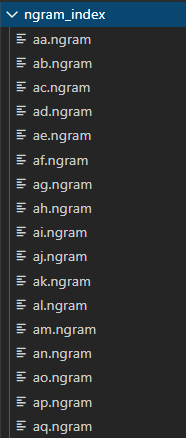
알고리즘 기말 대체과제

Approximate string matching 프로그램 보고서

2019320137

컴퓨터학과 황상민

1. 색인 파일 구조

2-gram단위로 색인을 해야 하므로, aa부터 zz까지 26\*26개 경우의 수를 각각 색인 파일로 만들었다. 총 676개의 색인 파일을 생성하여, ngram\_index라는 디렉토리 아래에 저장하였다.

각각의 ngram 파일들은, 각 패턴을 포함하는 문자열과 해당 문자열이 words.txt에서 몇 번째로 나타나는지에 관한 인덱스 정보를 포함한다.

Figure 1  
색인 파일 디렉토리 구조

예를 들어, aa.ngram 파일에는 aa 패턴을 포함하는 모든 문자열과 문자열의 인덱스가 한 줄에 하나씩 텍스트 형태로 작성되어 있다.

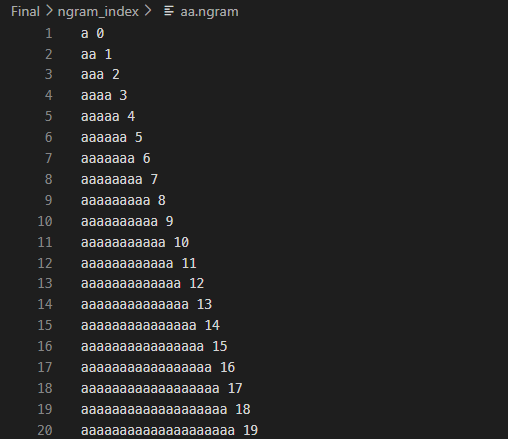


Figure 2  
ngram 파일 구조

한 단어를 색인할 때는 그 단어가 포함하는 모든 패턴에 해당하는 ngram 파일에 색인한다. 예를 들어, “abc” 라는 단어는 ab.ngram과 bc.ngram 모두에 색인한다.

이때 words.txt에 등장하는 한 글자 단어들은, 그 문자를 포함하는 모든 ngram 파일에 색인하였다. 예컨데 a라는 단어는 a로 시작하는 ngram(a\_.ngram)과 a로 끝나는 ngram(\_a.ngram) 파일에 모두 색인하여, 총 52개 ngram 파일에 색인하는 방식이다.

또한 search.c에서 쿼리로 한 글자 단어가 들어올 것을 대비하여, 길이가 2 이내인 단어들은 small.ngram이라는 파일에 따로 색인하였다.

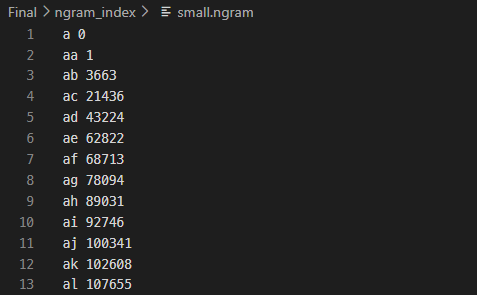


Figure 3  
small.ngram 파일

2. index.c 구현

앞서 설명한 색인 파일 구조에 맞게 파일을 생성하는 index.c 소스를 작성하였다. index.c 소스의 실행 흐름을 간단히 요약하여 나타내면 아래와 같다.

1. 현재 소스 코드와 같은 경로에 ngram\_index 디렉토리를 생성한다(sys/stat.h의 mkdir 함수 이용).

2. 생성한 디렉토리 하위에 26\*26개의 ngram 파일과 small.ngram 파일을 작성하기 위해 file write 스트림을 연다.

3. words.txt에서 단어들을 받아오기 위한 file read 스트림을 열고, 한 줄씩(한 단어씩) 받아온다.

4. 받아온 단어들을 2-gram 단위로 분석하여, 해당되는 ngram 파일에 색인한다.

