

과목명 HW #1

알고리즘 H/W #2

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.

2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.

3. 나는 보고서의 내용을 조작하지 않겠습니다.

4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.

5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2018년 4 월 18일

학부 정보통신공학과

학년 3학년

성명 김 권 중

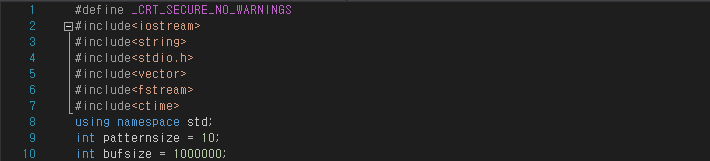
학번 12141668



1. 개요

이번 내용은 텍스트 파일 속에서 특정한 단어를 찾는 것입니다. 우선 KMP와 Boyer\_Moore의 방식의 스트링 탐색을 이용하기 위해 구현하였습니다. 또한 각각의 성능을 비교하기 위해 실행시간 측정을 하였습니다. 성능 비교는 같은 단어를 같은 텍스트 파일에서 실행하였습니다. 또한, 좀 더 자세한 성능 비교를 위해 특정 단어를 RFC2616.txt파일에 넣어 실행시켰습니다. 특정 단어로는 패턴이 있는abababacd, 패턴이 없는 qwerty를 추가하였습니다. 그 다음으로 아스키 코드에 없는 한글 단어를 추가하였습니다. 이러한 데이터 셋을 이용하여 성능 비교 분석을 해 보았습니다.

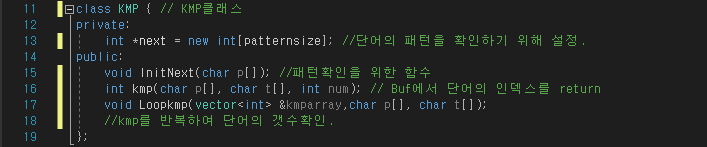
1. 상세 설계내용



첫째 줄은 main에서 fopen을 사용하였을 경우 오류가 뜨는 것을 방지하기위해 사용하였습니다.

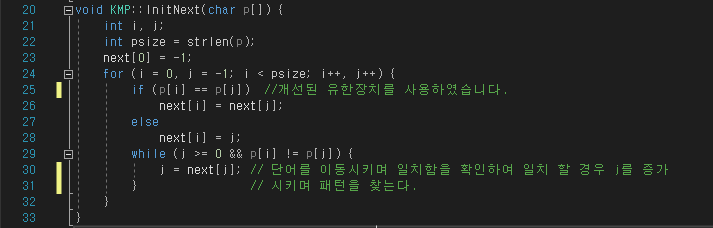
전역변수로 patternsize를 10, 텍스트 파일을 저장하기위한 bufsize를 1000000으로 설정하였습니다.

KMP알고리즘



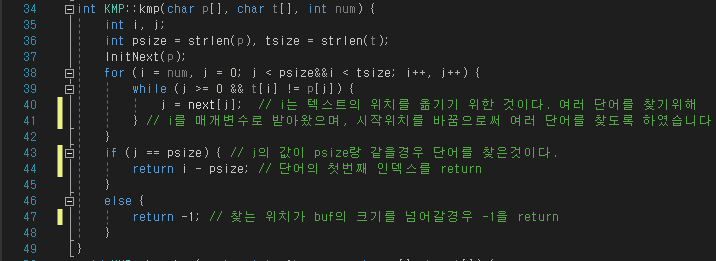
KMP알고리즘을 실행하기 위해 클래스를 이용하여 선언해주었습니다.

Patternsize는 전역변수로 10을 설정해주었습니다.



InitNext 함수입니다. 매개변수로 단어를 받아올 수 있도록 하였습니다.

