다중바이트 문자집합 텍스트에서의 문자열 검색 알고리즘

(String Matching Algorithm on Multi-byte Character Set Texts)

김 은 상 † 김 진 욱 ††
(Eunsang Kim) (Jin Wook Kim)

박 근 수 †††
(Kunsoo Park)

요 약 문자열 완전일치 검색 알고리즘은 지금까지 많은 연구가 되어왔지만, EUC-KR 등 다중바이트 문자집합에 대해서는 연구된 것이 부족한 상황이다. 이 논문에서는 기존의 KMP 알고리즘을 사용할 때 EUC-KR과 같은 다중바이트 문자집합 텍스트에서 오검색이 발생할 수 있음을 보이며, 문자 단위의 접두사 함수를 적용하여 오검색이 발생하지 않도록 개선한 KMP 알고리즘을 제안한다. 또한, 널리 사용되고 있는 편집기인 Vim과 Emacs의 검색 알고리즘 및 기존의 오토마타 방식의 연구 결과에 비해 논문에서 제안한 알고리즘이 더 빠른 속도를 보이는 실험 결과를 제시한다.

키워드 : 문자열 완전일치 검색, EUC-KR, 다중바이트 문자집합, 오검색, KMP

Abstract An extensive research on exact string matching has been done, but there have been few researches on the matching in multi-byte character set texts such as EUC-KR. This paper shows that false matches may occur in multi-byte character set texts such as EUC-KR when using KMP algorithm, and presents a refined KMP algorithm without false matches applying a character-based prefix function. And also, Experimental results show that our algorithm is faster than string matching algorithms of widely used editors, Vim and Emacs, and the existing automata-based algorithm.

Key words : Exact string matching, EUC-KR, Multi-byte character set, False match, KMP

· 본 연구는 기초기술연구회의 NAP 과제 지원으로 수행되었습니다. 이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사 드립니다.
· 이 논문은 2010 한국컴퓨터종합학술대회에서 'EUC-KR 텍스트에서 오검색을 제거한 문자열 검색 알고리즘'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서울대학교 컴퓨터공학부
eskim@theory.snu.ac.kr
†† 정 회 원 : 서울대학교병원 의료정보센터 교수
gnugi@snuh.org
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
kpark@theory.snu.ac.kr

논문접수 : 2010년 7월 28일
심사완료 : 2010년 8월 24일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제10호(2010.10)

1. 서 론

문자열 완전일치 검색(exact string matching) 문제는, 유한한 개수의 알파벳 Σ개로 이루어진 텍스트 T와 패턴 P가 주어졌을 때 텍스트 T에서 패턴 P가 존재하는 모든 위치를 찾는 문제이다. 문자열 완전일치 검색을 위해서 많은 알고리즘이 개발되었고, 이 알고리즘들은 3가지 접근방식으로 분류할 수 있다. KMP(Knuth-Morris-Pratt)[1] 알고리즘과 같은 접두사 접근방식(Prefix-based approach), 보이어-무어(Boyer-Moore)[2] 알고리즘과 같은 접미사 접근방식(Suffix-based approach), 그리고 BOM(Backward Oracle Matching) [3] 알고리즘과 같은 요소 접근방식(Factor-based approach)이 그 3가지다.

표준 ASCII 문자집합에서는 영어 알파벳, 숫자, 특수문자 등을 7 비트로 표현 가능하고, 그 외의 서양 언어 특수 문자를 포함하더라도 8 비트로 표현할 수 있다. 그러나 한글, 중국어, 일본어 등은 훨씬 많은 문자(character)를 가지고 있으며, 이러한 언어의 문자 1개를 표현하기 위해서 2바이트(byte) 이상을 사용하게 되었다. 편의를 위해 기존의 ASCII 문자들을 1바이트로 표현하기 때문에, 하나의 텍스트에 1바이트 문자와 다중바이트(multi-byte) 문자가 공존하게 되었다. 예를 들면, EUC-KR(Korean Extended-Unix-Code) 텍스트에서는 1바이트로 표현되는 ASCII 문자들과 2바이트로 표현되는 한글문자가 공존한다.

