# 웨이블릿 트리를 이용한 문자열 매칭 위치의 효율적인 히스토그램 계산

김성환, 탁해성, 조환규
부산대학교 전자전기컴퓨터공학과
e-mail:{sunghwan,tok33,hgcho}@pusan.ac.kr

# Efficient Histogram Calculation for String Matching Occurrences Using Wavelet Trees

Sung-Hwan Kim, Hae-Sung Tak, Hwan-Gue Cho
Dept. of Electric and Computer Engineering, Pusan National University

요 약

문자열 매칭은 긴 텍스트 문자열 상에 짧은 질의 문자열이 나타나는 모든 위치를 찾는 문제이다. 텍스트 문자열이 고정되어 있는 경우에는 접미사 트리나 접미사 배열과 같은 자료구조를 이용하여 보다 효율적인 문자열 매칭을 수행할 수 있다. 이 때 사용자 인터페이스에 관련되어, 또는 다른 통계적 처리를 수행하기 위하여 주어진 질의 문자열의 출현 위치에 대한 히스토그램을 계산할 필요성이 있다. 그러나 질의 문자열의 출현 횟수가 많은 경우 각 출현 위치를 모두 순회하며 집계해야 하므로 시간적으로 매우 비효율적이다. 본 논문에서는 웨이블릿 트리를 이용하여 접미사 배열을 색인함으로써 히스토그램 계산에 있어서 질의 문자열의 출현 횟수와는 시간적으로 독립적인 집계 기법을 제안한다. 또한 실험을 통하여 질의 문자열의 출현 횟수가 많을수록 제안 기법의 성능이 우수함을 보인다.

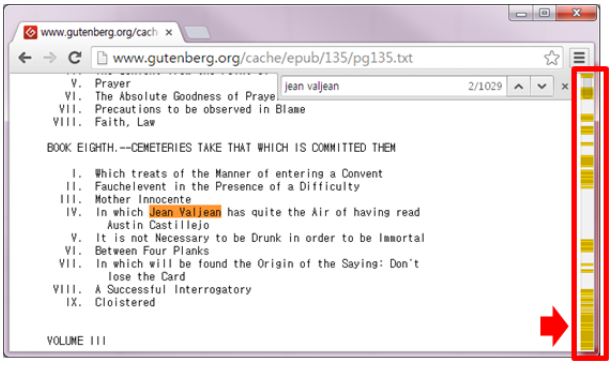
## 1. 연구동기

문자열 매칭은 컴퓨터과학의 고전적인 문제 중의 하나로 긴 텍스트 문자열 상에서 짧은 질의 문자열이 나타나는 모든 위치를 찾는 것이 목적이다. 문자열 매칭은 워드 프로세서나 웹 브라우저 등에서 현재 열려진 문서 내용 중 원하는 키워드를 검색하는 간단한 기능부터 기존의 단어 기반의 검색 엔진의 한계를 넘어 임의의 부분 문자열 검색을 위한 엔진 개발까지 다양한 분야에서 응용 및 연구되고 있다[1].

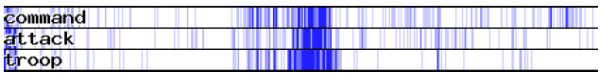
문자열 매칭과 관련하여 사용자 서비스를 제공하는 경우 질의 문자열이 출현한 위치들을 가시화할 필요성이 있다. 이러한 가시화 작업은 검색의 편의성을 증대시키며, 시각적 분석 기법등과 같은 다양한 분야에 이용될 수 있다. 다음은 문자열 매칭 결과의 가시화 사례이다.

그림-1(a)는 웹 브라우저 상에서 특정 단어를 검색한 경우를 보여준다. 우측의 스크롤바에 입력한 단어가 출현하는 위치가 나타나 있다. 사용자는 이를 통해 원하는 정보를 얻기 위해 어느 위치로 이동해야 할 것인지를 쉽게 파악할 수 있다.