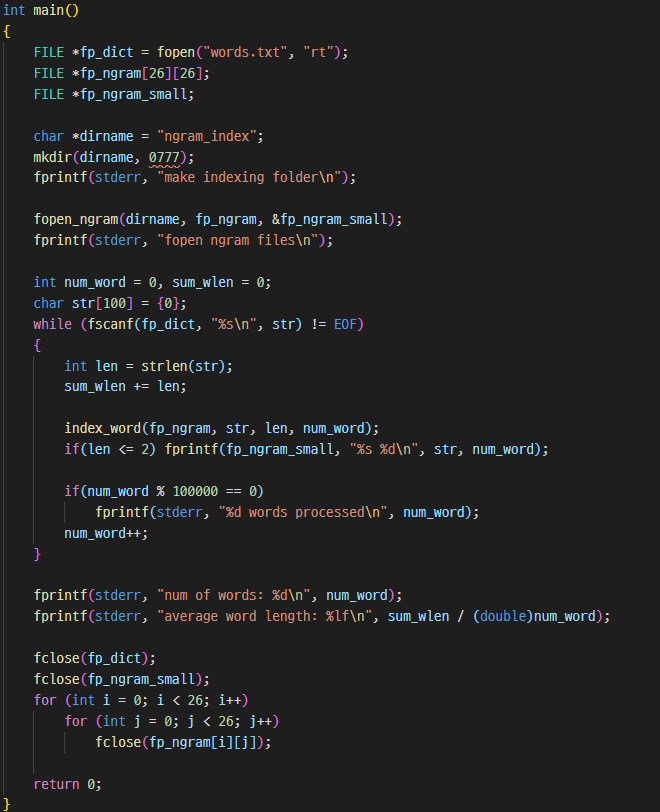


Figure 4  
index.c의 main 함수

실제 코드이다. 먼저 mkdir 함수를 호출하여 디렉토리를 생성하고, fopen\_ngram 함수를 호출하여 file write stream을 여는 동작을 수행한다.  
이후 fscanf로 words.txt의 단어들을 하나씩 받아온 뒤, index\_word 함수를 호출하여 색인한다. index\_word 함수는 단어를 2-gram 단위로 분석하여 해당되는 ngram 파일에 색인하는 함수이다.  
이때 단어 길이가 2 이하일 경우 fp\_ngram\_small 파일 포인터를 이용하여 small.ngram에 추가로 색인하는 모습을 볼 수 있다.

3. Search.c 구현

1) 후보 선정 방식

기본적으로, 입력한 문자열에 대해 유사 문자열이 될 수 있는 후보군은 입력을 2-gram 단위로 분석했을 때, 해당되는 2-gram 파일에 색인된 문자열 모두이다. 예컨데 입력이 “power”이었다면, po, ow, we, er 총 4개 ngram 파일에 속한 문자열들이 후보가 될 수 있다.

문제는 이때 검색된 후보들이 매우 많아질 수 있다는 것이다. 대략 하나의 ngram 파일에 10만 개 정도의 단어가 있다고 가정하면, (입력에서 등장한 2-gram 패턴의 수) \* 10만 개의 후보가 생성된다.

본 보고서에서도 이러한 점을 인지하여, 검색의 정확도를 잃지 않으면서 후보의 수를 줄이는 방법을 고민해보았다. 대안으로 생각한 것은 여러 2-gram 파일에서 중복해서 등장한 단어 위주로 후보군을 한정하는 것이었다. 그러나 이 방법은 중복 단어인지 검사하는 과정이 으로 매우 비효율적이라는 점에서 한계가 있었다. AVL tree를 이용하여 map 자료구조를 구현하더라도, 후보의 수만큼 노드를 생성해야 하므로 속도 향상에 비해 메모리 측면에서 얻는 비효율이 너무 크다고 생각하였다.

따라서 후보를 한정하는 방식을 도입하기에는 시간/공간적인 오버헤드가 너무 크다고 생각하여, 본 보고서에서는 **매칭된 ngram 파일에 속한 모든 문자열을 편집 거리 계산 후보에 포함시키는 방법**을 택하였다.

2) 링크드리스트 기반 유사 문자열 집합 산정 방식

후보를 산정한 후에는 각 후보에 대해 편집 거리를 계산한 후, 편집 거리가 가장 작은 상위 10개 문자열을 산출해야 한다.

다뤄야 하는 후보의 수가 매우 많기 때문에, 모든 후보들을 실행 도중에 메모리상에 올려 두는 방식은 메모리 효율 측면에서 매우 비효율적이다. 따라서 본 보고서에서는 메모리 사용을 최대한 줄이는 방향을 고안하였다.

