link를 가져올지 forward link를 가져올지 차이만 있습니다. Forward link를 가져오기 위한 쿼리를 기준으로 설명을 하면, 우선 id\_from을 기준으로 group by를 통해 가져온 뒤 해당 id\_from을 가지는 id\_to들을 데이터베이스 빌트인 함수인 GROUP\_CONCAT()을 통해 ‘,’로 split이 가능한 형태의 필드로 가져옵니다. 이후, 사용할 때 ‘,’을 기반으로 split하여 forward link를 계산합니다.

Source code [line number]: datasource.py [87-98, 100-111]

Bulk 업데이트를 하기 위한 구문으로 검색엔진 서비스를 제공하기 전 필요한 정보를 모두 데이터베이스에 넣는 SQL입니다. 빠른 insert를 위해 한번에 집어 넣은 뒤 commit하는 방식으로 구현되어있습니다.

Source code [line number]: datasource.py [113-135]

검색엔진이 검색을 할 때 inverted index에 쿼리를 날릴 때 사용하는 부분입니다. Join의 성능을 위해 우선적으로 inverted index에서 필요한 레코드를 불러온 뒤 wiki테이블과 doc\_info 테이블을 join하는 방식으로 구현되어있습니다. 각 테이블들은 검색 결과를 도출하는데 필요한 title, term freq, doc term size, page rank score에 대한 정보를 가져올 수 있습니다.

## **프로그램 실행 예시**

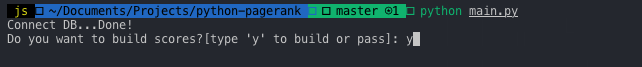


그림 1 프로그램 실행 시

프로그램을 실행할 시, Database 연결 여부를 확인한 뒤, 검색에 필요한 index 정보들을 계산할지 묻는 부분이 있습니다. Index 정보들을 만들기 위해서는 ‘y’를 입력 시 테이블 building을 시작하며, 그 외 문자를 입력하거나 엔터를 칠 경우 무시하고 넘어갑니다.

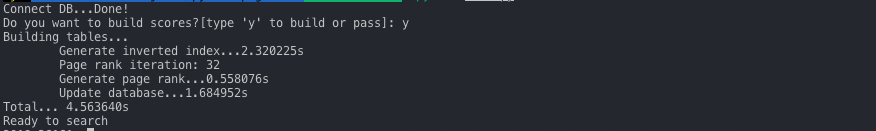


그림 2 테이블 building 시

Building을 할 경우 각 inverted index를 계산하는 시간, page rank를 계산하는 시간과 만들어진 정보를 데이터베이스에 넣는 시간을 모두 찍어내며, 총 걸린 시간을 제공해 줍니다. 주로 page rank를 계산하는 시간이 많이 걸리며 page rank 계산 대상이 wiki가 아닌 link 테이블 전체 도큐먼트를 계산하기 때문에 다른 phrase에 비해 시간이 걸립니다.

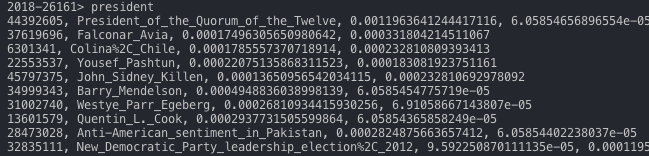


그림 3 단일 키워드 검색 결과 예시

이 후, president에 대한 상위 10개의 검색 결과를 보여주며 차례대로 document id, title, tf-idf score, page rank score를 출력하며, tf-idf score \* page rank score 기반으로 정렬하여 출력합니다.

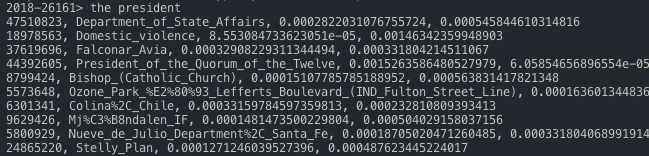


그림 4 멀티 키워드 검색 결과 예시

멀티 키워드 검색에 대한 것도 검색이 가능하며, 검색 결과 형식은 모두 같습니다.

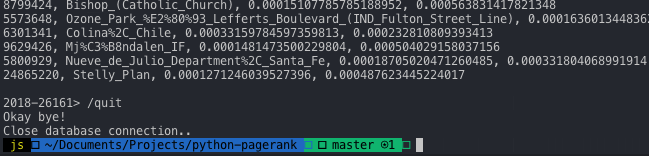


그림 5 종료 예시

프로그램을 종료하기 위해서는 유일한 예약어인 ‘/quit’를 입력하여 빠져 나올 수 있으며, Ctrl-C를 이용하여 빠져나오는 것도 가능합니다.

## **평가 및 결론**

처음 구현하는 데 있어서 어려움을 느꼈던 부분은 간단한 수식으로 나타낼 수 있었던 page rank 계산 방법이 어떤 특정 케이스에서는 어떻게 해야하는지에 대한 부분이 명확하지가 않아 논문이나 서베이를 통해 추가적인 조사가 필요했었습니다. 뿐만 아니라 wiki 테이블에 존재하는 도큐먼트에 대한 검색으로 구현으로 진행 중이였다가 link 테이블에 존재하는 주어지지 않은 document에 대한 페이지랭크를 반영하는 것이 검색엔진으로서 올바르다 판단되어 link 테이블에 존재하는 도큐먼트를 대상으로 page rank를 계산 하였으며, 검색 키워드 토크나이즈는 명시된 대로 word\_tokenize로 명시되어있기 때문에 lemmatize나 normalize, stopword처리는 하지 않았음을 말씀드립니다.

위와 같은 이유에서 invert table을 만드는데 약 2.2초 정도에 적당한 수준의 시간이 걸리고 있는 반면, link table에 존재하는 모든 도큐먼트를 대상으로 page rank를 dangling node까지 고려하여 계산하는데 약 0.5초정도의 시간이 걸리는 것을 확인할 수 있었습니다. Page rank의 경우 도큐먼트의 대상, dangling node를 고려 여부에 따라 속도가 크게 차이가 나는 것 또한 확인하였습니다. Link table에 존재하는 도큐먼트의 수가 약 6300개정도 되는 것으로 확인이 되었는데 생각보다 적은 문서에도 비교적 오랜 시간이 걸린다는 점을 알고 검색엔진을 만드는 구글, 네이버와 같은 기업이 페이지랭크 기반으로 얼마나 많은 연구와 최적화가 있었을지 어느정도 알 수 있는 프로젝트였습니다. 또한, 최종적으로 제시한 테이블 스키마가 되기 위해 여러 시행착오들로부터 나이브하게 정보를 가져와 계산하는 방식에서 TF-IDF나 page rank를 계산하는데 필요한 정보들을 미리 계산하고 저장 해놓는 방식까지 발전해가면서 검색엔진을 만드는데 있어 필요한 최소한의 정보들이 무엇인지 생각해보는 시간이었습니다.

결론적으로, building을 하는데 걸리는 시간은 약 4.5초정도로 준수한 시간이 걸린다고 생각되며, building 하는데 걸리는 시간을 조금 더 두고 검색 쿼리를 찾는데 있어 바로바로 반응하는 것이 합당하다 생각됩니다. 최종적으로 검색 쿼리 응답시간이 매우 빠르게 작동하기 때문에 훌륭한 검색엔진이라 생각됩니다.

감사합니다.