HW2

필요 모듈 정의

In [34]:

```
from collections import defaultdict # 데이터를 담을 dictionary를 만들어줄 모듈
import math                         # 수학적 함수 사용을 위한 모듈
import sys                          # 로컬 file을 읽어올 함수
from functools import reduce        # 재귀적 함수 기능을 구현해주는 모듈
import re                           # 데이터 정규화 작업을 위하여 사용한 모듈
```

필요 변수 정의

In [35]:

```
# 사용할 document 리스트
document_filenames = {0 : "data/Beauty and the Beast.txt",
                      1 : "data/Get Out.txt",
                      2 : "data/Big Sick, The.txt",
                      3 : "data/Guardians of the Galaxy Vol 2.txt",
                      4 : "data/La La Land.txt",
                      5 : "data/Logan.txt",
                      6 : "data/War for the Planet of the Apes.txt",
                      7 : "data/It.txt",
                      8 : "data/Thor Ragnarok Script.txt",
                      9 : "data/COCO.txt"}

prog = re.compile(r'^[a-zA-Z0-9]+$') # 정규화 문서를 term으로 구분할때 숫자와 문자만 있는 term을 사용하기 위하여 선언
removal = 'a,able,about,across,after,all,almost,also,am,among,an,and,any,are,as,at,be,because,been,but,by,can,cannot,could,dear,did,do,does,either,else,ever,every,for,from,get,got,had,has,have,he,her,hers,him,his,how,however,i,if,in,into,is,it,its,just,least,let,like,likely,may,me,might,most,must,my,neither,no,nor,not,of,off,often,on,only,or,other,our,own,rather,said,say,says,she,should,since,so,some,than,that,the,their,them,then,there,these,they,this,tis,to,too,twas,us,wants,was,we,were,what,when,where,which,while,who,whom,why,will,with,would,yet,you,your'.split(",")
# 제외할 단어 리스트

N = len(document_filenames) # document 개수

dictionary = set() # 전체 term을 담을 변수
terms = defaultdict(dict) #각 term별로 문서 id와 토큰 개수를 저장하는 변수
document_frequency = defaultdict(int) # term이 몇 개의 문서에서 보이는지 저장하는 변수
length = defaultdict(float) # 각 문서 vector의 유클리드 거리를 담을 변수
```

- initialize_terms()
  - 각 문서마다 tokenize를 진행한다. 각 문서마다 terms를 가져오고 중복을 제거한 unique한 terms를 만들어 dictionary 함수에 넣는다.
  - 그리고 term 마다 문서 당 빈도수를 넣는다. (weight를 구하기 위해 저장)

- tokenize()
  - 문서안에 term들을 뽑아 낸다. removal에 해당하는 term과 특수문자가 들어간 term을 제외하고 terms를 구성하여 반환한다.

In [36]:

```
def initialize_terms():
    # 각 문서마다 tokenize하고 dictionary 구성한뒤 posting에 term의 개수와 저장
    global dictionary, terms
    for id in document_filenames:
        f = open(document_filenames[id],'r', encoding="utf-8")
        document = f.read()
        f.close()
        terms_per_doc = tokenize(document)
        unique_terms = set(terms_per_doc)
        dictionary = dictionary.union(unique_terms)
        for term in unique_terms:
            terms[term][id] = terms_per_doc.count(term) #term 빈도

def tokenize(document):
    # token으로 만드는 작업
    _terms = document.lower().split()
    return [ term for term in _terms  if term not in removal and prog.match(term
) ]
```

- initialize_document_frequencies()
  - 해당 term이 몇 개의 문서에서 보이는지 그 빈도수를 저장한다. (idf를 구하기 위하여 저장)

In [37]:

```
def initialize_document_frequencies():
    # 문서당 term의 빈도수
    global document_frequency
    for term in dictionary:
        document_frequency[term] = len(terms[term])
```

- weight()
  - term의 weight(tf * idf)를 구하여 반환해준다.

- inverse_document_frequency()
  - term의 idf를 구하여 반환해준다.

In [38]:

```
def weight(term,id):
    # tf.idf 구한다.
    if id in terms[term]:
        return terms[term][id]*inverse_document_frequency(term)
    else:
        return 0.0

def inverse_document_frequency(term):
    #idf 구한다.
    if term in dictionary:
        return math.log(N/document_frequency[term],2)
    else:
        return 0.0
```

- initialize_lengths()
  - 문서 terms의 weight의 유클리드 거리를 구한다. similarity를 구하기 위하여 미리 문서마다 length를 구하여 저장해 놓는다.

- similarity()
  - 해당 query와 문서 사이의 cosine similarity를 구하게 되는데 쿼리 vector의 거리를 제외시키고 미리 문서마다 구해놓은 length만을 이용한다.
  - query vector의 tf * idf의 값과 document의 weight를 곱하여 length를 나누어 cosine similarity를 구한다.

In [39]:

```
def initialize_lengths():
    # 각 문서의 유클리드 거리를 구한다.
    global length
    for id in document_filenames:
        l = 0
        for term in dictionary:
            l += weight(term,id)**2
        length[id] = math.sqrt(l)

def similarity(query,id):
    # query and id에 해당하는 document 사이의 cosine similarity을 구한다.
    # 구한 유클리드 거리를 나눌 때 쿼리의 유클리드 거리는 사실상 랭크의 순위를 구하는데 영향을 주지 않는다. 그래서 document 거리만 나눈다. query vector와 document의 weight를 곱하여 similarity에 누적시킨 후 그 document 유클리드 거리로 나눈다.
    similarity = 0.0
    for term in query:
        if term in dictionary:
            similarity += inverse_document_frequency(term)*weight(term,id)
    similarity = similarity / length[id]
    return similarity
```

- intersection()
  - 받은 query가 포함된 문서들의 교집합을 구한다. reduce 함수를 이용하여 교집합 연산을 재귀적으로 하고 반환한다.