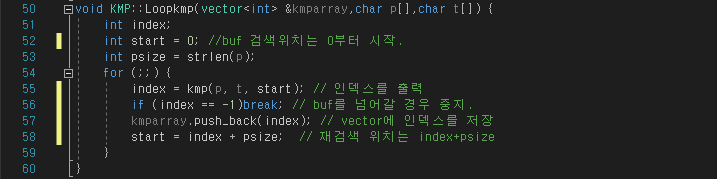
단어를 옮김으로써 같을 경우 j를 증가, 일치하지 않을 경우 다시 j를 감소 시킴으로써 단어의 패턴을 찾아 주었습니다.



kmp함수입니다. 매개변수로 단어, text 그리고 재 검색 위치를 받아왔습니다.

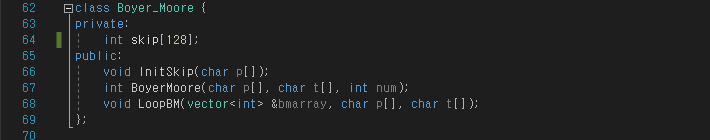
InitNext를 이용해 단어의 패턴을 확인하였습니다. next배열이 움직이면서 text와 비교합니다. Text랑 패턴이 일치하면 j의 크기가 증가하며, j랑 패턴의 사이즈가 일치할 경우 단어를 찾은 것 이며, 단어의 첫번째 위치 인덱스를 반환합니다. 반복적인 검색을 통해 텍스트를 넘어갈 경우 -1을 리턴시켰습니다.

반복적인 단어 검색을 위하여 for의 I 인덱스의 시작을 num으로 바꿔주었습니다. 이는 loopkmp를 이용하여 텍스트 전체를 검색하도록 하였습니다.



Kmp를 반복적으로 돌리기 위해 만든 함수입니다. Vector와 단어, 텍스트를 받아옵니다.

무한루프를 통해 kmp를 돌며 index를 출력합니다. Index가 -1일 경우 break 하여 무한루프를 나오게 됩니다. 또한 재시작 위치는 index + psize를 하여 출력된 단어의 다음 위치부터 확인하도록 하였습니다.

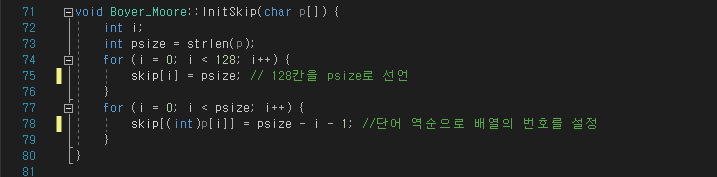


Boyer\_Moore 클래스입니다.

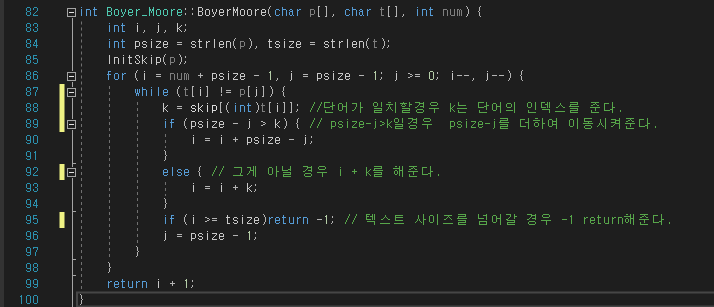
skip배열은 아스키코드를 저장하기 위해 128사이즈로 설정하였습니다.

KMP클래스와 마찬가지로 3가지의 함수를 정의해주었습니다.

Boyer\_Moore알고리즘

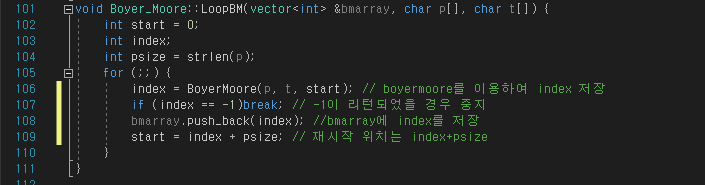


skip배열을 전부 psize로 선언해준다. 다음 단어의 배열의 역순으로 1부터 psize-1까지 채워준다. 예를들면 ABCD이면 A의 인덱스에 0, B의 인덱스에 1, C의 인덱스에 2, D의 인덱스에 3이 들어간다.



