과학 데이타베이스에서 부분 문자열의 발생 빈도 예측

배진욱°, 이석호
서울대학교 전기·컴퓨터공학부
oblody@db.snu.ac.kr°, shlee@cse.snu.ac.kr

Frequency Estimation of Substring for Scientific Database

Jinuk Bae° Sukho Lee
School of Electrical Engineering and Computer Science, Seoul National University

요 약

대량의 짧은 문자열들에 대해 부분 문자열의 발생 빈도를 예측하는 문제는 카운트 서픽스 트리를 미리 생성한 후 이를 이용함으로써 처리될 수 있다. 카운트 서픽스 트리는 모든 부분 문자열의 발생 빈도를 저장한 뒤 가지치기를 함으로써, 제한된 트리 크기와 발생 빈도 예측이라는 두 가지 목표를 처리한다. 하지만, 영기서열에서 처럼 저장된 문자열의 길이가 길어질 경우 카운트 서픽스 트리를 생성하기가 대단히 어려워진다는 문제점이 발생한다. 이 논문에서는 선삽입, 후가지치기 방식의 카운트 서픽스 트리 대신, 처음부터 길이가 q 이하인 문자열들만을 삽입하는 큐그램 트리를 제안한다. 큐그램 트리는 제한된 트리 크기에 따라 저장할 부분 문자열의 크기를 미리 결정할 수 있으며, 데이타베이스에 저장된 문자열의 전체 길이가 N일 때 O(N) 시간에 생성 가능하다. 실험 결과 제한된 부분 문자열을 가지고 있음에도 불구하고 긴 부분 문자열의 발생 빈도를 매우 정확하게 예측할 수 있음을 보였다.

1. 서론

샘플링[1]이나 히스토그램[2] 등을 이용하여 숫자 데이타에 대한 발생빈도 (또는 선택도)를 추정하는 문제는 오랫동안 연구되어왔다. 반면에 그림 1의 질의처럼 와일드문자가 포함된 문자열이 조건으로 주어졌을 때 투플들의 선택도를 추정하는 문제는 [3]에서 처음으로 제기되었고, [4]에서 보다 정확한 추정 방법을 제시하였다. [5, 6, 7, 8]에서는 둘 이상의 차원이나 나뭇가지 모양의 질의로 확장된 문제를 다루었다.

```
select *
from part
where color like '*green*';
```

그림 1. 와일드문자가 사용된 질의

[3, 4]에서는 문자열 데이타에 포함된 서픽스(suffix)들의 발생 회수를 서픽스 트리[9]의 변형인 카운트 서픽스 트리(Count Suffix Tree)에 저장한다. 이 때, 히스토그램의 크기에 제한이 있는 것과 마찬가지로 이 트리의 크기도 메모리 내에 들어올 수 있는 작은 크기이어야 하므로 가지치기 과정이 수행된다. 선택도를 추정할 때는 문자열 패턴을 부분 문자열들로 쪼갠 뒤 각 부분 문자열의 카운트 값을 조합하여 추정한다. 그런데, 카운트 서픽스 트리는 염색체 서열과 같이 매우 긴 문자열들에 대해서는 적용하기가 어렵다. 염색체 서열들의 서픽스들을 트리에 삽입하기도 어렵거니와 설사 삽입했다고 할지라도 공간상의 제약으로 나중에 가지치기될 가능성이 높기 때문이다. 그러므로, 생성 도중의 트리 크기가 대단히 크고, 생성 시간이 대단히 오래 걸린다는 문제점이 생긴다.

이 논문에서는 그림 2와 같이 매우 긴 문자열들로 이루어진 데이타베이스가 주어졌을 때 질의 문자열의 선택도를 추정하는 문제를 해결하기 위해, 긴 서픽스들을 트리에 삽입하는 대신 일정한 길이의 부분 문자열(q-gram)[12, 13]들만을 슬라이딩 윈도우 방식으로 트리에 삽입하는 카운트 큐그램 트리를 제안한다.

이 논문의 구성은 다음과 같다. 2절에서 관련 연구들을 살펴보고, 3절에서 새로운 자료구조인 큐그램 트리에 대한 구조와 크기 등을 살펴본다. 그리고 4절에서 실험 평가를 하고 5절에서 결론을 짓는다.