- search()
  - 사용자에게 받은 query을 terms으로 만들고 이를 문서와의 similarity를 구하여 ranking을 도출한다. 문서들의 rank는 'decreasing order of cosine of angle'에 따라 정하며 ranking 순으로 출력한다.

In [40]:

```
def intersection(sets):
    #주어진 집합리스트 중에서 교집합을 구하기 위하여 만든 함수.
    return reduce(set.intersection, [s for s in sets])


def search():
    query = tokenize(input("찾을 쿼리 입력 : "))
    if query == []:
        sys.exit()

    # 토큰화 된 query 와 교집합을 이루는 doc id를 가져온다.
    relevant_document = intersection(
            [set(terms[term].keys()) for term in query])

    # 아무런 문서가 없다면..
    if not relevant_document:
        print("매치되는 doc이 없습니다.")
    else:
        # 보낸 query와 문서의 similarity를 구하여 ranking을 도출한다.
        scores = sorted([(id,similarity(query,id))
                         for id in relevant_document],
                        key=lambda x: x[1],
                        reverse=True)
        print("ranking")
        for (id,score) in scores:
            print(str(score)+": "+document_filenames[id])
```

In [ ]:

```
def main():
    initialize_terms()
    initialize_document_frequencies()
    initialize_lengths()
    while True:
        search()

if __name__ == "__main__":
    main()
```

```
찾을 쿼리 입력 : rocket
ranking
0.13507772544858532: data/Guardians of the Galaxy Vol 2.txt
0.05928239766596676: data/War for the Planet of the Apes.txt
0.0016157218375811 82: data/It.txt
0.0006190252631741595: data/Beauty and the Beast.txt
0.0003702308633817864: data/Thor Ragnarok Script.txt
찾을 쿼리 입력 : thor
ranking
2.941613239716275: data/Thor Ragnarok Script.txt
찾을 쿼리 입력 : count
ranking
0.002423582756371773: data/It.txt
0.0009661124730461199: data/La La Land.txt
0.0008055038967432544: data/COCO.txt
0.0005014712844385952: data/Logan.txt
0.0004966092847374459: data/Guardians of the Galaxy Vol 2.txt
찾을 쿼리 입력 : rocket thor count
매치되는 doc이 없습니다.
찾을 쿼리 입력 : rocket thor
ranking
2.941983470579657: data/Thor Ragnarok Script.txt
```

결과(해당 문서의 포함되어있는 query의 수)

- rocket
  - "data/Beauty and the Beast.txt" : 1
  - "data/Get Out.txt" : 0
  - "data/Big Sick, The.txt" : 0
  - "data/Guardians of the Galaxy Vol 2.txt" : 276
  - "data/La La Land.txt" : 0
  - "data/Logan.txt" : 0
  - "data/War for the Planet of the Apes.txt" : 140
  - "data/It.txt" : 2
  - "data/Thor Ragnarok Script.txt" : 1
  - "data/COCO.txt" : 0

- thor
  - "data/Beauty and the Beast.txt" : 0
  - "data/Get Out.txt" : 0
  - "data/Big Sick, The.txt" : 0
  - "data/Guardians of the Galaxy Vol 2.txt" : 0
  - "data/La La Land.txt" : 0
  - "data/Logan.txt" : 0
  - "data/War for the Planet of the Apes.txt" : 0
  - "data/It.txt" : 0
  - "data/Thor Ragnarok Script.txt" : 720
  - "data/COCO.txt" : 0

- count
  - "data/Beauty and the Beast.txt" : 0
  - "data/Get Out.txt" : 0
  - "data/Big Sick, The.txt" : 0
  - "data/Guardians of the Galaxy Vol 2.txt" : 1
  - "data/La La Land.txt" : 2
  - "data/Logan.txt" : 1
  - "data/War for the Planet of the Apes.txt" : 0
  - "data/It.txt" : 3
  - "data/Thor Ragnarok Script.txt" : 0
  - "data/COCO.txt" : 2

- rocket thor
  - "data/Thor Ragnarok Script.txt"

문제점

- query를 3개의 term으로 만든다고 가정한다면 (예 : query : rocket thor count)
- 3가지의 term을 모두 가진 doc을 intersection으로 찾고 이 문서들 가운데에서 rank을 만든다.
- 즉, rocket thor는 "결과" 처럼 "data/Thor Ragnarok Script.txt" 문서가 만족한다.
- 하지만 count는 "data/Thor Ragnarok Script.txt" 이 문서에 없기 때문에
- "data/Thor Ragnarok Script.txt"이 문서는 rocket thor count과 가장 유사할 수 있지만 제외된다.
- 그래서 query를 입력 받을 때 intersection의 제외 조건을 바꿀 필요성이 있다.
- 개선의 방향으로는 구글, 네이버에서 검색을 하게 되면 제외되는 query가 생긴다. 예 : Missing: "-----" 처럼 말이다.
- 더 높은 가치의 rank를 보이기 위하여 제외시킬 query를 알려준다면 좋은 결과를 도출 할 수 있을 것 같다.