지금까지의 문자열 완전일치 검색 알고리즘들은 대부분 다중바이트 문자에 대한 고려가 없이 연구되었기 때문에, 다중바이트 문자를 포함한 문자열에 그대로 적용할 수 없는 문제가 있다. EUC-KR 텍스트에서 기존의 문자열 완전일치 검색 알고리즘을 사용했을 때 오검색(false match)이 발생하는 것도 그 중의 한가지이다. 오검색 문제를 해결하기 위해서 다중바이트 문자집합의 DFA(Deterministic Finite Automata)와 Aho-Corasick 오토마타(Aho-Corasick Automata, AC Automata)를 결합한 방법[4]이 제안되었으며, 패턴에 포함된 문자 단위로 bit-vector를 구성하여 bit-parallel shift-and 알

고리즘을 적용한 방법[5]도 연구된 바 있다. 조사한 바에 의하면, 다중바이트 문자집합을 위한 문자열 검색 알고리즘 중에서 문자 단위의 접미사 접근방식과 문자 단위의 요소 접근방식은 제안된 적이 없다. 그리고 실제로 널리 사용되고 있는 공개소스 편집기인 Vim[6]과 Emacs [7]에서는 오검색 문제를 해결하기 위해서 각각 Naïve 알고리즘을 사용하거나, 텍스트와 패턴을 UTF-8 인코딩으로 변환하여 Horspool[8] 알고리즘을 사용하고 있다.

이 논문에서는 처음으로 문자 단위의 접두사 함수를 정의하고, 이를 적용하여 EUC-KR 문자집합 텍스트 상에서 오검색을 피하고 문자열 완전일치 검색을 할 수 있도록 개선한 KMP 알고리즘을 제안한다. 논문의 구조는 다음과 같다. 2장에서는 EUC-KR 문자집합과 오검색에 대해서 간략히 설명한다. 3장에서는 기존의 KMP 알고리즘을 사용했을 때 오검색이 발생할 수 있다는 것을 밝히고, 4장에서 문자 단위의 접두사 함수를 적용하여 오검색이 발생하지 않도록 개선한 KMP 알고리즘을 제안한다. 5장에서는 Vim 편집기에서 사용되고 있는 Naïve 알고리즘과 논문에서 제안한 KMP 알고리즘의 속도를 비교한 실험 결과를 설명하고, 6장에서 결론 짓는다.

2. 배경 지식

2.1 EUC-KR 문자집합

EUC-KR은 KS X 1001 문자집합과 KS X 1003 문자집합을 사용하는, 대표적인 한글 완성형 인코딩 방식이다. 먼저 KS X 1003은 한국산업표준(KS)[9]으로 지정된 "정보 교환용 부호(로마 문자)"이며, ASCII와 거의 동일한 문자집합이지만, 역슬래시(\)가 원화 기호(₩)로 바뀐 것만 다르다. 다음으로 KS X 1001은 한국산업표준으로 지정된 "정보 교환용 부호계(한글 및 한자)"이며, KS X 1001로 번호가 변경되기 전에는 KS C 5601이었다. KS X 1001은 0xA1A1~0xFEFE 사이의 영역에서 한글 2350자, 한자 4888자와 그 외 특수 문자 등을 표현하는 문자집합이다.

즉, EUC-KR은 1바이트 문자집합인 KS X 1003을 0x00~0x7F 사이의 영역에서 표현하며, 2바이트 문자집합인 KS X 1001을 0xA1A1~0xFEFE 영역에서 표현하게 된다. 예를 들어, "abc가나다"라는 문자열은 그림 1과 같은 바이트 순서(byte sequences)를 갖게 된다.

a	b	c	가		나		다	
61	62	63	B0	A1	B3	AA	B4	D9

그림 1 EUC-KR 문자열 바이트 순서 예시

2.2 오검색(False match)

EUC-KR과 같은 다중바이트 문자집합 텍스트 상에서 기존의 문자열 완전일치 검색 알고리즘을 사용하면 오검색(false match)이 발생할 가능성이 있다.

