k개의 오차를 허용하는 순위 패턴 매칭

(Order preserving matching with k mismatches)

이 인 복*
(Inbok Lee)

요 약

순위 패턴 매칭 문제는 패턴과 텍스트가 주어졌을 때, 텍스트의 부분 문자열 중 패턴과 순위 동형을 만족하는 것들을 찾는 문제이다. 이 논문에서는 순위 패턴 매칭에 k개의 오차를 허용하는 문제를 푸는 알고리즘을 제안한다. 제안하는 알고리즘은 기존의 알고리즘에 비하여 간단하고 구현이 쉬우며, 평균적인 경우 선형 시간 복잡도를 가진다. 또한 실험을 통해서, 제안된 알고리즘이 현실적인 데이터에 대해서 효율적으로 동작함을 보인다.

■ 중심어 : 문자열 매칭 ; 근사 패턴 매칭 ; 시계열 데이터 분석

Abstract

Order preserving matching refers to the problem of reporting substrings of a given text where there exists order isomorphism with the pattern. In this paper, we propose a new algorithm based on filtering and evaluation. The proposed algorithm is simple and easy to implement, and runs in linear time on average. Experimental results show that it works efficiently with real world data.

■ keywords : string matching ; approximate pattern matching ; time series data analysis

Ⅰ. 서 론

일반적인 패턴 매칭 문제는 텍스트 $T[1,n]$과 패턴 $P[1,m]$이 주어졌을 때, T의 부분문자열 중 P와 일치하는 부분 문자열의 위치를 찾는 문제이다. 최근 이 문제의 변형으로 순위 패턴 매칭 문제(Order-Preserving Pattern Matching Problem)가 연구되고 있다. 순위 패턴 매칭은 T에서 P와 서로 대응하는 문자에 대해 순위 동형(Order-isomorphic)을 만족하는 T의 부분 문자열의 위치를 찾는 문제이다. 순위 동형이란 P와 길이가 같고 상대적 순서가 같은 T의 부분 문자열을 의미한다. 순위 패턴 문제의 정의는 다음과 같다.

문제 1 : 정수 알파벳으로 이루어진 문자열인 텍스트 $T[1,n]$과 패턴 $P[1,m]$이 주어졌을 때, P의 각각의 값을 정렬해서 $P[\pi_1] < P[\pi_2] < ... < P[\pi_m]$인 $\pi = (\pi_1, \pi_2, ..., \pi_m)$ 순열을 구하자. 길이 m인 T의 부분문자열 $T[i,i+m-1]$이 P와 순위 패턴 매칭이 존재하는 경우는

$T[i+\pi_1-1] < T[i+\pi_2-1] < \cdots < T[i+\pi_m-1]$

일 때이다. 이러한 i를 모두 구하시오.

예제 1: $T = (4,20,16,3,21,10,8,22,2,16,23,9)$, $P = (20,34,27,25)$ 일 때, $\pi = (1,4,3,2)$가 된다. 이 예제에서 $T[4,7] = (3,21,10,8)$이 순위 패턴 매칭을 만족한다. (그림 1)

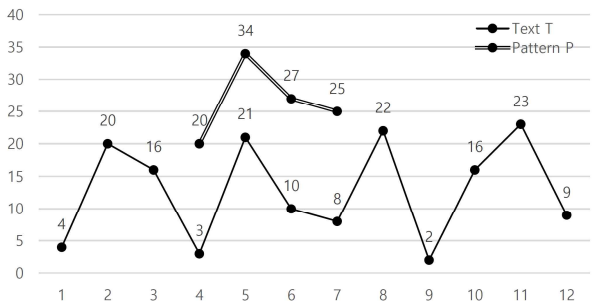

그림 1. 순위 패턴 매칭의 예

* 정회원, 한국항공대학교 소프트웨어학과 교수

접수일자 : 2020년 06월 04일
게재확정일 : 2020년 06월 11일
교신저자 : 이인복, e-mail : inboklee@kau.ac.kr