그 방법은, 편집 거리가 작은 상위 10개 문자열만을 졍렬된 링크드리스트에 저장해 두는 것이다. 매 후보의 편집 거리를 계산한 후에, 편집 거리가 지금까지 계산한 후보들 중 상위 10등 안에 들어갈 경우에만 링크드리스트에 해당 후보를 추가한다. 만약 10위 안에 들지 못한다면 그 후보는 유사 문자열 집합에 절대 포함될 수 없으므로, 메모리상에 저장하지 않는다.

이때 한 문자열을 링크드리스트에서 중복해서 삽입하지 않도록 했다. 링크드리스트의 노드 수가 최대 10개이므로, 중복 검사에 드는 비용은 그리 크지 않다. strcmp로 비교하면 비용이 크므로 문자열과 같이 저장한 인덱스 번호를 이용해 같은 문자열인지 아닌지를 판별하였다.

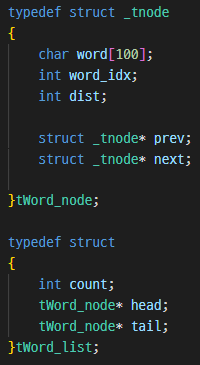


Figure 5  
편집 거리가 작은 상위 10개 문자열을 저장할 링크드리스트

3) 주요 함수 설명

(1) main 함수

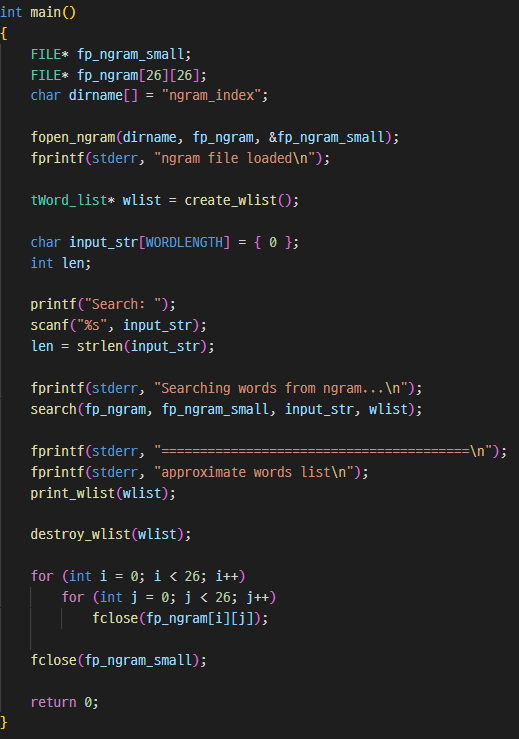


Figure 6  
search.c의 main 함수

전체적인 실행 흐름은 아래와 같다.

1. ngram 파일을 읽어들이기 위한 파일 스트림을 만든다.  
2. 위에서 언급한 용도로 사용될 링크드리스트를 생성한다.  
3. 사용자로부터 입력을 받는다.  
4. 입력 문자열에 대해, ngram 파일을 바탕으로 가장 유사한 10개의 문자열을 산정한다. search 함수가 호출되는 부분이다. 이 문자열은 편집 거리가 가장 작은 순서로 링크드리스트에 저장된다.  
5. 산정한 유사 문자열을 출력하고, 사용한 메모리를 모두 해제한다.

(2) search 함수

main 함수 실행 흐름의 4. 에 해당되는 부분으로, ngram 파일을 읽어 가장 유사한 10개 문자열을 산정하는 과정이다.