오검색이란, 텍스트 상에서는 실제로 해당 패턴이 존재하지 않지만 검색 알고리즘이 패턴이 존재한다고 결과를 출력하는 것을 말한다. 예를 들면, 텍스트가 '영도'(BFB5 B5B5)이고 '도'(B5B5)라는 문자열을 검색한다고 하자. 1바이트 기반 문자열 검색 알고리즘에서는 텍스트를 BF B5 B5 B5 라는 4개의 바이트로 인식하고, 검색하고자 하는 문자열을 B5 B5라는 2개의 바이트로 인식하기 때문에 그림 2와 같이 오검색을 하게 된다. 패턴의 첫 번째 문자의 첫 번째 바이트와 텍스트의 첫 번째 문자의 두 번째 바이트가 일치하고, 패턴의 첫 번째 문자의 두 번째 바이트와 텍스트의 두 번째 문자의 첫 번째 바이트가 같아서 텍스트의 첫 번째 문자의 두 번째 바이트 위치에서 오검색이 발생한 경우이다.

EUC-KR 인코딩은 2바이트 문자의 첫 번째 바이트와 두 번째 바이트를 모두 A1~FE 영역에서 표현하기 때문에, 패턴의 문자들이 모두 같은 영역에서 표현되는 경우에 패턴과 텍스트를 비교할 때 문자의 경계를 알 수 없게 됨으로 인해 오검색이 발생한다. 반면에, UTF-8[1]과 같이 다중바이트 문자 인코딩의 첫 번째 바이트와 두 번째 이상의 바이트의 영역이 다른 경우에는 오검색이 발생하지 않는다.

영		도	
BF	B5	B5	B5

도	
B5	B5

그림 2 EUC-KR 텍스트 상에서 발생하는 오검색의 예시

다중바이트 문자집합에서 오검색 문제를 해결하기 위한 기존의 연구 결과는 다음과 같다. 먼저 다중바이트 문자집합의 인코딩을 수용하는 DFA와 AC 오토마타를 결합하여 동기화와 검색을 동시에 수행하는 방법[4]이 제안되었다. 또한, 패턴에 포함된 문자 단위로 Bit- vector를 구성하고 패턴에 포함된 문자들의 AC 오토마타를 사용하여 텍스트의 현재 문자가 패턴에 존재하는 문자인지를 검색하는 문자 단위 Bit-parallel Shift-and 알고리즘[5]도 연구되었다. 그리고 Vim 편집기[6]에서는 텍스트의 현재 문자와 패턴의 첫 번째 문자를 비교하는 작업을 계속 진행하다가 서로 같을 경우 C library의 strncmp()함수를 사용하여 패턴의 길이만큼 바이트 순서가 일치하는

1) UTF-8은 1바이트 인코딩에서 4바이트 인코딩까지 있으며, 1바이트 문자부터 4바이트 문자까지의 첫 번째 바이트는 각각 00~7F, C0~DF, E0~EF, F0~F7의 영역을 사용하고, 두 번째 이상의 바이트는 80~BF의 영역을 사용한다.

지 검사하는 Naïve 알고리즘을 사용하고 있다. Emacs 편집기[7]에서는 텍스트와 패턴을 UTF-8 인코딩으로 변환하여 오검색이 발생하지 않도록 한 후에 접미사 접근방식 중의 하나인 Horspool[8] 알고리즘을 사용하고 있다.

이러한 오검색을 피하기 위해서는 기본적으로 패턴과 텍스트가 서로 문자 단위로 동기화(Synchronization)되었는지 확인하는 것이 필요하다. 동기화 여부를 확인하는 간단한 방법으로는, 패턴의 첫 번째 바이트와 비교하는 텍스트 상의 바이트가 어떤 문자의 첫 번째 바이트인지 확인하는 것이다. 그런데 동기화 여부를 확인하기 위해서는 텍스트를 처음부터 차례대로 읽어가면서 텍스트의 현재 바이트가 어떤 문자의 첫 번째 바이트인지 확인해야 하기 때문에 일반적으로 접두사 접근방식의 문자열 검색 알고리즘이 선호된다. 이 논문에서는 기존의 KMP 알고리즘을 문자 단위로 동작하도록 개선하여 동기화와 문자열 검색을 동시에 수행하는 알고리즘을 제안하도록 할 것이다.

