SW프로그래밍(2) Term Project

20125757 노요한

중앙대 융합공학부 디지털이미징공학

Github : <https://github.com/dygks232/SW_Programming_2_Project>

e-mail : [shqkdro@naver.com](mailto:shqkdro@naver.com)

String Searching & Ranking Program using Boyer-Moore Algorithm

Yohan Noh

Department of Integrative Engineering

|  |  |  |
| --- | --- | --- |
| Ⅰ. 서론  어떤 문자열 S에서, 어떤 패턴 P를 찾는 문자열 검색 문제는 오래전부터 연구가 되어온 분야이다.  이 연구들은 주로 문자열 S의 특정 인덱스 i에서 시작하는 문자열이 패턴 P와 일치하는지 한 글자 씩 비교하다가 완전히 일치했을 때, 혹은 일치하지 않았을 때, 바로 다음 인덱스인 i+1에서 검사하는 것은 비효율적이라는 점에서 시작되었다.  이를 해결하기 위해 개발된 많은 알고리즘들[1]이 있는데, 가장 유명한 것으로는 Knuth-Morris-Pratt algorithm이 있으며, 이 외에도 Rabin-Karp algorithm, |  | Bitap algorithm, Boyer-Moore algorithm 등이 있다.  문자열 검색 알고리즘의 대상이 되는 문자열 S가 하나가 아닌, 여러 개의 집합이 된다면 이는 새로운 분야의 연구로 발전시킬 수 있다. 특정 단어를 검색했을 때, S의 집합 중 어느 것을 우선적으로 보여줄 것인지가 중요해지는데, 가장 알기 쉬운 예가 검색 포털 사이트이다.  본 프로젝트에서는 문자열 검색을 위해, 본인이 배웠었고 다뤄봤던 KMP algorithm이 아닌, Boyer-Moore algorithm의 원리를 터득하고 사용해 보며, Ranking의 기준을 설정함으로써 텍스트 파일들의 목록에서 특정 키워드를 검색하고, 결과를 보여주는 소프트웨어를 만들고자 한다. |
| Ⅱ. 보이어-무어 알고리즘[2][3]  문자열 S에서 특정 패턴 P를 찾을 때, S의 인덱스 i에서 검사 후, 다음에는 어느 인덱스를 검사해야 효율적인지를 해결하기 위해서, 이 알고리즘은 Good Suffix Shift, Bad Character Shift의 2가지의 shift로 구성되어 있다.    그림 1. Good Suffix Shift (접미사 완전 일치)  보이어 무어 알고리즘은 패턴 P는 왼쪽에서 오른쪽으로 움직이지만, 정작 그 문자 비교 작업은 오른쪽에서 왼쪽으로 진행된다. 그림 1에서 보다시피 S[0]에서 P와의 비교를 진행하는데, 오른쪽에서부터 검사를 시작하여 G, A가 일치한 후, S[5]와 P[5]가 불일치했다. 이 때, 일치한 패턴 P의 부분문자열 “AG”가 Good Suffix가 되며, 아직 검사하지 않은 왼쪽의 부분문자열에서 “AG”의 위치를 찾는다. 이후 패턴 P의 또다른 “AG”의 위치를 S의 “AG”의 위치와 일치시키는 작업이 Good Suffix Shift이다.      그림 2. Good Suffix Shift (접미사 부분 일치) |  | 아직 탐색하지 않은 P의 나머지 부분에서 Good Suffix를 찾지 못할 경우에는, 그림 2와 같이 접미사의 길이를 하나씩 줄여가면서 이를 나머지 부분에서 찾으면 된다. Shift 작업은 그림 1과 마찬가지로 “AG”의 위치를 맞춰주면 된다.  Good Suffix의 길이를 줄여가며 나머지 부분에서 찾더라도 끝내 못 찾은 경우에는, 패턴 P를 패턴의 길이만큼 shift시켜주면 된다.    그림 3. Bad Character Shift  그림 3과 같이 첫 번째 글자부터 일치하지 않을 경우에는, S의 불일치한 인덱스에 있는 문자가 Bad Character가 되며, P의 나머지 부분에서 해당 문자를 탐색하여 S와 P의 Bad Character가 일치하게 shift를 시켜주면 된다.  이 알고리즘에는 Suffix가 어느 길이만큼 일치할 때, 얼마만큼 shift를 해야 하는지에 대한 것들과 같은 정보가 담긴 Table들을 미리 생성하는 전처리 과정이 있다. S의 길이를 n, P의 길이를 m, 문자의 종류의 수를 k(보통 1byte로 나타낼 수 있는 종류의 수인 256로 설정한다.)라 할 때, 본 알고리즘은 전처리 하는 데에 , 검색하는 데에 최소 최대 의 시간 복잡도가 소요되고, 의 공간 복잡도가 필요하다.  구현 코드[4]는 구글 위키피디아의 코드를 참고하였다. |
| Ⅲ. Ranking  S의 집합에서 P를 검색할 때, 집합의 원소 각각의 결과에 대해서 순위를 매기는 것은 고려할 수 있는 많은 것들이 있다.  프로젝트를 시작할 때에는 검색 키워드의 출현 빈도, 대소문자의 차이, 키워드 사이의 공백 여부 의 3가지에 대하여 순위를 매기고자 하였지만, 제공되는 입력 데이터의 특성상 오탈자로 인한 키워드 사이의 공백 여부는 고려하지 않아도 문제가 없었기에 이를 제외하였다.  1순위로 검색 키워드와 완전히 동일한 키워드의 수가 높은 순으로 텍스트 파일을 나열하며, 2순위로 검색 키워드와 동일하지만 대소문자의 차이가 있는 키워드의 수가 높은 순으로 나열한다.  대소문자의 차이만 존재하는 키워드의 수를 측정하기 위해 C언어의 string.h의 strlwr(), strupr()을 사용하여 다시 검색하는 방법을 사용했다.  Ⅳ. 결과  본 프로젝트는 Windows OS, Visual Studio 2017 x86에서 구현하였다.  “industry”를 검색했을 때의 결과는 그림 4와 같다. |  | 완전히 일치하는 키워드의 수가 왼쪽에 기록되고, 대소문자의 차이만 존재하는 키워드의 수가 오른쪽에 기록된다.  Ⅴ. 결론  이렇게 개발한 프로그램의 동작 속도를 C의 strstr()과 비교해 보려 하였다. 하지만 0.1~0.2초 내외의 차이만 존재하였고, strstr()이 앞서는가 하면, 본 프로그램이 앞서는 경우도 있었다. strstr()은 인덱스 i에서 검사 후 인덱스 i+1에서 다시 검사하는 단순한 알고리즘임에도 이러한 결과가 나왔다.  조사해본 결과, C의 strstr()은 내부에서 최적화가 되어있으며, 대게 어셈블리 언어로 작성되어 있기 때문에 이러한 결과가 나온다고 한다.[5] |

|  |  |  |
| --- | --- | --- |
| 참고문헌  [1] <https://en.wikipedia.org/wiki/String_>  searching\_algorithm  [2] <http://itguru.tistory.com/101>  [3] <http://xenostudy.tistory.com/72>  [4] <https://en.wikipedia.org/wiki/Boyer>  %E2%80%93Moore\_string\_search\_  algorithm  [5] <https://stackoverflow.com/questions>  /7586990/strstr-faster-than-algorithms |  |  |