그림-1(b)은 영문 문서에서 단어 “command,” “attack,” “troop”이 출현한 위치를 가시화 한 것이다. 각 단어의 출현 밀도에 따라 픽셀별로 채도가 다르게 표현되어 있어 문자열 전반에 걸친 분포를 쉽게 확인할 수 있다. 이와 같은 가시화 작업은 비슷한 의미나 문맥을 가지는 단어들의




(a) 웹 브라우저의 문자열 매칭 가시화 기능. 오른쪽 스크롤바에 검색한 문자열의 출현 위치가 표시된다.




(b) 단어 출현 패턴을 이용한 시각적 분석 기법. 특정한 의미의 단어는 연관된 주제가 서술되는 지점에 집중적으로 분포하며, 비슷한 의미를 가지는 단어가 유사한 위치에 동시 출현한다는 점을 시각적으로 분석할 수 있다.

(그림 1) 문자열 매칭 가시화의 응용 사례

동시 출현 경향을 확인하고, 이러한 단어 출현 위치나 유형을 통하여 특정 주제가 서술되는 시점이나 단어 간의 유사성을 확인할 수 있도록 한다.

일반적으로 문자열의 길이는 가시화를 위한 공간보다 큰 경우가 많으므로 이러한 작업을 위해서는 우선 질의 문자열 출현 위치를 블럭 단위로 집계하는 작업이 필요하

다. 예를 들어 길이 10,000인 소설에서 특정한 주인공의 이름을 검색하여 결과를 가로 100픽셀의 공간에 나타내고 싶다면 텍스트의 매 100번째 지점까지의 주인공 이름 출현 위치를 누계하여 각 픽셀에 표현하여야 하는 것이다.

문제는 만일 주어진 텍스트 문자열이 많은 수의 질의 문자열을 포함하고 있다면 집계 과정에 과도한 시간이 소요될 수 있다는 점이다. Knuth-Morris-Pratt 알고리즘과 같은 온라인 문자열 매칭을 한다면 이를 피할 수 없지만, 만일 말뭉치와 같은 매우 긴 텍스트 문자열 상에서 문자열 매칭을 수행하는 경우 텍스트 문자열을 미리 전처리함으로써 텍스트 문자열 길이에 관한 시간 복잡도 의존성을 줄일 수 있는데, 그럼에도 불구하고 질의 문자열의 모든 출현위치에 대하여 순회를 하면서 집계를 수행한다면 이는 매우 비효율적이라고 할 수 있다.

본 논문에서는 특히 접미사 배열에 기반을 둔 텍스트 문자열 색인 기법에 있어서 웨이블릿 트리를 추가적으로 이용하여 질의 문자열 출현 위치의 집계를 보다 효율적으로 수행할 수 있는 기법을 제안한다. 제안 기법은 접미사 배열과 웨이블릿 트리의 특성을 이용하여 패턴의 출현 횟수와는 독립적으로 집계를 수행할 수 있다.

## 2. 배경지식

### 2.1 접미사 배열

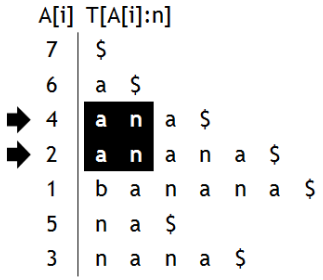
알파벳 $\Sigma$ 상의 길이 $n$인 문자열 $T[1:n]$가 있다고 하자. 편의상 $T[n]=\$(\notin\Sigma)$라고 가정한다. 이 때 $T$의 어떤 접미사 $T[i:n]$에 대하여 $T[j:n]<T[i:n]$인 모든 $j$의 개수를 $r_T(i)$라고 하자.