이 문제는 주식 가격이나, 음악 데이터의 표절여부 검사와 같은 시계열 데이터 분석[1-3]에 사용될 수 있으며, 처음 [4]에서 순위 패턴 매칭(Order-preserving pattern matching)이라는 이름으로 정의되었다. [4]에서는 단일 패턴에 대해 KMP (Knuth-Morris-Pratt) 알고리즘의 변형인 $O(n+m\log m)$ 시간 알고리즘을, 다중 패턴에 대해 Aho-Corasick 오토마톤을 이용한 $O(nm\log m)$ 알고리즘을 제시했다. [5]에서 또한 가장 가까운 값의 인덱스를 이용한 KMP 접근으로 $O(n+sort(m))$시간에 순위 패턴 매칭을 풀 수 있음을 보였다. $sort(m)$은 m개의 정수를 정렬하는데 걸리는 시간이다. [6]에서는 보이어-무어(Boyer-Moore) 알고리즘에 기반한 방법을 제시했다. [7]에서는 단순한 문자열이 아닌 트리와 DAG(Directed Acyclic Graph)에서의 순위 패턴 매칭을 연구했다.

[8, 9]에서는 답의 후보를 먼저 찾고 후보만 검증함으로써, 평균 시간 복잡도를 줄였는데, [8]에서는 인접한 문자의 대소를 0과 1로 표현하여 비트연산을 통해 필터링하는 방법을 제시했다. [9]에서는 최소값(또는 최대값)을 이용한 필터링 방법을 제시한다.

순위 패턴 매칭에 k개의 오차를 허용하는 문제는, 주어진 텍스트 $T[1,n]$과 패턴 $P[1,m]$에서 T의 부분 문자열 중 P와 길이가 같은 부분문자열 T'중 P와 k-동형(k-isomorphic)을 만족하는 것들을 찾는 문제이다. k-동형은 T'와 P에서 각각 같은 위치에 있는 k개의 문자들을 삭제했을 때, 결과인 문자열들이 순위 동형인 경우를 말한다.

예제 2: $T=(1,10,6,4,8,5,7,9,3)$, $P=(1,4,2,5,11)$에서 $T[4,8]=(4,8,5,7,9)$와 P는 서로 순위 동형이 아니지만, 이 둘은 1-순위 동형인데 ($k=1$) 각각 네 번째 원소를 제거하면 $(4,8,5,9)$과 $(1,4,2,11)$가 되어 순위 동형이기 때문이다(그림 2).

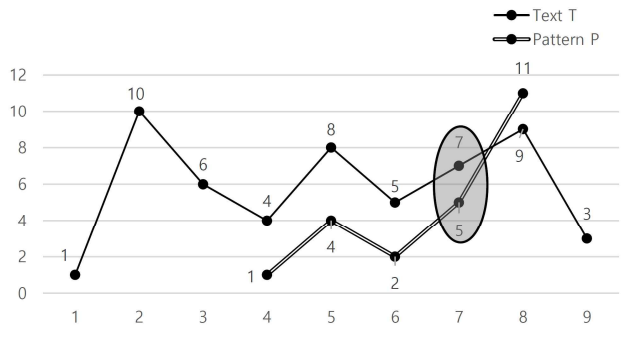

그림 2. k=1인 순위 패턴 매칭의 예

문제 2 : 문제 1과 같이 정수 알파벳으로 이루어진 문자열 텍스트 $T[1,n]$과 패턴 $P[1,m]$이 주어졌다고 하자. 1 이상 m 이하인 서로 다른 수 k개를 골라서, $T[i,i+m-1]$과 P에서 해당하는 위치의 문자를 제거하여 만든 길이 $m-k$인 문자열을 T'', P'이라고 하자. 이 둘이 순위 동형이라면, $T[i,i+m-1]$과 P는 $k-$동형이다. 이러한 i를 모두 구하시오.