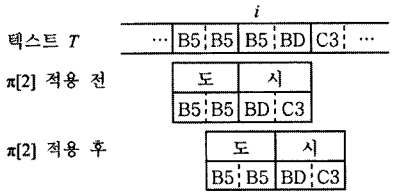
3. 오검색이 발생하는 기존의 KMP 알고리즘

먼저 논문에서 사용되는 기호들의 의미는 다음과 같다. 텍스트 T의 길이를 n, 패턴 P의 길이를 m으로 표기한다. $T[i]$는 텍스트 T의 i번째 바이트를 의미하며, 동일하게 $P[i]$는 패턴 P의 i번째 바이트를 의미한다. 또한 $|S|$가 문자열 S의 바이트 길이이고 $1 \le i \le j \le |S|$인 조건 하에서, $S[i..j]$는 문자열 S의 부분문자열 $S[i]S[i+1]...S[j]$을 의미한다. 그리고 S_i는 문자열 S의 i번째 문자를, $||S||$가 문자열 S의 문자 길이이고 $1 \le i \le j \le ||S||$인 조건 하에서 $S_{i,j}$는 문자열 S의 부분문자열 $S_iS_{i+1}...S_j$를 의미한다. EUC-KR 텍스트에서는 1바이트 문자와 2바이트 문자가 공존하기 때문에, 바이트와 문자의 의미가 서로 다르다. 예를 들면, 그림 1에서 5번째 문자는 "나"이지만, 5번째 바이트는 "A1"이다.

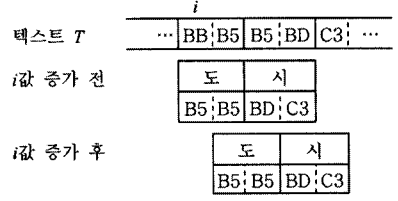
KMP[1] 알고리즘은 접두사 함수(prefix function)를 사용하여 문자열이 존재하는 위치를 검색함으로써, Naïve 알고리즘에서 필요없는 비교 연산을 줄이도록 한 알고리즘이다. 접두사 함수 π는 $\pi : \{1,2,\cdots,m\} \rightarrow \{0,1,\cdots,m-1\}$인 함수이며, $\pi[q] = max\{k : P[1..k]$는 $P[1..q]$의 접미사$\}$로 표현할 수 있다. 즉, π는 $P[1..q]$의 접미사이면서 패턴 P의 가장 긴 접두사의 길이를 의미한다. KMP 알고리즘의 흐름은 전처리(preprocessing) 과정에서 접두사 함수를 만들고, 검색(matching) 과정에서 접두사 함수를 사용하여 문자열이 존재하는 위치를 검색하게 된다. 텍스트와 패턴이 q바이트만큼 일치하고 $q+1$번째 바이트가 서로 다른 바이트일 때(mismatch), 패턴을 $q-\pi[q]$만큼 이동(shift)하면 $\pi[q]$바이트만큼 텍스트와 패턴이 일치하고 $\pi[q]+1$번째 바이트부터 다시 텍스트와 패턴을 비교하게 된다. KMP 알고리즘은 이 작업을 반복하여 검색 과정을 진행한다.

그림 2와 같이 EUC-KR 텍스트 상의 어떤 문자의 두 번째 바이트부터 패턴과 비교하고 그 위치에서 패턴이 존재하는 경우에 오검색이 발생했던 것처럼, KMP 알고리즘에서도 이와 같은 현상이 발생한다. KMP 알고리즘에서 텍스트 상의 어떤 문자의 두 번째 바이트부터 패턴과 비교하게 되는 경우는 다음과 같이 2가지가 있다.

- $\pi[q] = k$, $k > 0$일 때 (즉, $P[1..k] = P[q-k+1..q]$), $P[q-k+1]$이 어떤 문자의 두 번째 바이트인 경우이다. 예를 들어, 패턴이 '도시'(B5B5 BDC3) 라고 하자. 아래의 그림에서처럼 $P[3]$에서 $T[i]$와 미스매치가 발생하면 $\pi[2] = 1$이므로 패턴을 1바이트 이동시킨다. 패턴을 이동시킨 후에 텍스트 $T[i-1]$이 $P[1]$과 정렬(align)되게 된다. 그런데 $T[i-1]$은 어떤 문자의 두 번째 바이트이므로, 텍스트 상의 어떤 문자의 두 번째 바이트부터 패턴과 비교하는 경우가 된다.