$$r_T(i)=|\{j\in[1,n]:T[j:n]<T[i:n]\}|$$

이 때, $r_T$는 접미사간의 사전순서에 따른 순위를 나타내므로 $\Sigma$에 잘 정의된 순서가 있고 $\$<\alpha(\forall\alpha\in\Sigma)$라고 정의하면 전단사 함수가 되며, 따라서 역함수 $r_T^{-1}(i)$도 정의할 수 있다. 그러면 $T$의 접미사 배열 $A_T$는 크기 $n$인 배열로 다음과 같이 정의된다[2].

$$A_T[i]=r_T^{-1}(i)$$

접미사 배열은 접미사들을 사전 순으로 정렬해놓은 순서를 나타내기 때문에, 공통의 접두사를 공유하는 접미사들은 접미사 배열 상에서 서로 인접한 위치에 있다. 따라서 문자열 매칭의 결과는 접미사 배열 상에서의 어떤 연속적인 구간으로 나타낼 수 있다. 그림-2는 문자열 “banana$”의 접미사 배열을 나타내며, 검은색으로 표시된 구간은 질의 문자열 “an”에 대한 매칭 결과를 나타낸다. “an”이 부분문자열로 등장한다는 것은 해당 위치에서 시작하는 접미사의 접두사가 “an”이라는 것과 같으므로 문자열 매칭이 발생한 위치들의 집합(1,3)은 접미사 배열 상에서 화살표로 표시된 연속된 구간으로 표현할 수 있음을 확인할 수 있다. 모든 매칭 위치를 알기 위해서는 이 구간 내의 모든 원소를 순회하면 된다.




(그림 2) 문자열 banana$에 대한 접미사 배열(왼쪽 수열)과 그에 대응하는 접미사(오른쪽 문자열). 문자열 an에 대한 매칭 결과가 검은색으로 표시되어 있다.

### 2.2 웨이블릿 트리

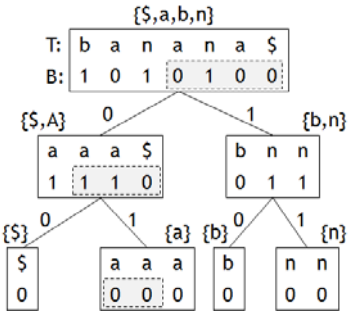
웨이블릿 트리[3]란 본래 임의의 알파벳 $\Sigma$상의 문자열에 대한 Rank 및 Select 질의를 처리하기 위한 자료구조로 문자열 처리와 관련한 다양한 문제에 활용되고 있다. Rank 질의란 주어진 문자열 $T$의 특정한 위치 $i$까지 문자 $c$가 출현한 횟수를 반환하며, Select 질의는 역으로 $i$번째 $c$의 위치를 반환하는 질의이다.

$$Rank_T(c,i)=|\{j\in[1,i]:T[j]=c\}|$$

$$Select_T(c,i)=j\ s.t.\ Rank(c,j-1)=i-1\ \wedge\ T[j]=c$$

이진 문자열 상에서의 Rank와 Select 질의는 상수시간에 수행이 가능하다[5]. 웨이블릿 트리는 이러한 사실을 이용하여 각 질의를 $O(\lg|\Sigma|)$시간에 수행할 수 있도록 설계되어 있으며 다음과 같이 구축한다. 주어진 알파벳 $\Sigma$를 두 부분집합 $\Sigma_0$, $\Sigma_1$으로 분할($\Sigma_0\cup\Sigma_1=\Sigma$, $\Sigma_0\cap\Sigma_1=\phi$)한 후, $T$의 각 문자 $T[i]$에 대하여 $T[i]\in\Sigma_0$이면 0으로, $T[i]\in\Sigma_1$이면 1로 나타낸 이진 문자열 $B$를 루트 노드에 할당한다. 이후, 각 알파벳 $\Sigma_0$과 $\Sigma_1$로 이루어진 $T$의 부분서열(subsequence)을 구하여 왼쪽과 오른쪽 노드로 각각 할당한 후 자식 노드들에 대하여 알파벳을 더 이상 분할할 수 없을 때 까지 같은 작업을 재귀적으로 반복한다. 그림 3에 웨이블릿 트리의 예가 있다.