Gawrychowski 등은 [10]에서 처음으로 k개의 오차를 허용하는 순위 패턴 매칭 문제에 대한 해법을 제시했다. 이 방법은 여과와 검증 과정으로 이루어지는데, 여과 과정은 vEB (van Emde Boas) 트리를 이용하여 각 문자와 그 문자의 알파벳 순서로 직전인 문자의 인덱스의 위치의 차이를 저장한 시그니처를 만든다. 부분 문자열 $T[i,i+m-1]$와 P가 k개 이하의 오차가 있다면, $T[i,i+m-1]$와 P의 시그니처는 $3k$이하의 오차가 있다는 성질을 이용한다. 이 과정에는 $O(n\log\log m)$ 의 시간이 소요된다. 검증 과정에서는 [12]에서 소개된 가중 LIS 알고리즘인 HIS (Heaviest increasing Subsequence)에 적용할 수 있는 문제로 축소시켜 $O(k\log\log k)$ 시간에 검증을 진행할 수 있음을 보였다. 전체 시간 복잡도는 따라서 $O(n(\log\log m+k\log\log k))$이다.

위에서 보인 [10]의 해법은 이론적으로 매우 빠르지만, 복잡한 전처리 과정을 거치며, 문제를 HIS 문제로 축소하기 위해 많은 처리를 수행한다. 또한 실제 데이터를 통한 실험으로 알고리즘의 성능 평가를 시행한 결과가 없다. 본 논문에서는 따라서 실제 데이터에 적용할 수 있으며, [10]보다 쉽게 구현할 수 있으며 높은 성능을 보이는 알고리즘을 제시하고, 실험을 통하여 효율성을 검증한다.

Ⅱ. 본 론

1. 알고리즘

가. 용어 정의

문자열에서 사용할 문자들은 크기 비교가 가능한 정수라고 가정하자. 1부터 m 사이의 서로 다른 정수의 순열 $\pi=(\pi_1,\pi_2,\ldots,\pi_m)$은 패턴의 각각 위치 글자들이 $P[\pi_1]<P[\pi_2]<\ldots<P[\pi_m]$를 만족하는 순열이다. 패턴 P를 $k+1$개의 동일한 크기로 나눈 조각의 길이를 $\ell(=m/(k+1))$라고 하자. 편의상 ℓ은 $k+1$의 배수로 가정한다. 또, T의 부분 문자열 $T[i,i+\ell-1]$의 최소값의 위치를 $Min(T[i])$로, 최대값의 위치를 $Max(T[i])$라고 하자.

나. 알고리즘 개요

본 논문에서 제시하는 알고리즘은 [10]의 아이디어와 마찬가지로 여과 단계와 검증 단계로 이루어진다.

먼저 검증 과정은 [10]과 같이 최장 증가 부분 서열(Longest Increasing Subsequence, LIS) 기반 알고리즘으로 수행한다. 패턴 P에서 π를 구하려면 P의 글자들을 모두 정렬해야 하므로 $O(m\log m)$의 시간이 걸린다. 텍스트 T의 부분 문자열 $T[i,i+m-1]$을 π를 이용하여 새로운 문자열 $T[i+\pi_1-1]T[i+\pi_2-1]\cdots T[i+\pi_m-1]$을 만들고, 이 문자열의 LIS를 구한다. 이 값이 $m-k$ 이상이라면 $T[i,i+m-1]$와 P 사이에 k개의 오차를 허용하는 순위 매칭이 존재한다. 길이 m인 문자열의 LIS를 구하는데 $O(m\log m)$ 시간이 필요하기 때문에, 이 방법을 T의 길이 m인 모든 부분 문자열에 대해 적용하면 $O(nm\log m)$ 시간이 걸린다. 본 논문에서 제시하는 여과 방법을 사용하면, 생략가능한 시작점을 찾아 제거하고 그렇지 않은 부분에서만 검증을 수행함으로써, 평균 수행 시간을 단축시킨다.

여과 과정에서는 텍스트 T의 부분 문자열 $T[i,i+m-1]$와 패턴 $P[1,m]$을 각각 $k+1$ 개의 동일한 길이 ℓ인 조각으로 나누고, 각 조각이 포함하고 있는 글자들의 최대값과 최소값을 구한다. 이 둘의 위치를 이용하여 여과 여부를 결정한다. 둘을 모두 이용하는 이유는 실험을 통해서, 최대값/최소값 중 하나만을 사용한 경우보다 둘을 모두 사용한 경우가 성능이 좋기 때문이다.

