Internet Application

<HW #2>

산업공학과 황영석

2010-12086

1 번 문제 코드 및 설명(주석)

1.1 번 : 따옴표 제거

```
@Override
protected String tokenSimpleFix(String token) {
    char[] chars = token.toCharArray();
    int j = 0;
    //1번과제
    // TODO: 적절한 code 입력
    for (int i = 0; i < chars.length; i++) {
        char c = chars[i];
        boolean isApostrophe = (c=='\'');
        boolean isApostropheOnTheFirst = (chars[0]=='\'');

        // TODO: 적절한 code 입력
        if(isApostrophe){ //만약 임의의 따옴표를 발견했다면
            if(isApostropheOnTheFirst){ //따옴표가 토큰 첫머리에 있는 따옴표냐?
                //그럼 Do nothing
            }else{//안 그러면 중간머리에 있는 따옴표고
                break; //그 뒤는 다 없애버릴 것이므로 루프를 벗어나자
            }
            j--; //따옴표면 문자 복붙을 안해야하니까
        }else{
            chars[j] = chars[i]; //따옴표를 뺀 문자를 복붙합시다
        }
        j++;
    }
    // Find JavaDoc and understand the behavior of String(char[], int, int) constructor.
    // TODO: 적절한 code 입력
    token = new String(chars, 0, j);

    return token;
}
```

1.2 번 : .com 인정 (세부 토큰화 시키지 않기. 주석의 '토큰으로 인정하자'는 그냥 처음에 들어온 큰 토큰을 저장하자는 뜻)

```
@Override
protected void tokenAcronymProcessing(String token, int start, int end) {
    token = tokenComplexFix(token);

    // remove start and ending periods
    while (token.startsWith(".")) {
        token = token.substring(1);
        start = start + 1;
    }

    while (token.endsWith(".")) {
        token = token.substring(0, token.length() - 1);
        end -= 1;
    }

    // TODO: 적절한 code 입력
    // HINT: use if + return statement
    if (token.endsWith(".com")) { //만약 토큰이 .com으로 끝난다면
        addToken(token, start, end); //그 토큰은 dot(.)이 문자열 중간에 붙어있지만 토큰으로 인정하자
        return;
    }
```

1.3 번 : 마침표 사이 글자 2 자 이상일 경우 큰 토큰 인정.

```
// does the token have any periods left?
if (token.indexOf('.') >= 0) {

    // is this an acronym?  then there will be periods
    // at odd positions:
    boolean isAcronym = token.length() > 0;

    for (int pos = 1; pos < token.length(); pos += 2) {
        if (token.charAt(pos) != '.') {
            isAcronym = false;
        }
    }

    if (isAcronym) {
        token = token.replace(".", "");
        addToken(token, start, end);
    } else {
        // TODO: 적절한 code 입력
        // HINT: use String.split() method
        //       and addToken(token, start, end) statement;
        boolean moreThan2=false; //moreThan2라는 boolean을 정의.
                                 // periods 사이에 문자가 2자 이상이면 true, 아니면 false. default는 false로 설정
        String [] tokens;
        tokens = token.split("[.]",-1); //구둣점을 delim으로 토큰을 스플릿하자. 그리고 tokens라는 스트링의 어레이에 넣자

        for(int i=0; i < tokens.length ; i++){
            if(tokens[i].length() > 1) { //만약 tokens의 각 원소(스트링)가 2자 이상이면 moreThan2는 true다
                moreThan2 = true;
            }
        }

        if(moreThan2){ //periods 사이의 문자가 2자 이상인 게 있다면
            addToken(token, start, end); //그 토큰은 분리하지 말고 그냥 하나의 큰 토큰으로 인정하자 i.e. 300.194.38.9
            return;
        }

        int s = 0;
        for (int e = 0; e < token.length(); e++) {
```

2 번 문제

2.1 기본 Galago

2 번 문제에서는 bible.txt 를 토큰화 프로그램을 이용하여 출력한 결과를 바탕으로 ▲Zipf 분포를 그리며, ▲Zipf's Law 에서의 k 값들을 구하고, ▲주어진 bible.txt 가 Zipf's Law 를 따르는지 설명해야 한다. 그런데 교과서의 내용을 바탕으로 문제를 해석할 때, 분명하지 않은 점이 두 가지 있다.