웨이블릿 트리에서 부모 노드에서 연속적인 구간에 있다면 자식 노드에서도 대응되는 문자들이 각각 연속적인 구간에 위치한다는 점을 이용하여 다양한 연산을 수행할 수 있다[4].




(그림 3) 문자열 banana$에 대한 웨이블릿 트리. 각 노드에서는 해당 노드에 대응하는 알파벳 집합을 절반으로 분할하여 각 알파벳 부분집합으로 이루어진 부분서열을 자식노드로 가진다. 따라서 부모노드에서 연속적인 구간은 자식노드에서도 연속적이다.

## 3. 문제 정의

길이 $n$인 텍스트 문자열 $T$와 길이 $m$인 질의 문자열 $Q$가 있을 때, $T$에 대한 $Q$의 매칭 $M_T(Q)$는 $T$상에 $Q$가 출현한 모든 위치의 집합이다.

$$M_T(Q) = \{i \in [1, n-m+1] \mid T[i:i+m-1] = Q\}$$

매칭 결과 $M_T(Q)$ $M_T(Q)$에 대한 $k$-히스토그램 $H_{Q,T}^{(k)}$는 길이 $k$의 배열로, $T$를 $k$등분한 후, $Q$의 출현 위치를 각 등분에 따라 집계한 결과를 나타낸다.

$$H_{Q,T}^{(k)}[j] = \sum_{i \in M_T(Q)} I_T^{(k)}(i,j)$$

(단, $n(j-1)/k < i \leqq nj/k$이면 $I_T^{(k)}(i,j) = 1$, 이외의 경우에는 $I_T^{(k)}(i,j) = 0$)

편의를 위하여 $T$의 길이 $n$과 히스토그램의 크기 $k$는 모두 2의 거듭제곱이라고 가정한다. 다음은 히스토그램 계산에 관한 예제이다.

**예제 1** $T$의 길이가 16이고 $M_T(Q) = \{2,4,6,9,12,15\}$일 때 $H_{Q,T}^{(4)}$, $H_{Q,T}^{(8)}$는 다음과 같다.

| $i$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $H_{Q,T}^{(4)}[i]$ | 2 | 1 | 2 | 1 |

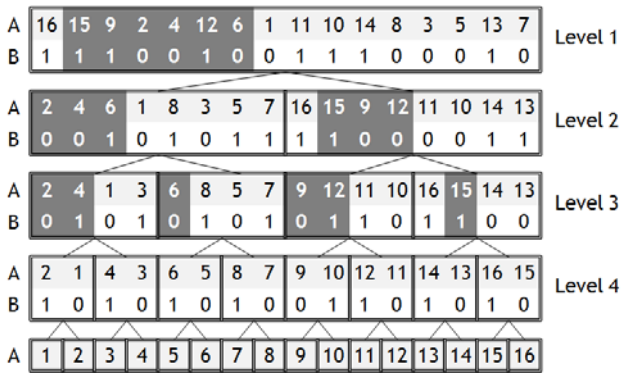
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| $H_{Q,T}^{(8)}[i]$ | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |

접미사배열 기반의 문자열 매칭을 이용한 탐색의 경우 매칭 결과가 접미사배열 상의 구간 $[i,j]$로 나타나는데, 이때 $M_T(Q) = \{A[i],\cdots,A[j]\}$이고, 이를 이용하여 히스토그램을 계산하기 위해서는 히스토그램 배열을 0으로 초기화한 후 $A[i]$부터 $A[j]$까지 순회를 하며 해당 지점에 1씩 더해주는 방법을 사용해야 한다. 그러나 이러한 방식은 질의 문자열의 출현횟수에 의존적이기 때문에 대규모 말뭉치 상에서의 탐색과 같이 매우 많은 수의 매칭이 발생할 수 있는 경우 비효율적일 수 있다. 본 논문에서는 웨이블릿 트리를 이용하여 접미사 배열을 색인함으로써 매칭 수가 많은 경우에 대하여 히스토그램 계산 속도를 증가시킬 수 있는 방법을 제안한다.

