EUC-KR 텍스트에서 오검색을 제거한 문자열 검색 알고리즘[1]

김은상○★, 김진욱★★, 박근수★

★서울대학교 컴퓨터공학부, ★★인하대학교 컴퓨터정보공학부

eskim@theory.snu.ac.kr, gnugi@inha.ac.kr, kpark@theory.snu.ac.kr

String Matching Algorithm without False Matches for EUC-KR Texts

Eunsang Kim○★, Jin Wook Kim★★, Kunsoo Park★

★School of Computer Science and Engineering, Seoul National University,

★★School of Computer Science and Information Technology, Inha University

1. 서론

한글, 중국어, 일본어 등은 영어 알파벳에 비해서 훨씬 많은 문자를 가지고 있으며, 이러한 언어의 문자 1개를 표현하기 위해서 2바이트 이상을 사용하게 되었다. 편의를 위해서 기존의 ASCII 문자들을 1바이트로 표현하기 때문에, 하나의 텍스트에 1바이트 문자와 다중바이트 문자가 공존하게 되었다.

문자열 완전일치 검색 알고리즘은 지금까지 많은 연구가 되어왔지만, EUC-KR 등 다중바이트 문자집합에 대해서는 연구된 것이 부족한 상황이다. 그리고 지금까지 연구된 문자열 완전일치 검색 알고리즘들은 다중바이트 문자를 포함한 문자열에 그대로 적용할 수 없는 문제가 있다. EUC-KR 텍스트에서 기존의 문자열 완전일치 검색 알고리즘을 사용했을 때 오검색이 발생하는 것도 그 중의 한가지이다.

오검색이란, 텍스트 상에서는 실제로 해당 패턴이 존재하지 않지만 검색 알고리즘이 패턴이 존재한다고 결과를 출력하는 것을 말한다. 예를 들면, EUC-KR 문자집합에서 텍스트가 '영도'(BFB5 B5B5)이고, '도'(B5B5)라는 문자열을 검색한다고 하자. 텍스트의 첫 번째 문자의 두 번째 바이트('영'의 두 번째 바이트인 B5)와 패턴의 첫 번째 바이트가 일치하고, 텍스트의 두 번째 문자의 첫 번째 바이트('도'의 첫 번째 바이트인 B5)와 패턴의 두 번째 바이트가 같아서 텍스트의 '영'과 '도' 사이에서 오검색이 발생하게 된다. EUC-KR은 2바이트 문자 인코딩의 첫 번째 바이트와 두 번째 바이트가 모두 A1 ~ FE 영역에서 표현되기 때문에 오검색이 발생하게 된다.

오검색은 실제로 사용하는 편집기에서도 쉽게 발견할 수 있는데, 공개 소프트웨어로 널리 사용되고 있는 Vim [1] 편집기에서도 추가적으로 언어 파일을 설치하지 않을 경우에 오검색이 발생하고 있음을 확인하였다.

이 논문에서는 EUC-KR 문자집합을 사용한 텍스트 상에서 오검색을 피하고 문자열 완전일치 검색을 할 수 있도록 개선한 알고리즘을 제안한다.

2. 본론

기존의 Naïve 알고리즘은, 텍스트의 위치를 1씩 증가시키면서 해당 위치의 텍스트와 패턴을 비교하고, 패턴의 길이만큼 바이트 순서가 동일하다면 패턴이 해당위치에 존재한다고 출력하는 간단한 동작을 한다. 기존의 Naïve 알고리즘에서 오검색이 발생하지 않도록 개선하는 방법은, 현재 위치의 텍스트 바이트 $T[i]$가 2바이트 문자의 첫 번째 바이트일 경우($T[i] \geq 0x80$), i를 2 증가시켜서 다음에 비교할 $T[i]$가 2바이트 문자의 2번째 바이트가 되지 않도록 하면 된다. 즉, 텍스트와 패턴을 비교할 때 텍스트의 첫 번째 바이트 $T[i]$가 2바이트 문자의 두 번째 바이트가 되지 않도록 건너뛰는 것이다.

KMP [2] 알고리즘은 접두사 함수를 사용하여 문자열이 존재하는 위치를 검색함으로써, Naïve

[1] 본 연구는 기초기술연구회의 NAP 과제 지원으로 수행되었습니다. 이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사 드립니다.

