다중바이트 텍스트에서의 빠른 문자열 알고리즘[1]

김은상○★, 김진욱★★, 박근수★

★서울대학교 컴퓨터공학부, ★★서울대학교병원 의료정보센터

eskim@theory.snu.ac.kr, gnugi@snuh.org, kpark@theory.snu.ac.kr

Fast String Matching Algorithm on Multi-byte Texts

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

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

★★Medical Information Center, Seoul National University Hospital

1. 서론

다중바이트 문자집합은, 문자 1개를 표현하기 위해서 1바이트 또는 2바이트 이상을 사용하는 문자집합을 의미한다. ASCII 문자집합의 경우 1바이트로 문자를 표현할 수 있으나, 한글, 중국어, 일본어 등의 문자집합은 훨씬 많은 문자를 가지고 있으므로 문자 1개를 표현하기 위해서 2바이트 이상이 필요하다.

다중바이트 문자집합 텍스트 상에서 문자열 완전일치 문제를 풀 때 기존의 문자열 검색 알고리즘을 사용하면 오검색이 발생할 가능성이 존재한다. 오검색이란, 실제로 텍스트에 패턴이 존재하지 않으나 검색 알고리즘은 해당 패턴이 텍스트에 존재한다고 출력하는 것을 말한다.

오검색 문제를 해결하기 위해서 실제로 널리 사용되고 있는 공개소스 편집기인 Vim[1]과 Emacs[2]에서는 오검색 문제를 해결하기 위해서 각각 Naïve 알고리즘을 사용하거나, 텍스트와 패턴을 UTF-8 인코딩으로 변환하여 Horspool[3] 알고리즘을 사용하고 있다. 또한, KMP[4] 알고리즘의 접두사 함수를 문자 단위로 구성하여 오검색이 발생하지 않도록 개선한 알고리즘[5]이 최근에 제안되었다.

현재까지의 오검색 문제를 해결하기 위한 알고리즘들은 검색 속도에 초점을 두지 않았으며, 오검색 문제를 해결하기 위해서는 텍스트를 전부 확인해야 하기 때문에 접두사 접근방식 (prefix-based approach) 알고리즘이 선호되었다. 그러나 기존의 문자열 검색 알고리즘 중에서는, 접미사 접근방식 (suffix-based approach)이 접두사 접근방식에 비해 훨씬 더 빠른 속도를 보인다는 것이 알려져 있다.

이 논문에서는 접미사 접근방식 알고리즘 중에서 Sunday 알고리즘을 문자 단위로 개선하여 EUC 인코딩 텍스트 상에서 오검색이 발생하지 않도록 한 알고리즘을 제안한다.

2. 본론

EUC 인코딩은 한글, 중국어, 일본어 등을 다중바이트로 표현하기 위한 인코딩이며, 공통적으로 2바이트 문자를 첫번째 바이트와 두번째 바이트 모두 0xA1 ~ 0xFE의 값으로 표현한다. 패턴이 모두 EUC 인코딩의 2바이트 문자로만 이루어진 경우에 기존의 문자열 검색 알고리즘을 사용하면 오검색이 발생할 수 있다.

먼저 기존의 Sunday 알고리즘을 사용하여 문자열 검색을 할 때 오검색을 피하기 위한 가장 간단한 방법은 패턴이 발견되는 위치에서 텍스트와 패턴이 동기화되었는지 확인하면 된다. 동기화 여부를 확인하기 위해서 패턴이 발견된 위치 T[j]에서부터 왼쪽으로 1바이트 문자나 3바이트 문자 또는 4바이트 문자의 첫번째 바이트가 나올 때까지 텍스트를 살펴본다. 처음으로 발견된 첫번째 바이트를 포함하는 문자의 마지막 바이트의 위치와 T[j]와의 위치의 차를 계산하면 T[j]가 2바이트 문자의 첫번째 바이트인지 두번째 바이트인지 알 수 있으며, T[j]가 첫번째 바이트인 경우에만 동기화가 된 것이다.