## 4. 웨이블릿 트리를 이용한 위치 색인

앞서 언급한 바와 같이 접미사 배열 $A$은 각 순위의 접미사들의 위치를 사전순서로 나열한 배열로 $[1,n]$의 순열(permutation)이므로, 따라서 알파벳 $\Sigma = [1,n]$ 상에서 정의되는 하나의 문자열로 표현할 수 있다. 따라서 접미사 $A$를 문자열로 취급하여 이를 웨이블릿 트리를 이용해 색인을 수행할 수 있다.

접미사 배열을 웨이블릿 트리로 나타내면 트리의 깊이는 $\lg n$이 되며 (웨이블릿 트리에서 문자를 하나만 포함하고 있는 말단 노드는 그 자체로 가지는 정보가 없기 때문에 저장하지 않는다), 각각의 접미사 배열 원소에 대하여 루트 노드에서 $\lg n$ 단계 노드까지 해당되는 모든 비트열을 접합하게 되면 해당 접미사 위치에서 1을 뺀 수를 이진수로 표현한 문자열이 된다.




(그림 4) 접미사 배열을 웨이블릿 트리로 구성한 예제. A는 접미사 배열 원소, B는 웨이블릿 트리 상에 저장된 비트열. 각 접미사 배열의 원소에 대하여 자식 노드로 탐색해 나가면 해당 원소에서 1을 뺀 값을 이진수로 표현했을 때 상위비트를 차례로 얻을 수 있다.

그림 4는 문자열 $T$="misissippissi$"에 대한 접미사배열을 웨이블릿 트리로 표현한 예를 나타낸다. 문자열의 길이 $n$이 16이므로 트리의 깊이는 $\lg 16 = 4$이다. $A[3] = 9$에 해당하는 비트열을 루트노드에서부터 차례로 추적해보면 루트노드에서 1, 그 자식노드에서는 0, 이후로는 차례로 0, 0이며, 이는 $A[3]-1 = 8$의 이진수 표현 1000과 일치함을 알 수 있다.

만일 계산하고자 하는 히스토그램이 $2^k$개의 빈(bin)을 가지고 있다면 어떤 접미사 배열의 원소 $A[i]$가 속하게 되는 빈(bin)의 번호는 $1 + \lfloor (A[i]-1)/2^{\lg n - k} \rfloor$ 이다. 이때, $\lfloor (A[i]-1)/2^{\lg n - k} \rfloor$는 $A[i]-1$를 이진수로 표현했을 때 상위 $k$개의 비트열이 된다. 예를 들어 앞선 예제에서 히스토그램의 빈(bin) 개수가 $2^3 = 8$이라면, $A[3] = 9$가 집계되는 빈(bin)의 번호는 $100_{(2)}$+1=5번 bin이 된다. 예제 1에서 $H_{Q,T}^{(8)}[5]$는 $M_T(Q)$의 원소 중 9가 반영된 것이다.

앞서 웨이블릿 트리에서는 부모 노드에서 연속된 구간은 자식 노드에서도 연속적으로 나타난다는 사실을 언급한 바 있다. 따라서 접미사 배열 상에서의 매칭 결과가 구간 $[i_1,i_2]$로 나타난다면 각각의 $i \in [i_1,i_2]$에 대하여 $A[i]$이 들어갈 빈(bin) 번호를 구한 후 히스토그램에 반영하는 것이 아니라 웨이블릿 트리 상에서의 비트 연산을 이용하여 일괄적으로 집계할 수 있다. $k$ 단계의 노드까지 탐색한다면 해당 단계의 각각의 노드는 히스토그램 빈(bin) 번호를 결정하는 상위 $k-1$개의 비트에 대한 정보를 가지고 있으며, 해당 노드에서의 0과 1이 마지막 $k$번째의 비트를 결정한다. 따라서 $k$단계의 각 노드에서 0과 1의 개수가 곧 히스토그램의 원소가 된다.