첫 번째는, 교과서에 Zipf 분포는 (Rank, Freq)의 산점도로 그리는 방법과 Log-Log 스케일로 그리는 방법 두 가지가 나와 있다는 것이다. 숙제가 어떤 방법을 유도하는 건지 확실치 않아서 두 그래프 모두 그렸다.

두 번째는, 프로그램으로 출력한 데이터를 가지고서는 교과서의 Data Set 예제에 나와있는 것처럼 단어 하나하나의 Rank 를 알 수 없다는 것이다. 교과서의 표 4.4 를 보면 동일한 출현횟수를 보여도 Rank 의 차이가 있을 수 있다. (아래 < 표 1 > 참조)

Rank	Word	Frequency
1002	summit	5100
1003	bring	5099
1004	star	5099
1005	iImmediate	5099
1006	chemical	5099

< 표 1 > : 교과서의 표 4.4 일부

프로그램으로는 데이터가 <n 번의 횟수를 보이는 단어의 수, 단어의 출현횟수 n> 형태로 나온다. 이 형태의 데이터로는 'n 번의 횟수를 보이는 단어들'의 Rank 는 동일하게 설정될 수 밖에 없다.

그리고 이 때 Rank 를 설정하는 방법이 크게 2 가지 있을 수 있다. 하나는 단어의 출현횟수 n 을 내림차순으로 정렬한 후, 그에

따라 Rank 를 부여하는 방법이다. 이렇게 rank 를 설정하면, <n 번의 횟수를 보이는 단어의 수, 단어의 출현횟수 n> 형태의 데이터 row 가 500 여 개이므로 rank 의 최댓값은 500 몇 정도가 된다.

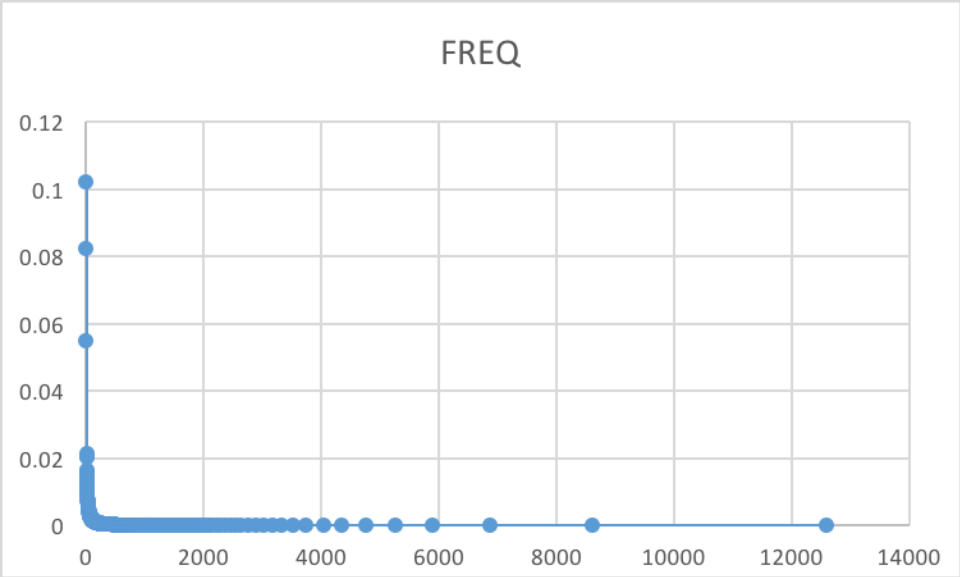
다른 방법은 n 번 횟수를 보이는 단어들이 여러 개라면 그 단어의 수만큼 rank 가 밀리는 것이다. 예를 들어, 위에 삽입한 표에서 Freq=5099 인 단어는 총 4 개이므로, 4 개의 단어 bring, star, immediate, chemical 은 모두 1002+4 = 1006 의 rank 를 동일하게 갖는다. (chemical 이 Freq=5099 인 단어 중 제일 마지막 단어이다.) 이 방식으로 rank 를 설정하면 rank 의 최대값은 단어의 총 갯수인 12600 정도가 나온다. (2.1 의 경우)