최대값/최소값을 구하는 것은 RMQ (Range Minimum Query)를 이용한다. 본 연구에서는 두 가지 방법을 사용하였는데, 하나는 평균적으로 $O(n)$, 최악의 경우 $O(n\ell)$ 시간이 걸리는 간단한 방법이며, 다른 하나는 최악의 경우에도 $O(n)$을 보장하는 양방향 큐를 이용한 방법이다.

본 논문에서 제시하는 여과 방법의 근거는, 주어진 문자열을 $k+1$개의 동일한 크기로 나누었다면, 가능한 오류의 개수가 k이기 때문에 비둘기집 원리에 의해서 최소한 한 개의 조각은 오류를 포함하지 않기 때문이다.

실험에서는 최소값/최대값을 모두 이용하였지만, 설명의 편의를 위해서 최소값만을 이용하여 여과하는 방법을 설명한다. 최대값은 최소값과 유사하게 구할 수 있으며, 둘 모두 사용하여 여과하는 것도 비슷하게 진행할 수 있다. P와 $T'=T[i,i+m-1]$을 $k+1$개의 길이 ℓ인 조각으로 나누자. P와 T'의 j번째 조각을 $F(P)_j$, $F(T')_j$로 표현하자.

관찰 1: $1\le j\le k+1$인 모든 j에 대해서 $F(P)_j[\pi_1]\neq F(T')_j[\pi_1]$이면, P와 T' 사이에는 최소 $k+1$개의 오차가 존재한다.

관찰 1에 대한 증명은 간단하다. 각각의 조각에서 가장 작은 원소의 위치가 다르기 때문에, 최소한 1개의 오차가 각각의 조각에 존재하고 조각의 개수가 $k+1$이기 때문이다.

중요한 것은 관찰 1의 대우인데, P와 T'가 $k-$순위 동형이라면, $1\le j\le k+1$일 때 $F(P)_j[\pi_1]=F(T')_j[\pi_1]$인 j가 최소한 하나 존재해야 하기 때문이다. 즉 P와 T'에서 최소값의 위치가 같은 조각이 한 개라도 존재하면 답의 후보가 될 수 있다. 최대값의 경우도 비슷하게 증명할 수 있다.

다. 전처리 단계

위에서 설명한 최소값/최대값을 이용한 여과를 구현하기 위해서는 길이 ℓ인 T의 모든 부분 문자열에 대해서 최소값, 최대값의 위치를 구해야 한다. 단순히 구현하면 $O(n\ell)$ 시간이 걸리게 된다. 이를 개선하기 위한 두 가지 방법을 생각해보자.

먼저 우리는 슬라이딩 윈도우에 기반한 평균적으로 $O(n)$, 최악의 경우 $O(n\ell)$ 시간이 걸리는 방법을 고려하려 한다. 이 방법에서는 길이 ℓ인 윈도우를 T의 왼쪽부터 오른쪽으로 이동하면서 윈도우의 최소값을 구한다. 먼저 $T[1,\ell]$의 최소값을 $O(\ell)$ 시간에 구하자. 이제부터는 현재 윈도우의 바로 오른쪽에 오는 숫자를 윈도우에 넣고, 현재 윈도우의 가장 왼쪽 숫자를 윈도우에서 제거하는 식으로 윈도우를 이동한다. 이 두 가지 경우에 최소값이 변하는지를 확인하면 된다. 새로 들어온 숫자가 현재까지의 최소값보다 작거나, 제거된 숫자가 최소값인 경우가 이 경우에 해당된다.

$T[i,i+\ell-1]$의 최소값의 위치를 알고 있다고 가정하자. 이제 $T[i+1,i+\ell]$의 최소값의 위치를 구하려고 한다.

