실시간 속기 자막 환경에서 멀티미디어 정보 검색을 위한 Prefix Array

The Prefix Array for Multimedia Information Retrieval in the Real-Time Stenograph

김동주*, 김한우**
Dong-Joo Kim*, Han-Woo Kim**

Abstract - This paper proposes an algorithm and its data structure to support real-time full-text search for the streamed or broadcasted multimedia data containing real-time stenograph text. Since the traditional indexing method used at information retrieval area uses the linguistic information, there is a heavy cost. Therefore, we propose the algorithm and its data structure based on suffix array, which is a simple data structure and has low space complexity. Suffix array is useful frequently to search for huge text. However, subtitle text of multimedia data is to get longer by time. Therefore, suffix array must be reconstructed because subtitle text is continually changed. We propose the data structure called prefix array and search algorithm using it.

Key Words : 멀티미디어 정보검색, 속기, 자막, suffix array, prefix array

1. 서론

근래 디지털 기반시설 확충과 정보기술의 발전으로 인한 디지털방송과 공중파 멀티미디어 방송의 확대, 초고속망의 보급으로 인하여 상용화를 앞둔 IPTV, 등과 같은 멀티미디어 서비스가 증가함에 따라 부가 서비스에 대한 요구가 날로 커지고 있다. 이에 개인용비디오녹화기(personal video recorder)나 사용자 주문형 비디오(video on demand)의 보급으로 방송프로그램을 녹화하여 언제든지 사용자가 원하는 시간에 보거나 시간 천이(time shift) 기능으로 방송프로그램을 시청하는 중에 멈추거나 되돌릴 수 있게 되었다. 이렇게 방송 중에 있는 디지털 방송뿐만 아니라 인터넷 스트리밍 서비스 등에서의 많은 방송 프로그램들의 효율적인 저장과 관리를 위해 구축되고 있는 멀티미디어 아카이브에서 필수적인 요소 중의 하나는 검색이다. 검색에는 동영상을 대상으로 저수준의 특징 정보를 이용하여 동영상 자체를 질의로 주거나 정지영상을 질의로 주고 이와 유사한 동영상을 결과로 제시하는 방법도 있으나 사용자의 개념적인 요구를 만족시키기에는 정확성이 많이 떨어질 뿐만 아니라 방송 중에 있는 영상을 녹화하면서도 언제든 즉시 검색을 수행하기에는 많은 부담이 존재한다. 본 논문은 방송 중에 있는 영상에 대해 동영상과 영상과 동기화되어 있는 텍스트로 된 명세를 이용하여 영상을 검색하는 자료구조와 알고리즘에 관한 것이다.

동영상에 대한 텍스트 명세는 MPEG-2에서 비디오, 오디오, 텍스트를 포함하고 있는 전송스트림(transport stream)과 PSI(Program Specific Information)로부터의 방송프로그램에 대한 각종 부가 정보에서 동기정보(synchronous information)를 의미하는데, 이들 정보 중 본 논문에서의 논의 대상은 자막이다. 자막은 폐쇄자막(closed caption)과 개방자막(open caption)으로 나눠지고 자막방송에서의 자막은 폐쇄자막을 의미하며, 이러한 자막방송은 사전제작 방식과 실시간 방식으로 나눠진다. 사전제작 방식은 드라마나 영화, 다큐멘터리 등과 같이 녹화방송의 대본을 통하여 사전에 방송 프로그램 전체의 자막을 생성하는 방식을 말하며, 온라인 방식이라고도 하는 실시간 방식은 스포츠나 뉴스와 같이 대본이 존재하지 않는 생방송에 속기사에 의해 실시간으로 자막을 생성하는 방식을 말한다. 여러 개의 방송 프로그램 자막들을 병합하는 문제가 남긴 하지만, 사전제작 방식의 경우에는 검색을 위한 정보 역시 사전에 생성 가능하지 때문에 해당 방송 프로그램의 송출 직전에 방송 프로그램 전체의 검색을 위한 정보를 먼저 송출함으로써 검색이 가능하게 된다. 반면에 실시간 방식은 가입자뿐만 아니라 방송국도 방송 프로그램 처음부터 전체 자막 내용을 알 수 없기 때문에 검색을 위한 자료구조나 정보를 사전에 생성할 수 없으므로 자막이 전송될 때마다 매번 다시 구축하거나 추가적으로 전송되는 자막 부분에 관한 정보를 이미 구축된 기존 정보에 추가해주어야만 한다.