- $\pi[q] = 0$이어서 패턴의 첫 번째 바이트부터 텍스트와 비교할 때, 텍스트의 위치가 어떤 문자의 두 번째 바이트인 경우이다. 예를 들면, 패턴이 '도시'(B5B5 BDC3)라고 하자. 텍스트와 패턴이 아래의 그림과 같을 때, $P[1]$과 $T[i]$에서 미스매치가 발생하고, 텍스트의 위치를 1 증가시켜서 $T[i+1]$부터 패턴과 비교하게 된다. 그런데 $T[i+1]$은 어떤 문자의 2번째 바이트이므로, 텍스트 상의 어떤 문자의 두 번째 바이트부터 패턴과 비교하는 경우가 된다.



위의 2가지 경우에 KMP 알고리즘에서도 오검색이 발생하며, 이를 피하기 위해서 동기화 작업이 추가로 필요하다.

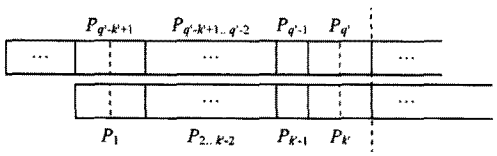
4. 오검색이 발생하지 않는 문자단위 KMP 알고리즘

이전 장에서 살펴보았듯이, 기존의 KMP 알고리즘을 사용하는 경우에는 오검색이 발생한다. 그러나 KMP 알고리즘을 바이트 단위가 아닌 문자 단위로 검색을 하게 된다면, 이전 장에서 살펴보았던 오검색이 발생하는 상황을 피할 수 있게 되며 동기화를 위해서 텍스트의 현

재 바이트가 어떤 문자의 첫 번째 바이트인지 확인해야 하는 작업이 불필요해진다.

이번 장에서는 오검색이 발생하지 않도록 문자 단위로 접두사 함수를 구성하고 검색 과정을 진행하는 KMP 알고리즘을 제안한다. 문자 단위의 접두사 함수는 이 논문에서 처음으로 제안되는 것이다.

먼저, 문자 단위의 접두사 함수 π'은 $\pi'[q'] = max\{k' : P_{1..k'}$ 은 $P_{1..q'}$의 접미사$\}$로 정의할 수 있다. 이를 도식화하면 아래와 같으며, 문자별로 $P_{q'-k'+1} = P_1$, $P_{q'-k'+2} = P_2$, $\cdots$, $P_{q'} = P_{k'}$이 된다. 이전 장에서 설명한 기존의 KMP 알고리즘의 접두사 함수는 바이트 단위로 정의되지만, 우리가 제안하는 새로운 접두사 함수는 문자 단위가 기본이 된다.



그런데 위에서 정의한 문자 단위의 접두사 함수를 실제로 구현하기 위해서는 바이트 단위로 수정할 필요가 있다.

EUC-KR 텍스트에 적용하기 위해서 문자 단위의 접두사 함수를 바이트 단위로 수정해보면, 접두사 함수 π는 $\pi[q] = max\{k : P[1..k]$는 $P[1..q]$의 접미사이고, $P[q]$와 $P[k]$가 모두 1바이트 문자이거나 2바이트 문자의 두 번째 바이트$\}$로 정의할 수 있다. EUC-KR 문자 집합은 1바이트 문자와 2바이트 문자의 바이트가 서로 다른 영역을 사용하는 특성을 가지고 있기 때문에, 1바이트 문자가 2바이트 문자의 바이트와 매치되는 경우가 발생하지 않는다. 그러므로 $P[q-k+1..q] = P[1..k]$이고 $P[q]$와 $P[k]$가 같은 종류의 바이트일 때 아래의 그림과 같이 마지막 문자인 $P_{q'}$과 $P_{k'}$은 서로 같은 문자가 된다. 또한 그 앞의 문자들도 자동적으로 매치가 된다. ($P_{q'-k'+1} = P_1$, $P_{q'-k'+2} = P_2$, $\cdots$, $P_{q'-1} = P_{k'-1}$) 그러므로 위와 같이 접두사 함수를 수정하면, EUC-KR 패턴 상에서 문자 단위 접두사 함수와 동일한 값을 갖게 된다.