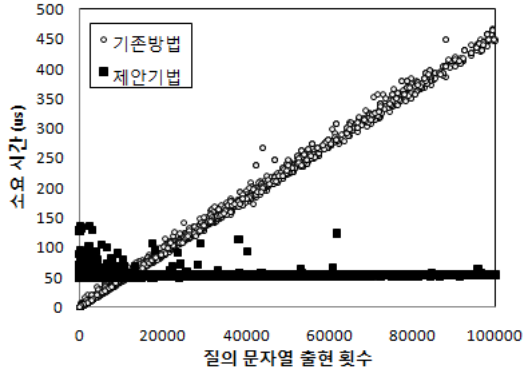
예를 들어, 그림 4에서 주어진 매칭 결과 구간이 [2,7]이라면 해당 구간 내에 있는 접미사 배열 원소 중 최상위비

트가 0, 1인 숫자가 각각 3개씩이라는 사실은 이진 문자열 상의 Rank 질의를 통해 쉽게 알 수 있다. 따라서 만약 $H^{(2)}_{Q,T}$를 구하고자 했다면 $H^{(2)}_{Q,T}[1]=H^{(2)}_{Q,T}[2]=3$이 된다. $H^{(4)}_{Q,T}$를 구하고자 한다면 다음 단계의 자식 노드 구간을 살펴보면 된다. 우선 루트 노드에서 구간 [2,7]에 해당하는 왼쪽, 오른쪽 자식 노드에서의 구간을 구한다. 왼쪽 자식 노드에서의 구간은 부모 노드 구간에 포함되는 0들의 순위와 같으므로 [1,3]이며, 이에 대응하는 접미사배열의 원소는 {2,4,6}이다. 마찬가지로 오른쪽 자식 노드에서의 대응 구간은 [2,4]이다. 이들 구간들은 그림 4에 음영으로 표시되어 있으며 히스토그램을 계산할 때에는 2단계 각 노드에서의 구간에 해당하는 비트열에서 0과 1의 개수를 세면 된다. 왼쪽 자식 노드는 $H^{(4)}_{Q,T}[1]$과 $H^{(4)}_{Q,T}[2]$를 나타낸다. 왼쪽 자식 노드에서 구간 [1,3]내에 있는 0의 개수는 2개이므로 $H^{(4)}_{Q,T}[1]=2$이고 1의 개수는 1개이므로 $H^{(4)}_{Q,T}[2]=1$가 된다. 오른쪽 자식 노드로부터는 $H^{(4)}_{Q,T}[3]$과 $H^{(4)}_{Q,T}[4]$를 구할 수 있는데, 마찬가지 방법으로 구하면 $H^{(4)}_{Q,T}[3]=2$, $H^{(4)}_{Q,T}[4]=1$임을 알 수 있다. 이는 예제 1에서의 결과와 동일하다.

제안 기법은 히스토그램의 크기가 $|H|=2^k$일때 $k$단계의 노드까지 순회를 하면 되며, 방문하게 되는 노드는 총 $2^k-1$개가 된다. 따라서 매칭 구간의 길이와는 상관없이 $O(|H|)$시간에 히스토그램을 계산할 수 있다.