이렇게 output 데이터의 형태가 고정되어 있어서 Rank 를 설정하는 방법이 2 가지가 있을 수 있으나, 후자가 더 그럴듯한 방법이라 생각해서 본 워드파일에서는 후자의 방법을 이용하여 k 값과 그래프를 나타내었다. 전자의 방법을 사용했을 때 결과도 크게는 다르지 않는데, 어쨌든 첨부한 엑셀 파일에 표시해 두었다.

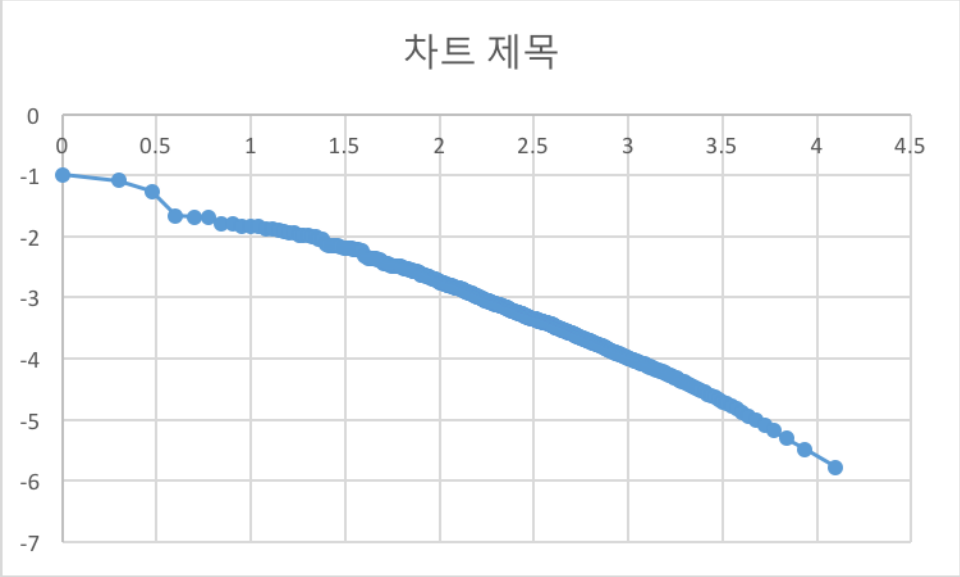
다음 페이지에는 결과 그래프를 그려놓았다.

다음은 기본 Galago 를 사용하였을 때 Zipf 분포를 일반 방법과 Log-Log 스케일 방법으로 그린 그래프이다.