본 논문에서는 실시간 자막 환경에서 시간에 따라 점진적으로 증가하는 자막 텍스트의 전문 검색을 위해 전송되어 온 자막에 관한 검색 정보를 이전 시간에 구축된 검색 정보에 추가하는데 적합한 자료구조와 알고리즘을 제안한다. 본 논문에서는 디지털방송 환경에 제한하여 기술하고 있지만, 시간에 따라 점진적으로 증가하는 실시간 자막은 단지 디지털방송 환경에서만 요구되는 것이 아니라, 전자법정에서 심리 과정을 녹화하는 경우나, 의회의 전자 속기, 전자 회의 등과 같은 동영상이나 음성의 아카이빙과 더불어 속기사의 속기 자막을 필요로 하는 곳이면 어디든 요구된다. 또한 실시간 자막 환경의 특수성을 반영한 방법이긴 하지만 사전제작 방식

저자 소개
* 김동주: 漢陽大學 컴퓨터工學科 博士課程
** 김한우: 漢陽大學 컴퓨터工學科 敎授 · 工博

에도 구분 없이 사용하여 셋톱박스나 개인용비디오녹화기에서 전송되어 오는 자막이 실시간 자막인지 사전제작 자막인지를 구분할 필요가 없다.

2. 점진적으로 증가하는 자막 검색

시간의 흐름에 따라 점진적으로 크기가 증가하는 실시간 자막을 검색하기 위한 검색 정보구축 방법 중의 하나는 전통적인 정보검색 분야에서 사용하는 단어 색인 방법이다. 이 방법을 적용하기 위해서는 전송되어 온 자막으로부터 색인어를 추출하여 영상 프레임 정보와 함께 저장하여야 한다. 그러나 이러한 색인기반 방법은 색인어 추출에 많은 비용이 들어 멀티미디어 전송 중에 실시간으로 검색 정보를 구축하여 언제든지 검색할 수 있도록 하기에는 많은 부담이 존재한다.

반면에 Knuth-Morris-Pratt 알고리즘[1]은 검색하고자 하는 패턴의 길이가 m이고 검색 대상인 텍스트의 길이가 n일 때 검색 정보의 구축비용이 전혀 없어 $\log(m+n)$의 시간에 검색이 가능하다. 그러나 이 알고리즘은 패턴을 모두 찾기 위해 매번 질의에 대해 항상 $\log(m+n)$의 시간이 걸리므로 검색이 빈번하거나 검색 대상의 텍스트가 매우 긴 경우에는 효율적이지 못한 방법이다.

속기사가 입력하는 실시간 자막뿐만 아니라 사전제작 자막을 포함한 일반적인 자막방송에서 자막은 방송 프로그램의 시작 전에 전체 자막이 한꺼번에 전송되는 것이 아니라 부분 문장 단위로 전송된다. 따라서 현재 전송된 자막에 대해 이전에 전송된 자막과 함께 검색을 위한 자료구조를 새롭게 구축해야만 한다. 이 문제를 해결하기 위해 이 장에서는 검색이 빈번하고 텍스트의 길이가 매우 긴 경우에 적합한 새로운 자료구조와 알고리즘을 제안한다. 알고리즘이 간단하면서도 빠르게 동작하는 suffix array[2] 자료구조와 알고리즘의 단점을 극복한 유사한 개념의 prefix array라는 자료구조와 구축 알고리즘을 제안하고 검색 방법을 제시한다.