## 5. 실험 및 결과

제안 기법의 검증을 위하여 문자열 매칭에서 주로 사용되는 Pizza&Chilli의 영어 말뭉치[6]에서 영어 알파벳만을 취한 후 128M로 절단하고, 길이별로 1,000개씩 질의 문자열을 무작위로 추출하여 매칭을 수행한 후 빈(bin)의 수가 1024인 히스토그램 계산을 수행하였다. 결과는 그림 5에 나타나있듯이 제안 기법은 측정 과정에서의 노이즈를 무시한다면 문자열 출현횟수랑은 무관하며, 출현횟수가 11,800이상일때 기존방법에 비해 우수한 성능을 보인다.




(그림 5) 히스토그램 계산 실험 결과(히스토그램 크기 1024). 기존 방법은 질의 문자열 출현 횟수에 비례하여 시간이 소요되지만, 제안 기법은 질의 문자열의 출현 횟수와는 독립적임을 알 수 있다. 교차 지점은 질의 문자열 출현 횟수가 11800인 지점이다.

## 6. 결론

문자열 매칭 결과를 가시화하기 위해서는 히스토그램의 계산이 필요하다. 일반적으로 질의 문자열의 모든 출현 위치를 순회하며 히스토그램을 구축할 수 있지만, 출현 위치의 개수가 히스토그램의 크기에 비하여 매우 많은 경우에는 많은 시간이 소요될 수 있다. 본 논문에서는 웨이블릿 트리를 이용하여 접미사 배열을 색인함으로써 접미사 배열에 기반을 둔 탐색 기법에 있어서 질의 문자열 출현횟수와는 독립적인 히스토그램 집계 기법을 제안하였다. 실험을 통하여 제안 기법은 히스토그램 집계에 소요되는 시간이 질의 문자열 출현횟수와 독립적이라는 사실을 확인하였으며, 128M 크기의 영문 말뭉치 상에서 11,800회 이상의 출현 횟수를 가지는 질의문으로 검색한 결과를 1024개의 빈(bin)을 가지는 히스토그램을 집계를 하고자 하는 경우 일반적인 기법에 비하여 속도 향상이 있음을 확인하였다. 추후에는 제안 기법을 활용한 다해상도 매칭 결과 가시화 기법을 구현할 예정이며, 한편으로는 압축 접미사 배열[7]이나 FM-index[8]와 같은 다양한 문자열 매칭 색인 기법과의 결합을 통하여 실제 문자열 매칭에 응용하는 경우 고려해야 하는 다양한 문제를 고려해보고자 한다.

## 감사의 글

이 논문은 2013년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No.2011-0015359).

## 참고문헌

[1] A. Fariña, N. R. Brisaboa, G. Navarro, F. Claude, Á. S. Places and E. Rodríguez, "Word-based Self-Indexes for Natural Language Text," ACM Transactions on Information Systems, Vol. 30, No. 1, pp.1:1-34, 2012.

[2] U. Manber and G. Myers, "Suffix Arrays: A New Method for On-line String Searches," SIAM Journal of Computing, Vol. 22, No. 5, pp.935-948, 1993.

[3] R. Grossi, A. Gupta and J. S. Vitter, "High-order Entropy-Compressed Text Indexes," in Proc. of the 14th ACM/SIAM Symposium on Discrete Algorithms (SODA), pp. 841-850, 2003.

[4] G. Navarro, "Wavelet Trees for All," Jornal of Discrete Algorithms, Vol. 25, pp. 2-20, 2013.

[5] V. Mäkinen and G. Navarro, "Rank and Select Revisited and Extended," Theoretical Computer Science, Vol. 387, pp.332-347, 2007.

[6] P. Ferragina and G. Navarro, Pizza&Chili Corpus, http://pizzachili.di.unipi.it/texts.html

[7] R. Grossi and J. S. Vitter, "Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching," SIAM Journal on Computing, Vol. 35, No. 2, pp.378-407, 2005.

[8] P. Ferragina and G. Manzini, "Opportunistic Data Structures with Applications," in Proc. of the 41th Symposium on Foundations of Computer Science (FOCS), pp.390-398, 2000.