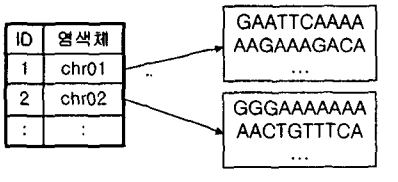


그림 2. 생물학 서열 데이타베이스

2. 관련 연구

문자열 애트리뷰트에 대한 선택도를 추정하는 문제는 [3]에서 처음 제기되었고, [4]에서 발전되었다. [3]과 [4] 모두 두 단계로 문제를 해결하고 있는데, 첫 번째는 문자열들을 읽어들이며 카운트 서픽스 트리를 생성하는 단계이다. 이 트리는 부분 문자열들이 나타나는 회수가 기록되지만, 공간적 제약으로 인해 임계값 p 미만의 노드들은 가지치기된다. 두 번째 단계는 질의 처리시 카운트 서픽스 트리를 이용하여 질의 문자열을 포함한 투플들의 수를 추정하는 단계이다. 만약, 카운트 서픽스 트리가 질의 문자열을 포함한다면 정확한 선택도를 알 수 있지만, 그렇지 않다면 트리에 포함된 정보만을 가지고 추정을 해야 한다.

[3, 4]에서 사용한 카운트 서픽스 트리는 서픽스 트리[9]와 두 가지 면에서 다르다. 서픽스 트리는 각 단말 노드에서 해당 문자열이 나타나는 지점을 가리키는 포인터를 가지고 있지만, 카운트 서픽스 트리는 이 포인터들을 가지고 있지 않다. 그리고 카운트 서픽스 트리는 각 노드(중간 노드와 단말 노드 모두)에 문자열이 나타나는 회수를 기록하는 카운트 항목을 추가

하였다.

3. 큐그램 트리

3.1. 구조

큐그램 트리는 삽입하는 문자열이 다르다는 점을 제외하고는 카운트 서픽스 트리와 동일하다. 카운트 큐그램 트리는 모든 서픽스를 트리에 삽입하는 대신, 큐그램들을 슬라이딩 윈도우 방식으로 삽입하여 트리를 구성한다. 그림 3은 삽입 예를 보여준다. 문자열 "AAACTTA"에 대해 카운트 서픽스 트리를 구성하면 "AAACTTA", "AACTTA", ..., "A"가 삽입된다. 반면에 q가 4인 카운트 큐그램 트리는 문자열의 마지막에 q-1개의 종료문자(#)를 덧붙인 뒤, 큐그램들("AAAC", "AACT", ..., "A###")을 삽입한다. 실제로 구현을 할 때는 # 문자를 이용한 변환 과정이 필요없지만, 설명을 쉽게 하기 위해 첨가하였다. 하나의 큐그램이 삽입되는 과정은 서픽스 트리와 마찬가지로 루트로부터 동일한 문자들을 따라오다가 동일한 문자가 없어지는 순간에 새로운 노드를 만들게 된다. 이 과정 중 만나는 노드들마다 카운트 값을 1 증가시키고, 새로 만들어지는 노드는 1로 설정한다.

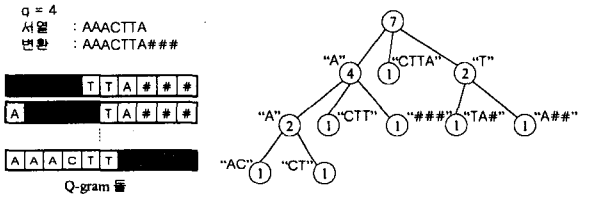


그림 3. 큐그램 트리

위와 같은 과정에 의해 생성된 큐그램 트리는 길이가 q 이하인 모든 부분 문자열들의 정확한 카운트 값을 가지고 있다. 그림 4는 이 특징을 보여주고 있다.

q = 4
문자열 = AAACTTA, 변환 문자열 = AAACTTA###

S[1:4]	S[2:5]	S[3:6]	....	S[7:10]	
AAAC	AACT	ACTT		A###	→ 길이 4인 부분문자열
AAA	AAC	ACT		A##	→ 길이 3인 부분문자열
AA	AA	AC		A#	→ 길이 2인 부분문자열
A	A	A		A	→ 길이 1인 부분문자열