알고리즘에서 필요없는 비교 연산을 줄이도록 한 알고리즘이다. 접두사 함수 π는 $\pi : \{1,2,\cdots,m\} \rightarrow \{0,1,\cdots,m-1\}$인 함수이며, $\pi[q] = \max\{k : P[1..k]$는 $P[1..q]$의 접미사$\}$로 표현할 수 있다. 즉, $\pi[q]$는 $P[1..q]$의 접미사이면서 $P[1..m]$의 가장 긴 접두사의 길이를 의미한다. KMP 알고리즘은 전처리 과정에서 접두사 함수를 만들고, 검색 과정에서 패턴과 텍스트를 비교하고 서로 다른 바이트인 경우에 접두사 함수를 사용하여 패턴을 이동시키고 다시 패턴과 텍스트를 비교하는 작업을 반복한다. 그런데 EUC-KR 텍스트 상에서 KMP 알고리즘을 사용할 때에도 역시 오검색이 발생하게 된다.

우리가 제안하는 개선된 KMP 알고리즘은 바이트 단위가 아닌 문자 단위로 접두사 함수를 구성하고 검색 과정을 진행하는 알고리즘이다. 개선된 KMP 알고리즘의 전처리 과정에서 만들어지는 접두사 함수 $\pi[q] = \max\{k : P[1..k]$는 $P[1..q]$의 접미사이고, $P[q]$와 $P[k]$가 모두 1바이트 문자이거나 2바이트 문자의 두번째 바이트$\}$이다. $P[q]$와 $P[k]$가 모두 1바이트 문자이거나 2바이트 문자의 두 번째 바이트라면, 즉 서로 같은 종류의 바이트라면, 접두사 함수 π에 의해서 패턴이 문자 단위로 이동하게 된다. EUC-KR 문자집합은 1바이트 문자가 0x00 ~ 0x7F 사이의 영역을 사용하고, 2바이트 문자의 바이트는 0xA1 ~ 0xFE 영역을 사용하므로, 1바이트 문자와 2바이트 문자의 바이트가 매치되는 경우가 발생하지 않는다. 그러므로 $P[q-k+1..q] = P[1..k]$ 이고 $P[q]$와 $P[k]$가 같은 종류의 바이트일 때, $P[q-k+1]$와 $P[1]$도 같은 종류의 바이트가 된다. 그러므로 위와 같이 접두사 함수를 정의하면, EUC-KR 문자집합을 사용한 패턴이 문자 단위로 이동하게 된다.

개선된 KMP 알고리즘은 검색과정에서는 문자 단위로 텍스트와 패턴을 비교하기 위해서, 텍스트 상에서 현재 비교하려는 바이트 $T[i]$가 1바이트 문자인 경우에는 1바이트만 비교하도록 하고, $T[i]$가 2바이트 문자의 첫 번째 바이트인 경우($T[i] \geq 0x80$)에는 2바이트를 한꺼번에 비교한다.

또한 우리는 위에서 제안한 2가지 알고리즘을 실험하여 속도를 비교하였다. 실험한 환경은 다음과 같다. GNU/LINUX (Fedora Core 11) 2.6.29.6-217.2.3 운영체제와 gcc 4.4.0 컴파일러를 사용하였으며, 2.4GHz Intel Core2 Quad CPU Q6600과 8GB RAM이 장착된 시스템에서 실험을 진행하였다. 실험 데이터는, 한글 웹사이트에서 모은 약 2,000 페이지 중에서 script와 HTML tag 등을 제외한 평문 (약 28MB)을 텍스트로 하고, 문자 길이가 2, 4, 6, 8, 10, 12, 18, 24, 30, 36, 42, 48, 54, 60인 문자열을 텍스트에서 랜덤하게 100개씩 추출하여 패턴으로 사용하였다. 이 100개의 패턴에 대하여 2가지 알고리즘이 각각 사용한 시간의 평균값을 확인한 결과, 개선된 KMP 알고리즘의 속도가 개선된 Naïve 알고리즘에 비해서 약 18% ~ 19% 빠른 것을 확인하였다.

3. 결론

이 논문에서는 EUC-KR 문자집합을 사용한 텍스트 상에서 오검색을 피하고 문자열 완전일치 검색을 할 수 있도록 개선한 KMP 알고리즘을 제안하였다. 일반적으로 ASCII 문자집합 기반의 텍스트 상에서는 Naïve 알고리즘과 KMP 알고리즘의 실제 속도가 거의 차이가 없음을 [3] 에서 확인할 수 있다. 그러나 다중바이트 문자집합 기반의 텍스트 상에서 오검색이 발생하지 않도록 개선한 KMP 알고리즘은 오검색을 제거한 Naïve 알고리즘보다 약 18% ~ 19% 빠른 속도를 보였다.

참고문헌

[1] Vim 편집기, http://www.vim.org

[2] D. E. Knuth, J. H. Morris Jr, and V. R. Pratt. Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6:323-350, 1977

[3] G. De V. Smit. A comparison of three string matching algorithms. *Software: Practice and Experience*, 12(1), pp.57-66, 1982