2.1 Suffix Array

Suffix Array는 대규모 텍스트를 효과적으로 검색하기 위해 설계된 자료구조로 텍스트에 존재하는 어떠한 부분 문자열도 간단하게 찾아낼 수 있을 뿐만 아니라 발생 빈도도 손쉽게 알아낼 수 있다. Suffix array는 suffix tree[3]에 비해 구축 시간은 더 오래 걸리지만 검색 시간이 더 빠를 뿐만 아니라 적은 메모리 공간을 차지한다. 이 자료구조는 개념적으로 주어진 텍스트의 모든 접미 문자열의 정렬된 목록이다. 즉, 길이 n인 텍스트를 $A=a_0a_1\cdots a_{n-1}$이라고 했을 때, 텍스트에 존재하는 한 접미 문자열은 텍스트의 특정 위치 i를 시작으로 텍스트의 끝 $n-1$에 이르는 길이 $n-i$인 문자열 $A_i=a_ia_{i+1}\cdots a_{n-1}$을 의미한다. 이 때 접미 문자열 집합 $\{A_0,A_1,...A_{n-1}\}$에서 k번째로 작은 접미 문자열의 시작 위치가 suffix array $SA[k]$에 저장된다. 여기서 작다는 의미는 사전적 순서에서 먼저 놓인다는 의미이다. 임의의 문자열을 검색하기 위해서는 suffix array를 이용하여 정렬된 접미 문자열을 이진 검색을 수행하면 된다. 만약 검색 대상 텍스트가 $A=abaababaabaab$라고 했을 때, suffix array SA를 구축하면 그림 1과 같다. 즉, suffix array SA의 첫 번째 원소 $SA[0]$가 가리키는 시작점은 접미 문자열 $A_{10}=aab$를 의미하고 $SA[1]$가 가리키는 시작점은 $A_7=aabaab$를 의미한다. 또한 배열 Lcp(Longest common prefix)는 이웃하는 suffix array의 최장 공통문자열의 길이로 $Lcp[k]$에는 $SA[k]$와 $SA[k+1]$가 가리키는 두 접미 문자열에서 가장 긴 공통 접두 문자열의 길이가 저장된다.

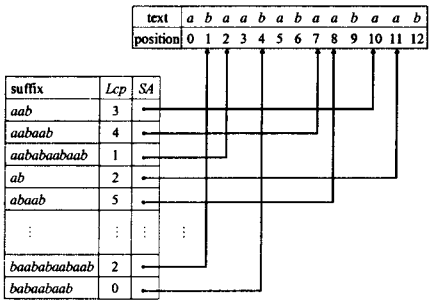

그림 1. Suffix Array의 구축 예

Suffix를 구축하는 간단한 방법은 효율적인 정렬 알고리즘을 이용하는 것이다. 이렇게 할 경우 $O(n\log n)$의 문자열 비교가 $O(n)$번 수행되어야만 하기 때문에 전체 수행시간은 $O(n^2\log n)$이 걸린다. 중복 비교를 피하기 위해 부분 정렬 결과를 사용하는 좀 더 나은 알고리즘은 최악의 경우 $O(n\log n)$이 걸린다. 또한 길이 m인 패턴 P를 검색할 때 걸리는 시간은 $O(m\log n)$이지만 Lcp 정보를 이용한다면 $O(m+\log n)$의 시간에 가능하다. 물론 이 정보는 본 논문에서의 Lcp와는 조금 다른 $Llcp$, $Rlcp$라는 것이다.