경우 1 $Min(T[i])>T[i+\ell]$:
$Min(T[i+1])=i+\ell$이 된다.
경우 2, $Min(T[i])=i$:
$T[i+1,i+\ell]$을 모두 조사하여 최소값을 구한다.
경우 3, 그 외의 경우:
$Min(T[i+1])=Min(T[i])$이다.

경우 1, 3인 경우에는 $O(1)$ 시간이 걸리며, 경우 2인 경우에는 $O(\ell)$ 시간이 걸린다. 최악의 경우에는 $O(n\ell)$ 시간이 걸리지만, 경우 2의 확률이 $1/\ell$이기 때문에 평균적으로 $O(1)$ 시간이 걸리며, 기댓값의 선형성(linearity of expectation)에 의해서 전체의 평균 시간 복잡도는 $O(n)$이다.

[9]에서 소개한 다른 방법은, 양방향 큐(deque)를 이용하여

최악의 경우 $O(n)$ 시간에 RMQ 문제를 풀 수 있다. 절차는 다음과 같다.

1. 먼저 비어 있는 양방향 큐에 $T[1]$을 삽입한다.
2. $2 \le i \le l$인 $T[i]$와 양방향 큐의 첫 원소를 비교한다. $T[i]$가 더 작다면 양방향 큐를 모두 비우고 $T[i]$를 삽입한다. 그렇지 않다면, $T[i]$를 양방향 큐의 마지막 원소로 삽입한다. $T[i]$의 처리가 끝났을 때, 양방향 큐의 첫 번째 원소의 위치가 $Min(T[1])$이 된다.
3. $\ell+1 \le i \le n$인 $T[i]$에 대해서, 먼저 양방향 큐의 첫 번째 원소가 윈도우 밖인지를 확인한다. 만약 윈도우를 벗어났다면 이 원소를 제거한다. 이 과정을 양방향 큐의 원소에 대해서 반복한다. $T[i]$의 처리가 끝나면, (양방향 큐의 첫 번째 원소의 위치) - (윈도우의 시작점) + 1의 값이 $Min(T[i-\ell+1])$이 된다.

위의 방법이 최악의 경우에 대하여도 $O(n)$ 시간이 걸림을 쉽게 보일 수 있다. 모든 $T[i]$는 양방향 큐에 한 번만 삽입되고, 일단 제거되면 다시 고려되지 않기 때문이다.

라. 여과 단계

전처리 단계에서 $T[1,n]$의 모든 길이가 ℓ인 부분문자열에 대해서, 최소값의 위치 $Min(T[i])$와 최대값의 위치 $Max(T[i])$를 구하였다. 이 값들과 P를 길이가 ℓ인 조각들에 대해서 구한 최소값/최대값의 위치를 비교하면 $T[i,i+m-1]$과 P 사이에 순위 매칭이 존재할 수 있는지 여부를 판정할 수 있다.

$1 \le j \le k+1$인 모든 j에 대해서 다음 식이 참인지를 확인한다: $F(P)_j[\pi_1] = F(T[i,i+m-1])_j[\pi_1]$. 이 중 하나라도 조건을 만족하면 $T[i,i+m-1]$과 P에 대해서 검증을 수행한다. 다음 $F(T[i+1,i+m])_j[\pi_1]$ 을 구해야 하는데, 이는 전처리에서 구한 값을 이용하면 $F(T[i+1,i+m-1])_j[\pi_1] = Min(T[i+1+(j-1)\ell])$임을 알 수 있다. 최대값의 경우도 비슷하게 구할 수 있다.

예제 3. $T=(1,7,6,8,2,9,4,3,5,10)$, $k=1$, $\pi(P)=(4,3,1,2,6,5)$인 경우의 예제가 그림 3, 그림 4이다. $k+1=2$이므로, $l=m/(k+1)=3$이다.