그런데 이러한 방법은 텍스트의 거의 대부분이 2바이트 문자로 이루어져 있을 때 동기화를 확인하기

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

위해서 텍스트를 살펴보는 비용이 너무 크게 발생한다는 것이다. 그러므로 이 문제를 해결하기 위해서는 검색 과정을 진행하면서, 패턴을 이동할 때마다 현재까지 가장 우측에 존재하는 어떤 문자의 마지막 바이트 z의 위치를 계속 갱신해야 한다.

한편, 기존의 Sunday 알고리즘의 이동 테이블은 바이트 단위로 구성된다. 그런데 이동 테이블을 문자 단위로 구성할 경우 2가지 장점이 있다. 먼저, 패턴에 동일한 바이트가 존재할 확률보다 동일한 문자가 존재할 확률이 더 적기 때문에 이동 테이블을 바이트 단위로 구성했을 때보다 값을 더 크게 할 수 있다. 다음으로, 문자 단위의 이동 테이블을 사용해서 패턴을 이동할 경우에는 해당 문자의 위치에서 자동으로 동기화가 되기 때문에 패턴이 일치할 경우에 추가로 동기화를 확인할 필요가 없다는 것이다. 그런데 문자 단위로 패턴을 이동할 경우에는, 패턴의 마지막 바이트와 정렬된 텍스트의 바이트가 어떤 문자의 마지막 바이트인지 알아야 하는 단점이 존재한다. 그런데 z값을 계속 갱신하는 과정에서 이를 확인할 수 있다.

문자 단위의 이동 테이블을 구현하기 위해서 3가지 방법을 사용하였다. 첫번째는, 2바이트 문자를 41377 (0xA1A1) ~ 65278 (0xFEFE) 사이의 integer 값으로 매핑하여 문자 1개 당 이동값을 계산한다. 두번째는, 2바이트 문자의 첫번째 바이트와 두번째 바이트 각각의 이동 테이블을 따로 만들어서 패턴의 이동값을 두 값의 최대값으로 계산한다. 세번째는, 두번째 방법에서 하위 바이트 즉, 두번째 바이트로 생성한 이동 테이블만을 사용하는 것이다.

우리는 위에서 제안한 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 알고리즘이 각각 사용한 시간의 평균값을 확인한 결과, 논문에서 제안한 New Sunday 알고리즘의 속도가 Vim, Emacs 알고리즘과 [5]의 알고리즘보다 훨씬 빠른 것을 확인하였다.

3. 결론

이 논문에서는, EUC 인코딩 텍스트 상에서 오검색이 발생하지 않고 문자열 완전일치 검색을 할 수 있도록 개선한 Sunday 알고리즘을 제안하였다. 특히 다중바이트 텍스트 상에서의 문자열 검색 알고리즘은 지금까지 많은 연구가 진행되지 않았으며, 제안된 알고리즘들도 대부분 그 검색 속도에는 초점을 두지 않았다. 논문에서 제안한 New Sunday 알고리즘은 문자 단위의 이동 테이블을 정의하고 이를 적용하여 오검색이 발생하지 않도록 하면서도 널리 사용되고 있는 편집기인 Vim과 Emacs에서 적용한 검색 알고리즘들과 [5]에서 제안한 New KMP 알고리즘보다도 훨씬 빠른 속도를 보였다.

참고문헌

[1] Vim 편집기, http://www.vim.org

[2] Emacs 편집기, http://www.gnu.org/software/emacs

[3] R. Nigel Horspool. Practical Fast Searching in Strings. *Software Practice and Experience*, 10(6):501-506, 1980

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

[5] Eunsang Kim, Jin Wook Kim, and Kunsoo Park. String Matching Algorithm without False Matches for EUC-KR Texts. *Korea Computer Congress 2010*, Vol.37, No.1(A), pp. 319-320, 2010