2.2 실시간 속기자막의 검색

방송중에 있는 멀티미디어 데이터가 실시간 속기 자막과 함께 전송되어 올 경우 검색대상 자막의 크기는 시간에 따라 점진적으로 커진다. 즉, 2.1절의 예와 같은 전체 문자열에 대해 매번 하나의 문자씩 전송된다면 그림 2와 같다.

time	text	suffix	SA	Lcp
t^0	a	A_0^0	0	0
t^1	ab	A_0^1	0	0
		A_1^1	1	0
t^2	aba	A_2^2	2	1
		A_0^2	0	0
		A_1^2	1	0
t^3	$abaa$	A_3^3	3	1
		A_2^3	2	1
		A_0^3	0	1
		A_1^3	1	0
⋮	⋮	⋮	⋮	⋮
t^{12}	$abaababaabaab$	A_{10}^{12}	10	3
		A_7^{12}	7	4
		A_2^{12}	2	1
		A_{11}^{12}	11	2
		⋮	⋮	⋮
		A_1^{12}	1	2
		A_4^{12}	4	0

그림 2. 시간에 따른 suffix array의 변화

그림 2의 예에서 A_i^j는 t^{j-1} 시간의 문자열에 t^j시간에 전송된 문자가 추가된 문자열에서 i번째 위치부터 시작되는 suffix이다. 그림 2의 경우 매 시간에 suffix array를 구축하

는데 $\log(t\log t)$의 비용이 들고 이를 n번 수행하기 때문에 전체 식 (1)의 비용이 든다.

$$O\left(\sum_{t=0}^{n-1} t\log t\right) \quad (1)$$

물론 그림 2의 예는 매우 극단적인 예로 특정 시간에 전송되는 텍스트는 문자나 단어 단위가 아닌 부분 문장 단위이기 때문에 식 (1)과 같은 비효율성은 존재하지 않지만 실제의 환경에서도 역시 매번 재구축해야만 하는 부담은 여전히 남는다.

2.3 Prefix Array

그림 2에서의 예와 같이 실시간 자막 환경에서 특정 시간에 전송되어 온 자막은 모든 suffix A_i를 변화시킨다. 따라서 2.2절에서와 같이 suffix array를 재구축해야 하는데, 더 나은 방법으로 $A_{SA[k]}$의 맨 뒤에 추가된 문자 $A_{SA[k]}[n-SA[k]]$가 $l=k+1$부터 $l=n-1$까지의 모든 $A_{SA[l]}[n-SA[k]]$보다 크거나 같을 때까지 비교를 수행하여 $A_{SA[l]}$의 위치를 찾아내 순서를 조정하는 방법이 있다. 물론 평균적인 경우에는 더욱 효율적으로 동작하기는 하지만 최악의 경우 특정 시간에 재구축하는 비용은 2.3절의 그림 2의 예와 동일하다. 따라서 본 논문에서는 이보다 효율적이고 간단한 방법으로 순방향 비교가 아닌 역방향 비교를 통하여 재구축 문제를 검색문제로 변경하여 $O(\log n)$ 시간에 재구축하기 위한 알고리즘과 자료구조 prefix array PA를 제안한다.

먼저 그림 2에서 새롭게 전송된 문자가 모든 suffix에 영향을 받지 않도록 하기 위해 prefix P_i를 $P_i=p_0p_1..p_i$로 정의하고 $REV(P_i)$를 $REV(P_i)=p_ip_{i-1}..p_0$로 정의한다. 즉, P_i를 텍스트 P에서 0번째 위치에서부터 i번째 위치까지의 길이 $i+1$인 prefix로 정의한다. 이렇게 하면 t^j 시간의 prefix들은 t^{j-1} 시간의 prefix에 prefix P_j만 새롭게 만들어지게 된다. 따라서 t^j 시간에 새로운 문자가 전송되더라도 t^{j-1} 시간의 모든 prefix $P_0,P_1,...,P_{j-1}$은 변경되지 않고 P_j만 PA에 추가된다. 그런데 PA는 모든 prefix가 순방향으로 비교 정렬되어 생성되는 것이 아니라 역방향으로 비교 정렬되어 생성된다. 이를 텍스트 $P=abaababa$에 대해 구축한 PA를 개념적으로 설명하면 그림 3과 같다.