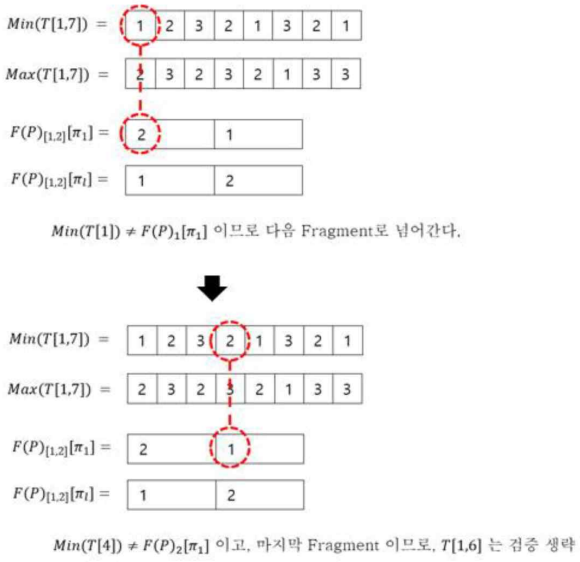

그림 3. T[1, 6]에서 여과 가능 여부 검사

그림 3을 참조하자. $T[1,6]$과 P를 비교하려면, $T[1,3]$의 최소값의 위치는 1, $P[1,3]$의 최소값의 위치는 2이므로 서로 다르다. $T[4,6]$의 최소값의 위치는 2, $P[4,6]$의 최소값의 위치는 1이므로 역시 서로 달라서, $T[1,6]$과 P는 검증이 필요없이 1-동형이 아님을 알 수 있다. 이제 그림 4를 참조하여 $T[2,7]$과 P를 비교해보자. $T[2,4]$와 $P[1,3]$은 최소값의 위치가 일치하지만 최대값의 위치는 일치하지 않는다. $T[5,7]$과 $P[4,6]$의 최소값, 최대값의 위치는 일치하기 때문에, 1개의 조각에 대해서 최소값, 최대값이 모두 일치하므로 $T[2,7]$과 P는 검증 단계로 진행한다.

마. 시간 복잡도 분석

p를 여과 단계에서 부분 문자열이 걸러질 확률 (검증을 수행하지 않을 확률), $q=1-p$를 검증을 수행할 확률이라고 하자. 최소값, 최대값을 모두 사용하여 여과를 수행할 때, 적어도 한 쌍의 조각이 최소값, 최대값이 모두 일치할 확률은 $\ell \cdot (\ell-2)!/\ell! = 1/(\ell-1)$이 된다. 이를 그래프로 그리면 그림 5와 같고, 약간만 패턴이 길어지더라도 이 확률은 매우 작아짐을 볼 수 있다. 따라서, $O(m \log m)$ 시간이 걸리는 검증 단계에 진입할 확률이 적어지고 효율적인 여과 과정을 수행하게 된다.