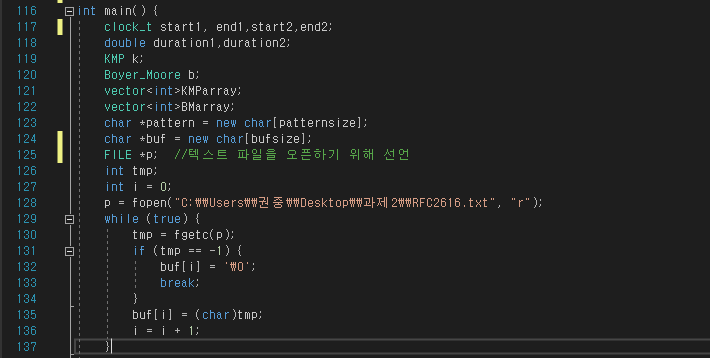
BoyerMoore 함수입니다. 단어와 텍스트 그리고 재 시작 위치를 위한 num을 받아옵니다.

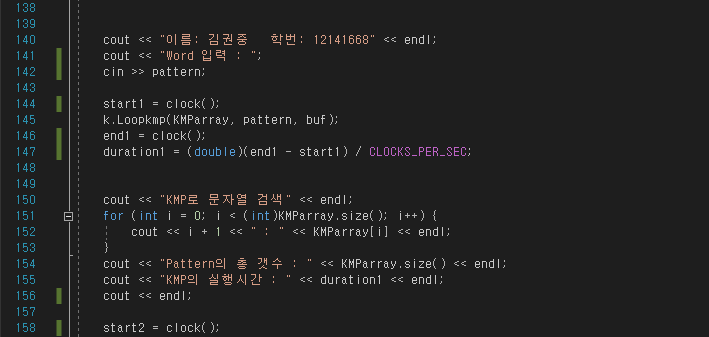
boyermoore역시 i로 텍스트의 위치를 조정합니다. 따라서 i에 num을 추가하여 텍스트를 지속적으로 읽을 수 있도록 하였습니다.

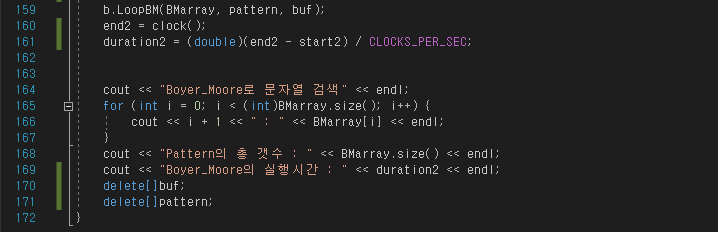


무한루프를 이용해 텍스트의 처음부터 끝까지 단어를 확인하고 저장하였습니다. -1이 리턴 되었을 경우 무한루프를 멈추게 합니다. Boyermoore를 이용해 찾은 인덱스는 bmarray에 저장하였습니다.

메인함수

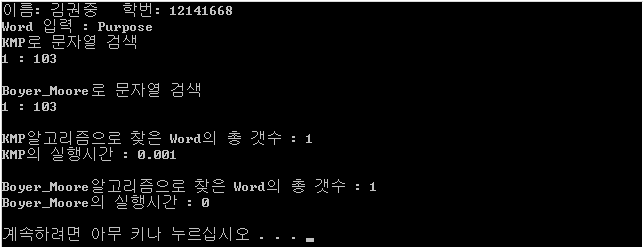






Fopen을 이용하여 RFC2616.txt를 불러왔습니다. 한 줄 씩 읽기에 for를 이용하여 문장 전체를 읽도록 하였습니다. Buf에 저장하기 위해 1000000을 선언해주었습니다. Clock을 이용하여 loopkmp와 loopbm의 측정 시간을 저장하여 출력하였습니다. KMParray, BMarray를 vector로 선언 해 주었으며, 각각의 출력된 인덱스를 저장하였습니다.