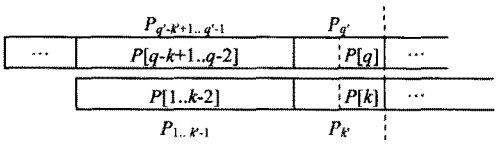


그림 3에서, 개선된 KMP 알고리즘의 전처리 과정을 통해 생성된 패턴 '도도돈'(B5B5 B5B5 B5b7)의 접두사 함수를 확인할 수 있다.

검색 과정에서는 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 경우에

		도		도		돈	
		B5	B5	B5	B5	B5	B7
기존의 KMP 알고리즘	i	1	2	3	4	5	6
	$\pi[i]$	0	1	2	3	4	0
개선된 KMP 알고리즘	i	1	2	3	4	5	6
	$\pi[i]$	0	0	0	2	0	0

그림 3 기존의 KMP 알고리즘과 개선된 KMP 알고리즘의 접두사 함수 비교

는 2바이트($T[i]$와 $T[i+1]$)를 한꺼번에 패턴과 비교한다. EUC-KR의 1바이트 문자는 0x00~0x7F 사이에서 정의되므로, $T[i] \geq$ 0x80인지를 확인하면 $T[i]$가 2바이트 문자의 첫 번째 바이트인지 알 수 있다.

5. 실험 결과

논문에서 제안하고 있는 알고리즘, EUC-KR DFA와 오토마타를 결합한 PMM(Pattern Matching Machine) [4] 및 Vim 편집기, Emacs 편집기의 검색 알고리즘 속도를 비교하는 실험을 진행하였다. 4가지 알고리즘 모두 직접 구현하였으며, 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.

Vim과 Emacs의 검색 알고리즘은 다음과 같이 구현하였다. 먼저 Vim 편집기의 알고리즘은 텍스트와 패턴의 첫 번째 문자를 비교하고 서로 일치한 경우에 C library의 strncmp() 함수를 사용하여 검색을 진행한다. EUC-KR 텍스트 상에서 이와 동일한 동작을 하기 위해 Naïve 알고리즘을 수정하였다. 수정한 사항은, 현재 텍스트의 문자가 2바이트 문자인 경우에 다음 비교할 위치를 2바이트만큼 이동하여 다음 문자부터 비교 작업을 반복하도록 한 것이다. 다음으로 Emacs 편집기의 검색 알고리즘은 텍스트와 패턴을 UTF-8 인코딩으로 변환한 후 Horspool 알고리즘을 적용하는 것이다. 이를 구현하기 위해서 Emacs 편집기 코드에 포함되어 있는 EUC-KR to UTF-8 매핑 파일을 사용하여 EUC-KR 텍스트를 UTF-8 인코딩으로 변환하였고, Horspool 알고리즘보다 개선된 Sunday[10] 알고리즘을 적용하였다. 그리고 PMM 알고리즘은 기본적으로 AC 오토마타를 사용한 다중패턴(multi-pattern) 검색 알고리즘이지만, 다른 알고리즘과의 비교를 위해서 매번 1개의 패턴만으로 AC 오토마타를 구성하여 실험을 진행하였다.

실험 데이터는, 한글 웹사이트에서 모은 약 2000 페이지 중에서 script와 HTML tag등을 제외한 평문(약 28MB)을 텍스트로 하고, 바이트 길이가 2, 4, 6, 8, 10,

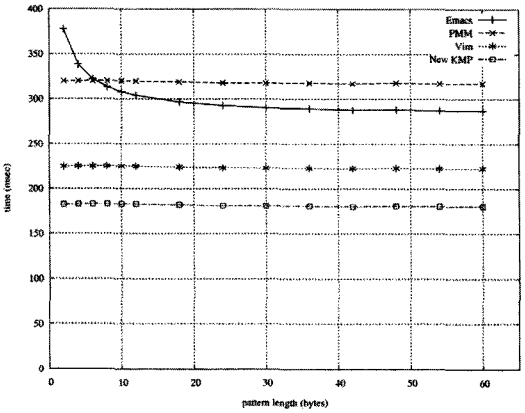


그림 4 문자열 완전일치 검색 속도 비교

12, 18, 24, 30, 36, 42, 48, 54, 60인 문자열을 텍스트에서 랜덤하게 100개씩 추출하여 패턴으로 사용하였다. 14가지 길이의 패턴에 대하여 4가지 알고리즘이 각각 사용한 시간의 평균값을 그림 4에 나타내었다. 논문에서 제안하고 있는 알고리즘을 New KMP라고 표기하였으며, Vim 편집기에서 사용하고 있는 Naïve 알고리즘을 Vim[4]의 알고리즘을 PMM, Emacs 편집기의 방법을 Emacs라고 표기하였다.

