1.6.7 *미니 검색 엔진*

이제, 일종의 검색 엔진으로 동작하는 프로그램을 작성해 볼 것이다.

"문서(documents)"들로 이루어진 파일이 하나 주어지고, 각 문서는 이 파일의 한 라인을 차지한다. 이때, 주어진 어떤 단어를 포함하는 문서들을 찾아내는 자료구조(*역 인덱스*라고 함)를 만들어 보자. 문서들은 문서 번호에 의해 식별된다. 파일 내의 첫 번째 라인에 의해 표현되는 문서는 0번, 두 번째 라인에 의해 표현되는 문서는 1번, 등등이다.

문자열에 대해 정의된 메서드인 `split()`을 사용할 수 있다. 메서드 `split()`은 문자열을 공백으로 나누어 서브 문자열(substring)로 분리하고 서브 문자열들의 리스트를 반환한다.

```
>>> mystr = 'Ask not what you can do for your country.'
>>> mystr.split()
['Ask', 'not', 'what', 'you', 'can', 'do', 'for', 'your', 'country.']
```

여기서, 마침표(period)는 서브 문자열의 일부로 간주한다. Lab을 좀 더 쉽게 하기 위해 문서 파일들이 제공되는데, 여기서 구두점(punctuation)은 공백을 사용해 단어들과 분리되어 있다.

흔히 리스트 원소들의 인덱스를 추적하면서 원소들 각각에 대해 이터레이션을 하고자 한다. 이를 위해 파이썬은 `enumerate(L)`를 제공한다.

```
>>> list(enumerate(['A','B','C']))
[(0, 'A'), (1, 'B'), (2, 'C')]
>>> [i*x for (i,x) in enumerate([10,20,30,40,50])]
[0, 20, 60, 120, 200]
>>> [i*s for (i,s) in enumerate(['A','B','C','D','E'])]
['', 'B', 'CC', 'DDD', 'EEEE']
```

Task 1.6.6: 주어진 문자열(문서)들의 리스트에 대해 딕셔너리를 리턴하는 프로시저, `makeInverseIndex(strlist)`을 작성해 보자. 이때, 반환되는 딕셔너리는 각 단어를 그 단어가 나타나는 문서들의 문서 번호로 이루어진 집합으로 매핑한다. 이러한 딕셔너리를 역 인덱스 자료구조라 한다. (힌트: `enumerate`를 사용)

Task 1.6.7: 역 인덱스와 질의 단어들의 리스트를 받아 들여 질의에 들어 있는 임의의 단어를 포함하는 모든 문서들을 지정하는 문서 번호들의 집합을 리턴하는 프로시저 `orSearch(inverseIndex, query)`를 작성해 보자.

Task 1.6.8: 역 인덱스와 질의 단어들의 리스트를 받아 들여 질의에 들어 있는 모든 단어들을 포함하는 모든 문서들을 지정하는 문서 번호들의 집합을 리턴하는 프로시저 `andSearch(inverseIndex, query)`를 작성해 보자.

작성한 프로시저를 다음 두 개의 파일에 적용해 보자.

- `stories_small.txt`
- `stories_big.txt`