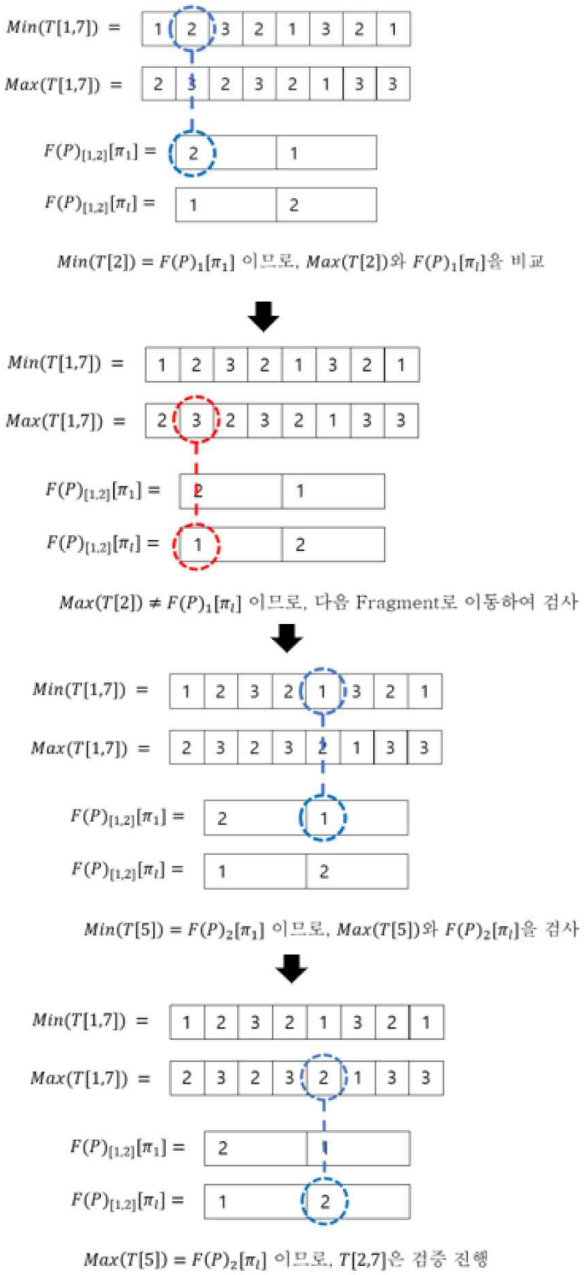

그림 4. T[2, 7]에서 여과 가능 여부 검사

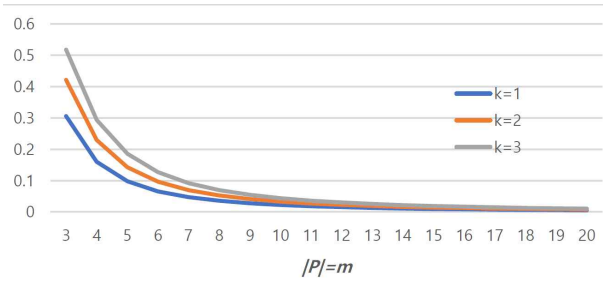

그림 5. 패턴의 길이에 따른 여과를 통과할 확률

III. 실험 결과

1. 실험 환경

실험은 Ubuntu 18.04.3 LTS OS환경에서 CPU는 인텔 i7-6700 3.40GHz에 16GB RAM의 컴퓨터에서 C++로 구현하였다. 여과 단계 없이 검증 단계만 실행한 브루트 포스, 슬라이딩 윈도우, 양방향 큐를 이용한 세 가지 경우를 길이 10^6의 랜덤 텍스트에서 실험하였다. 표 1은 $k=1$, 표 2는 $k=2$, 표 3는 $k=3$인 경우에 대한 실행 시간을 나타낸 표이다.

표 1. k=1, n=1000000인 경우 수행 시간

	$k=1$						
	$m=6$	$m=8$	$m=12$	$m=18$	$m=24$	$m=30$	$m=36$
Bruteforce	10.45	12.72	17.23	24.25	31.01	38.22	46.11
Sliding Win.	5.01	3.71	2.82	2.3	2.14	1.99	1.9
Deque	6.46	5.32	4.51	4.11	3.94	3.83	3.81

표 2. k=2, n=1000000인 경우 수행 시간

	$k=2$					
	$m=6$	$m=12$	$m=18$	$m=24$	$m=30$	$m=36$
Bruteforce	10.66	17.51	23.99	30.9	38.25	46.25
Sliding Win.	11.16	5.8	4.13	3.37	2.98	2.57
Deque	12.76	7.35	5.81	5.16	4.78	4.45

표 3. k=3, n=1000000인 경우 수행 시간

	$k=3$			
	$m=8$	$m=12$	$m=24$	$m=36$
Bruteforce	13.68	17.76	31.03	46.05
Sliding Win.	13.71	10.76	5.94	4.48
Deque	16.33	12.23	7.52	6.17

이론적으로는 슬라이딩 윈도우를 사용한 경우가 양방향 큐를 이용한 경우보다 시간 복잡도가 좋지 않지만, 실제 실험 결과에서는 더 좋은 성능을 보였다. 그 원인은 랜덤 데이터에서 최악의 경우에 해당하는 경우의 수가 적고, 짧은 길이의 패턴에 대해 양방향 큐를 유지하는 부담이 크기 때문으로 보인다.