4가지 알고리즘 중에서 New KMP 알고리즘이 속도가 가장 빠른 것을 확인할 수 있으며, Vim의 Naïve 알고리즘에 비해서 약 20% 빠른 속도를 보였다. 한편, PMM 알고리즘의 속도가 느린 것은 오토마타를 계속해서 확인하고 그 state를 유지 및 변화시켜가는 추가비용으로 인한 것이며, Emacs의 알고리즘의 경우, EUC-KR 텍스트를 UTF-8 인코딩으로 변환하는 시간과 검색 시간을 합한 결과가 New KMP 알고리즘보다 느린 것을 확인할 수 있었다.

5. 결 론

이 논문에서는, 처음으로 문자 단위의 접두사 함수를 정의하고, 이를 적용하여 EUC-KR 문자집합 텍스트 상에서 오검색을 피하고 문자열 완전일치 검색을 할 수 있도록 개선한 KMP 알고리즘을 제안하였다. 논문에서 제안한 New KMP 알고리즘은 널리 사용되고 있는 편집기인 Vim과 Emacs에서 적용한 검색 알고리즘들과 오토마타 방식의 기존 연구 결과보다 훨씬 빠른 실험 결과를 보였다. 일반적으로 ASCII 문자집합 기반의 텍스트 상에서 바이트 단위의 Naïve 알고리즘과 KMP 알고리즘의 실제 속도는 거의 차이가 없다[11]. 그러나 다중바이트 문자집합 기반의 텍스트 상에서 오검색이 발생하지 않도록 개선한 New KMP 알고리즘은 Vim 편집기에서 사용되는 Naïve 알고리즘보다 약 20% 빠른 속도를 보였다. 논문에서는 EUC-KR 인코딩 텍스트를 주 대상으로 하였지만, 오검색이 발생할 수 있는 다른 언어의 EUC(Extended-Unix-Code) 인코딩 텍스트에서도 논문에서 제안한 문자 단위의 접두사 함수를 해당 인코딩의 특성에 따라 수정하여 적용할 수 있을 것이다.

참 고 문 헌

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

[2] Robert S. Boyer and J. Strother Moore. A Fast String Searching Algorithm. *Communications of the ACM*, vol.20, no.10, pp.762-772, 1977.

[3] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Efficient experimental string matching by weak factor recognition. *12th Annual Symposium on Combinatorial Pattern Matching*, Lecture Notes in Computer Science, vol.2089, pp.51-72, 2001.

[4] Masayuki Takeda, Satoru Miyamoto, Takuya Kida, Ayumi Shinohara, Shuichi Fukamachi, Takeshi Shinohara, and Setsuo Arikawa. Processing Text Files as Is: Pattern matching over Compressed Texts, Multi-byte Character Texts, and Semi- structured Texts. *String Processing and Information Retrieval (SPIRE) 2002*, LNCS 2476, pp.170-186, 2002.

[5] Heikki Hyyrö, Jun Takaba, Ayumi Shinohara, and Masayuki Takeda. On Bit-parallel Processing of Multi-byte Text. *Proceedings of the 1st Asia Information Retrieval Symposium (AIRS) 2004*, LNCS 3411, pp.289-300, 2005.

[6] Vim Editor, http://www.vim.org

[7] GNU Emacs Editor, http://www.gnu.org/software/ emacs

[8] R. Nigel Horspool. Practical Fast Searching in Strings. *Software Practice and Experience*, vol.10, no.6, pp.501-506, 1980.

[9] Korean Industrial Standards, http://www.standard.go.kr

[10] Daniel M. Sunday. A Very Fast Substring Search Algorithm. *Communications of the ACM*, vol.33, no.8, pp.132-142, 1990

[11] G. De V. Smit. A comparison of three string matching algorithms. *Software: Practice and Experience*, vol.12, no.1, pp.57-66, 1982.