1. 실행 화면



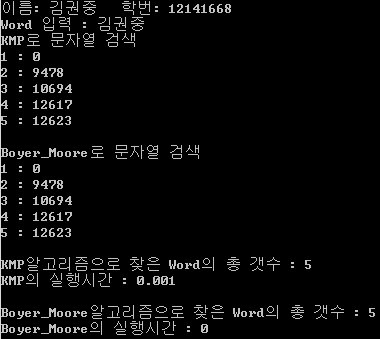
실행을 시켰을 때 나오는 터미널 창입니다. 단어를 입력하면 KMP, Boyer\_Moore 알고리즘으로 찾은 단어의 인덱스를 출력시킵니다. 각각의 인덱스를 나열하면, Word의 총 개수를 출력을 하였습니다. 또한 각각의 실행시간을 비교하기 위해 실행시간을 출력하였습니다. 103은 buf 배열에서 Word의 W의 인덱스입니다.

|  |  |
| --- | --- |
| Data set : 50 |  |
| Data set : 100 |  |
| Data set : 150 |  |
| Data set : 200 |  |
| Data set : 250 |  |

KMP와 Boyer\_Moore 알고리즘의 성능을 비교하기 위해 패턴이 있는 abababacd의 단어를 Data set을 50씩 증가시켜 비교해보았습니다.

|  |  |
| --- | --- |
| Data set : 50 |  |
| Data set : 100 |  |
| Data set : 150 |  |
| Data set : 200 |  |
| Data set : 250 |  |

이번엔 패턴이 있는 형식이 아닌 qwerty의 단어를 입력하여 비교해 보았습니다.



한글을 이용해 검색해보았습니다. 한글을 Skip배열안에 아스키코드로 저장을 하지 않았지만, Boyer\_Moore 역시 출력하는 것을 알 수 있었습니다.

1. 분석 및 결론

KMP와 Boyer\_Moore 알고리즘을 돌려보았습니다. 적은 양의 데이터 셋에서 각각의 알고리즘은 실행속도가 크게 차이 나진 않았습니다. 성능비교를 하기 위해 패턴이 있는 단어와 패턴이 없는 단어를 비교해보았습니다. 각각 50개씩 데이터 셋을 늘렸습니다. 실행 시 마다 실행속도가 달라 10번의 결과 후 평균값을 이용해 값을 내어주었습니다.

위 그래프는 adadadacd라는 word의 단어 데이터 셋을 50씩 늘려가며 비교한 그래프 입니다.

패턴을 가진 경우 KMP의 성능의 더 좋게 나 올 거란 예상과 달리, Boyer\_Moore의 실행속도가 더 빠른 것을 알 수 있었습니다.

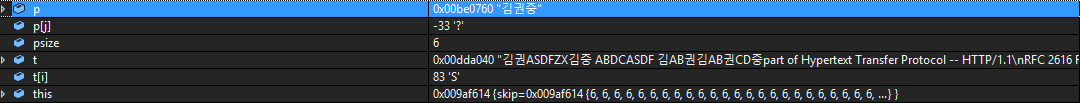
위 그래프는 qwerty라는 word의 단어 데이터 셋을 50씩 늘려가며 비교한 그래프입니다.

KMP와 Boyer\_Moore 알고리즘 둘다 크게 차이가 없어서 중간에 값이 Boyer\_Moore의 실행 속도가 더 느려진 경우가 있었지만 대부분의 경우 Boyer\_Moore의 성능이 더 빠른 것을 알 수 있었습니다.

다음으로 word에 입력한 것은 한글입니다. 텍스트 문서에 김권중을 넣었을 경우 KMP와 Boyer\_Moore의 성능을 확인해보고자 하였습니다.