A. 일반 방법(여기서 Freq 는 전체 단어의 수로 나눠서 확률값으로 나타냄)



B. Log-Log 스케일

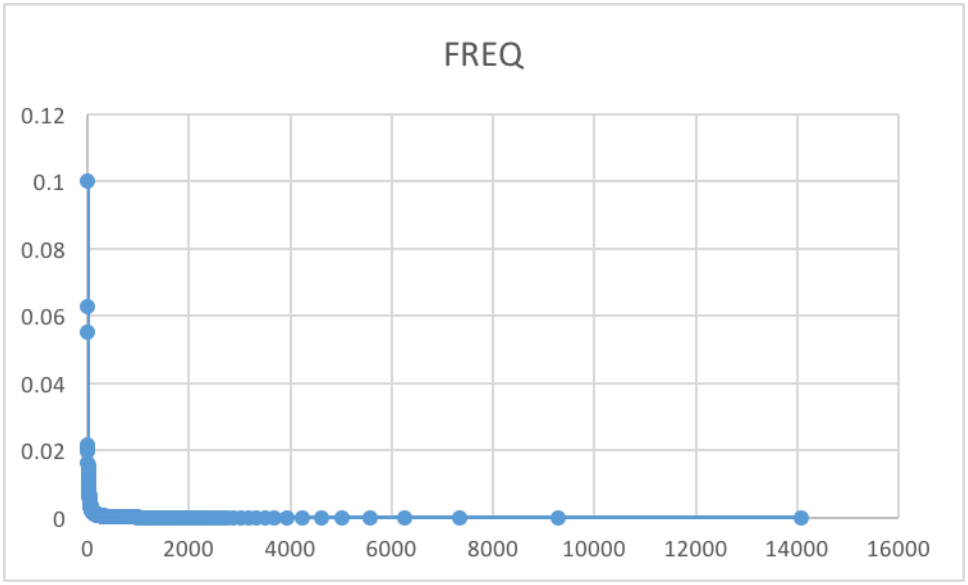


그래프를 보면 Zipf 분포를 꽤 따르는 것으로 보인다. 물론 Rank 의 양 끝은 완벽히 따르지 않는 것을 확인할 수 있다.

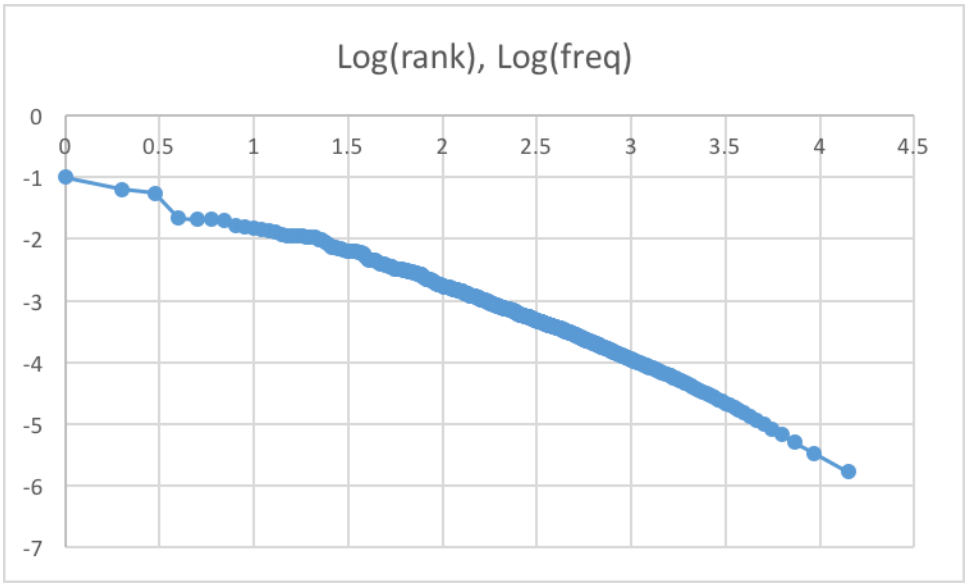
2.2 1 번 문제에서 수정한 토큰화 함수를 사용한 결과

다음은 1 번 문제에서 수정한 토큰화 함수를 사용하여 얻은 zipf 분포이다.

A. 일반 방법



B. Log-Log 스케일



2.2 역시 Zipf 분포를 꽤 따른다. 물론 Rank 의 양 끝은 완벽히 따르지 않는 것 역시 확인할 수 있다.

2.3 2.1 과 2.2 의 비교

2.1 과 2.2 모두 Zipf 분포를 꽤 따랐다. 각각의 자세한 k 값들은 첨부한 MS 엑셀 파일에 자세히 나와 있지만, 정리하면 다음 < 표 2 >와 같다.

	2.1	2.2
k 값	약 100,000 ± 30,000	약 100,000 ± 30,000
단어의 수	12600	14080
비고	k 값은 rank 초기와 후반부에 100,000 에서 많이 벗어남	

< 표 2 >

결과적으로 2.1 과 2.2 모두 가시적인 큰 차이 없이 Zipf's Law 를 꽤 따른다고 할 수 있다.