표 2와 표 3에서 $m=6$인 경우 브루트 포스 방법이 가장 빠른데, 이 경우는 답의 후보가 너무 많기 때문에 여과 단계를 통해 줄어드는 시간보다 전처리의 계산 시간이 많아지기 때문이다. 반면 $m \geq 8$부터는 브루트 포스에 비해 제안한 알고리즘의 성능이 우월함을 알 수 있다.

IV. 결 론

본 논문에서는 k개의 오차를 허용하는 순위 패턴 매칭에서 길이 m인 패턴을 $k+1$개의 동일한 길이의 조각으로 나누어,

각 조각의 최소값/최대값 위치를 비교하여 여과를 수행하여 일부 검증을 생략하여 시간 복잡도를 줄이고, 보다 이해와 구현이 쉬운 방법을 제시하였다. 실험 결과에서는 패턴이 매우 짧고 k 값이 큰 경우를 제외하고는 제안하는 알고리즘은 패턴 길이가 증가할수록 수행시간이 급격히 감소함을 보인다.

향후 연구를 통해 보다 다양한 형태의 여과 알고리즘을 생각해볼 수 있겠고, 여과 알고리즘의 조합을 통하여 더 효율적인 수행 시간을 보일 수도 있을 것으로 생각된다. 또한 다중 패턴 매칭에 대해서도 고려해볼 수 있겠다.

REFERENCES

[1] Inbok Lee, "Mining Regular Expression Rules based on q-grams," *Smart Media Journal*, vol. 8, no.3, pp. 17-22, 2019.

[2] 허만우, 박기철, 홍지만, "사물인터넷 게이트웨이 보안을 위한 사용자 민감 데이터 분류," *스마트미디어저널,* 제8권 제4호, 17-24쪽, 2019년 12월

[3] 권혁록, 홍택은, 김판구, "AMI시스템에서 유사도를 활용한 누락데이터 보정 방법," *스마트미디어저널,* 제8권, 제4호, 80-84쪽, 2019년 12월

[4] J. Kim, P. Eades, R. Fleischer, S.-H. Hong, C. S. Iliopoulos, K. Park, S. J. Puglisi, T. Tokuyama, "Order-preserving matching," *Theoretical Computer Science*, vol. 525, pp. 68-79, 2014.

[5] M. Kubica, T. Kulczynski, J. Radoszewski, W. Rytter, T. Walen, "A linear time algorithm for consecutive permutation pattern matching," *Information Processing Letters*, vol. 113, no. 12, pp. 430-433, 2013.

[6] S. Cho, J. C. Na, K. Park, J. S. Sim, "A fast algorithm for order-preserving pattern matching," *Information Processing Letters*, vol. 115, pp. 397-402, 2015.

[7] T. Nakamura, S. Inenaga, H. Bannai, M. Takeda, "Order preserving pattern matching on trees and DAGs," *Proc of SPIRE*, pp. 271-277, 2017.

[8] T. Chhabra, Jorma Tarhio, "A filtration method for order-preserving matching," *Information Processing Letters*, vol. 116, pp. 71-74, 2016.

[9] J. C. Na, I. Lee, "A Simple Heuristic for Order-Preserving Matching," *IEICE Transaction on Information and Systems*, vol. 102, no. D(3), pp. 502-504, 2019.

[10] P. Gawrychowski, P. Uznanski, "Order-preserving pattern matching with k mismatches," *Theoretical Computer Science*, vol. 638, pp. 136-144, 2016.

[11] T. Chhabra, E. Giaquinta, J. Tarhio, "Filtration Algorithms for Approximate Order-Preserving Matching," *Proc of SPIRE*, pp. 177-187, 2015.

[12] G. Jacobson, K.-P. Vo, "Heaviest increasing/common subsequence problems," *Proc of CPM*, pp. 52-66, 1992.

저 자 소 개

이 인 복 (정회원)

1997년 서울대학교 컴퓨터공학과 학사 졸업.
1999년 서울대학교 컴퓨터공학부 석사 졸업.
2004년 서울대학교 전기컴퓨터공학부 박사 졸업.
2006년~현재 한국항공대학교 소프트웨어학과 부교수.

<주관심분야 : 알고리즘, 대용량 데이터 처리, 네트워크 보안>