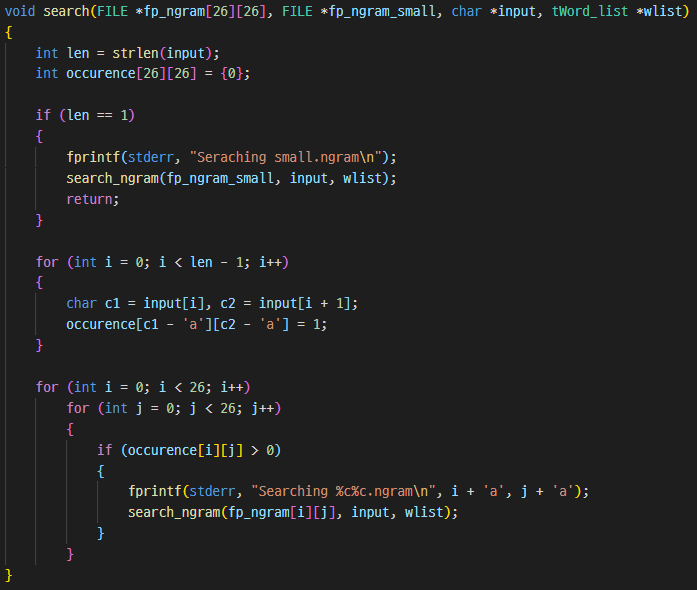


Figure 7  
search.c의 search 함수

입력 문자열을 2-gram 단위로 분석하여, 해당되는 2-gram 파일에 대해 search\_ngram을 호출한다. 만약 입력 문자열의 길이가 1이라면 fp\_ngram\_small이 가리키는 small.ngram 파일을 읽도록 한다.

(3) search\_ngram 함수

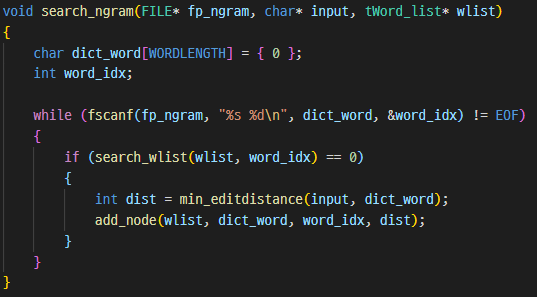


Figure 8  
search.c의 search\_ngram 함수

단일 ngram 파일을 읽어, 그 파일에 적힌 모든 문자열을 후보로 하여 입력 문자열과의 편집 거리를 구한다. 먼저 편집 거리를 구하기 전에, 현재 후보가 링크드리스트에 이미 존재하는지 검색하여(search\_wlist) 중복 삽입이 일어나지 않도록 한다. 만약 존재하지 않은 문자열이라면 편집 거리를 구하고(min\_editdistance), 링크드리스트에 삽입한다(add\_node).

(4) add\_node 함수

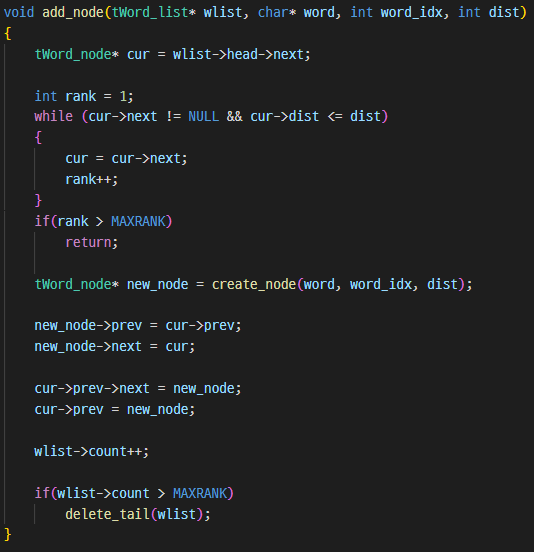
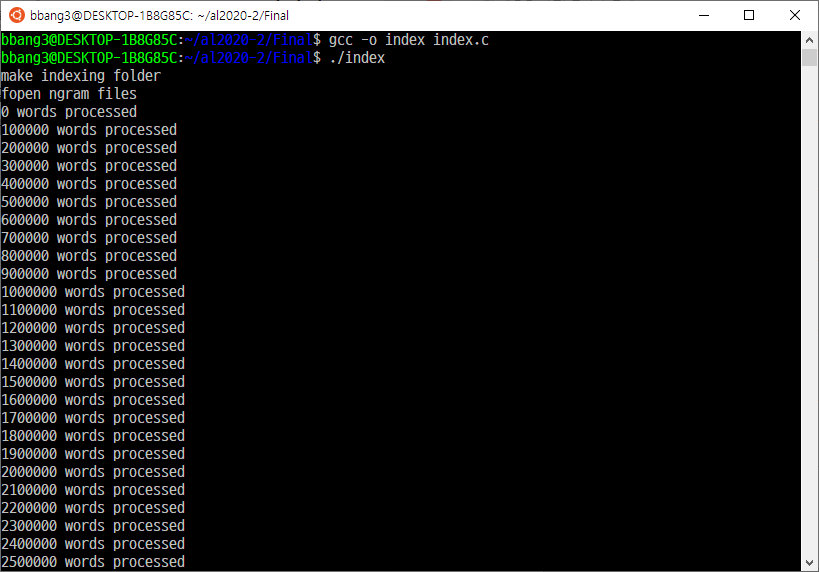
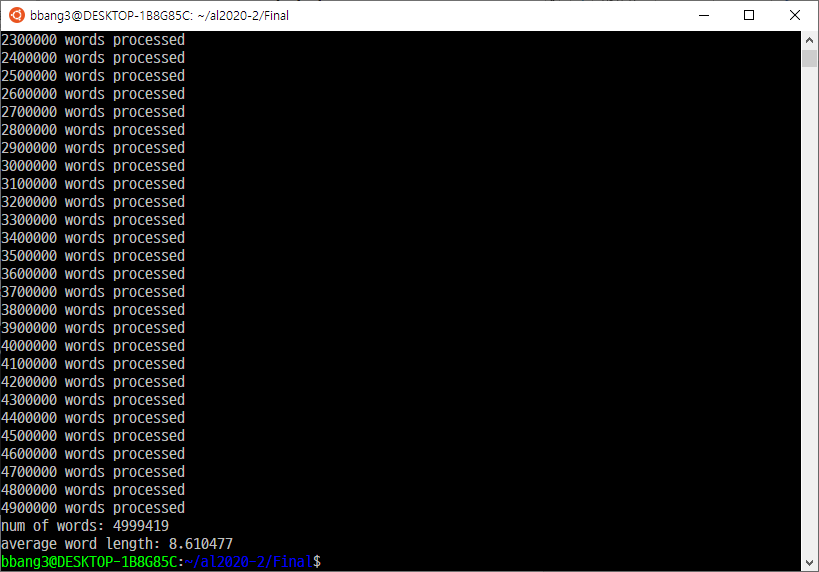