prefix		PA	Lcp
a	$REV(P_0)$	0	1
aaba	$REV(P_3)$	3	1
aba	$REV(P_2)$	2	3
abaaba	$REV(P_5)$	5	3
ababaaba	$REV(P_7)$	7	0
ba	$REV(P_1)$	1	2
baaba	$REV(P_4)$	4	2
babaaba	$REV(P_6)$	6	0

그림 3. Prefix Array

만약 그림 3의 텍스트에 새로운 문자 '*a*'가 추가된다면 $P_8=abaababaa$을 역순으로, 즉 $REV(P_8)=aabababa$을 비교하여 PA에 추가하면 된다. 이제 이 P_8를 추가하는 문제는 검색문제로 바뀐 것이다. 따라서 Lcp 정보를 이용한 검색 정보의 갱신은 매 시간마다 $O(t+\log t)$의 시간에 가능하고 n번 수행에 대한 전체 시간 복잡도는 식 (2)와 같다.

$$O\left(\sum_{t=0}^{n-1}(t+\log t)\right) \quad (2)$$

특정 패턴 $Q=q_0q_1...q_{m-1}$의 검색 경우 역시 역방향으로 비교한다. 그런데 2.1절이나 2.2절의 경우와는 달리 검색된 결과 $PA[k]$가 가리키는 값은 패턴과 일치하는 텍스트 상의 끝 위치이므로 일치하는 패턴의 시작위치는 $PA[k]-m+1$가 되며 suffix로 표현한다면 $A_{PA[k]-m+1}$가 된다. 이 경우 시간 복잡도는 suffix array의 경우와 마찬가지로 $O(m+\log n)$이 된다.

3. 실험 및 평가

일반적인 텍스트에 대한 순수 알고리즘의 성능 평가를 위해 100Mbytes 크기의 텍스트 파일을 한 줄씩 누적하여 SA나 PA를 구축해가면서 Intel P4 PC에서 실험하였다. 그림 4는 5Mbytes 단위로 누적되는 텍스트를 suffix array와 prefix array를 구축하는데 걸린 시간을 측정한 그래프이다. 제안하는 방법이 전반적으로 70.03%의 성능향상을 보였다.

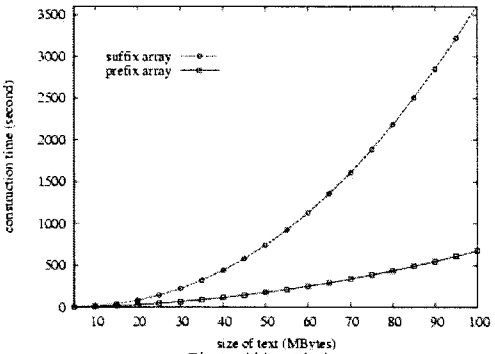

그림 4. 성능 평가

4. 결론

본 논문에서는 실시간 자막환경에서 시간에 따라 점진적으로 증가하는 자막 텍스트의 전문 검색을 위해, 전송 자막에 관한 검색 정보를 이전 시간에 구축된 검색 정보에 효과적으로 추가할 수 있는 prefix array 자료구조와 알고리즘을 제안하였다. 제안하는 prefix array는 전송 자막에 대한 검색정보 추가 문제를 검색문제로 변경하여 매우 효율적으로 검색정보를 갱신할 수 있었다.

참 고 문 헌

[1] Donald Knuth, James H. Morris, Jr, Vaughan Pratt, "Fast pattern matching in strings," SIAM Journal on Computing. pp.323-350, 1977.

[2] U. Manber, G. Myers, "Suffix arrays: a new method for on-line string searches," Proc. of the 1st annual ACM-SIAM symposium on Discrete algorithms, pp.319-327, 1990.

[3] E. Ukkonen, "Constructing suffix trees on-line in linear time in algorithms, software, architecture," Proc. IFIP 12th Word Computer Congress, pp.484-492, 1992.