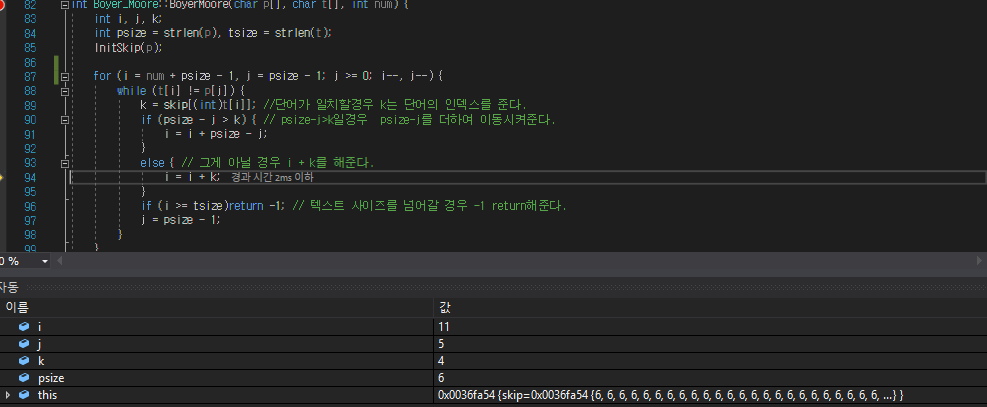
우선 KMP 알고리즘의 경우 특정 단어를 저장하여 비교하는 부분이 없기 때문에 자세한 성능분석을 하지 않았습니다.

Boyer\_Moore 알고리즘의 경우 skip이란 배열에 아스키코드를 저장하였기에 아스키코드에 없는 한글이 출력되지 않을 줄 알았습니다.



“김권중”을 입력한 경우입니다. 한글은 글자 하나를 2byte취급하여 psize는 6이 되었습니다.

한글은 아스키코드에 0~128에 존재하지 않기에 skip배열은 모두 6으로 처리가 되었습니다.



처음 생각한 내용과 달리 skip배열은 텍스트가 일치하지 않을 경우 인덱스의 위치를 바꾸어주기 위한 것이기 때문에, Boyer\_Moore역시 탐색이 되는 것을 확인하였습니다. K엔 항상 6이 들어가게됩니다.

“김권중”이란 단어는 아스키 코드에 없기에 i=i+k가 반복해서 실행됩니다. 위의 실행 결과 창에서 본 것 처럼 탐색되어 나오는 것을 확인하였습니다. 하지만 i=i+k에서 k는 6입니다. 6은 psize이며 이는 한글로 3글자입니다. 따라서 한글이 입력이 될 경우 배열 한 칸 씩 비교하여 검색되는 것을 알 수 있었습니다.

KMP 알고리즘의 장점:

KMP의 경우 word자체에서 패턴을 분석하고, 텍스트와 비교하여 텍스트를 탐색하는 알고리즘입니다. 따라서 word의 단어 크기의 배열만 있으면, 직선적 알고리즘이 아닌 불일치 문자를 건너뛰고 탐색을 할 수 있는 장점을 가졌습니다.

KMP 알고리즘의 단점:

많지 않은 내용의 text에서 크지않은 데이터 셋을 비교하다보니, Boyer\_Moore의 알고리즘이랑 비슷한 실행속도를 보였습니다. 하지만 미세하게 Boyer\_Moore의 실행 속도에 못 미치는 것을 알 수 있었습니다.

Boyer\_Moore 알고리즘의 장점:

Boyer\_Moore의 경우 위의 그래프에서 보듯이 KMP보다 속도가 빠른 것을 알 수 있었습니다. 또한 skip에 저장되지 않은 문자열도 탐색은 가능한 것입니다.

Boyer\_Moore 알고리즘의 단점:

InitSkip 함수에서 보듯이 미리 탐색할 문자에 대해 정보가 있어야 되는 것입니다. 현재 위 함수에선 특수문자와, 영어에 대해서만 skip 배열에 아스키코드로 정보를 준 상태 입니다. 허나 한글, 한문등의 저장되어 있지 않은 문자열에 대해서는 직선적 알고리즘과 같은 방식으로 문자열을 탐색하게 됩니다.