Figure 9  
search.c의 add\_node 함수

언급했던 대로, 후보 중 편집 거리가 가장 작은 10개만을 저장해두는 링크드리스트이다. 먼저 편집 거리(dist) 값을 비교하여, 지금 추가할 노드의 편집 거리가 링크드리스트 상에서 몇 번째로 작은지를 계산한다. 만약 10위 이후라면(rank > MAXRANK), 링크드리스트 상에 저장해둘 필요가 없는 후보이므로 삽입하지 않는다. 10위 이내라면, 10위에 있던 노드를 삭제하고(delete\_tail), 새로 들어온 후보를 링크드리스트에 추가한다. 이때 정렬된 상태를 유지하며 추가하도록 한다.

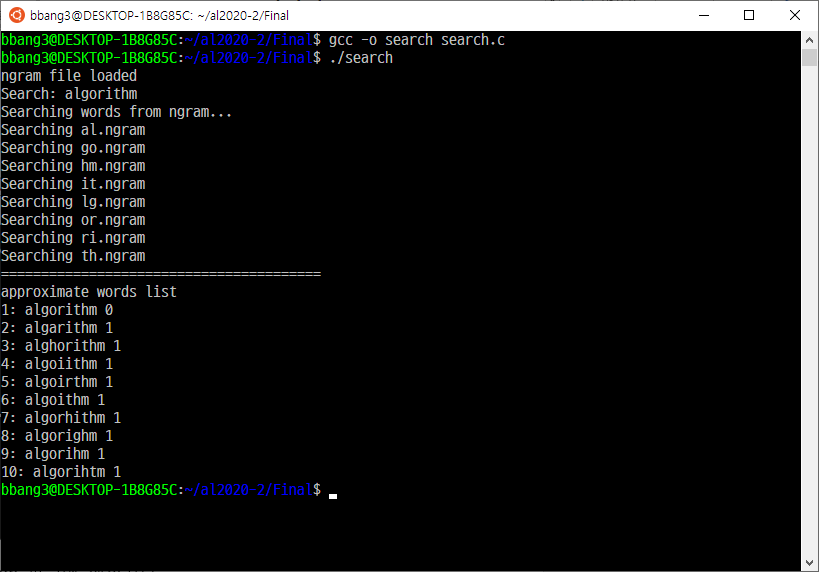
4. 실행 화면

먼저 index.c 파일을 컴파일하고 실행하여 ngram 파일을 생성한다.

words.txt에 든 모든 단어들이 각각 맞는 ngram 파일에 색인된다.

이후 search.c를 컴파일하고 실행한다.



입력한 문자열에 맞는 ngram 파일을 참조하여 편집 거리를 계산한 뒤, 상위 10개 유사 문자열을 출력하는 모습을 볼 수 있다.