그림 4. 큐그램 트리의 카운트 값들

3.2. 큐그램의 길이와 트리의 크기

큐그램 트리의 루트로부터 모든 단말 노드까지에 있는 문자열의 개수는 삽입된 큐그램의 길이 q와 동일하다. 이 때, 매 글자마다 노드가 생기고 모든 글자들에 대해 분기가 생기면 트리 크기가 최대가 된다. 이 트리는 높이가 q이고 팬아웃이 D인 완전 트리로 크기는 다음과 같다.

$$\frac{D^q-1}{D-1}\times I + D^q \times L \ ,$$

(I: 중간 노드의 크기, L: 단말 노드의 크기)

정해진 메모리의 크기가 M이라면, 트리는 M보다 작거나 같아야 한다(식 (1)). 식 (1)을 q에 관해 정리하면 식 (2)를 얻게 되는데, 식 (2)를 만족하는 최대값으로 q를 설정하면 된다.

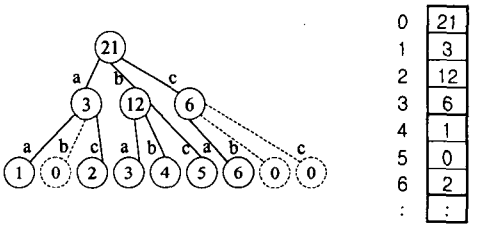
식 (1) : $\frac{D^q-1}{D-1}\times I + D^q \times L \le M$

식 (2) : $q \le \log_D \frac{M+I}{LD-L+I}$

식 (2)에서 주목할 점은 큐그램 트리의 크기는 문자열의 길이 N과 상관없다는 점이다. 즉, 아무리 문자열 길이가 길고, 그러한 문자열들의 개수가 많아도 트리의 크기는 O(Dq)로 한정될 수 있다. 만약, 카운트 값이 큰 노드들 위주로 저장하고 싶다면, q를 식 (2)를 만족하는 값보다 크게 설정한 후 가지치기를 하는 것도 가능하다.

3.3. 구현

실제로 큐그램 트리를 구현할 때, [3]에서 설명한 카운트 서픽스 트리의 구현 방법이나 서픽스 트리의 압축 기법[10]을 사용할 수 있다. 그러나, 서픽스 트리 계열은 많은 포인터들이 필요하고, 각 가변 길이 문자열들을 저장하고 있어야 하며, 문자열 삽입이나 추정을 위해 트리를 탐색할 때마다 가변 길이 문자열들과 비교하는 연산이 필요하다.



(a) 완전 트리 형태의 큐그램 트리 (b) 1차원 배열

그림 5. 큐그램 트리의 구현

DNA 서열처럼 도메인의 크기가 작으며 길이가 긴 서열들을 대상으로 큐그램 트리를 생성할 경우 완전 트리 또는 이에 가까운 트리가 생겨나기 쉽다. 이 성질을 이용하면 그림 5(b)와 같이 1차원 배열로 작고 빠르게 트리를 생성할 수 있다. 부모 노드의 위치가 m일 때, 자식 노드들은 $mD+i$ 이다. 이 때, i는 각 글자의 도메인 내의 순서로, $1 \le i \le D$이다. 그림 5(a)에서 "ac"는 'a'의 위치 1에 3을 곱한 뒤 'c'에 해당하는 3을 더한 6에 카운트 값이 저장된다. 1차원 배열 구현 방식에서는 문자열 공간과 포인터 공간이 절약되며, 문자열 비교가 없으므로 탐색이 빨라진다. 만약, 완전 트리에 가깝지 않다면 그림 5(a)의 "ab"나 "cb"처럼 카운트 값이 0이 되는 노드들이 많이 생겨서, 배열의 많은 부분이 낭비되는 문제점이 발생할 수 있다. 1차원 배열에 의한 트리 생성 시간은 큐그램들마다 q번의 배열 갱신이 이루어지므로 $O(q \times N)$이다.

4. 실험

미 국립 생명공학 정보센터[11]가 제공하는 호모사피엔스 염기서열 1과 2 데이타에 대해서, 큐그램 트리의 크기와 생성 시간을 측정하고, 부분 문자열 발생 빈도를 추정하였다. 염기 서열은 기본적인 염기 A, C, G, T와 미확인 염기인 N으로 구성된다. 즉, 도메인의 크기가 5이다. 실험은 PIII-1GHz CPU, 768MB 메모리가 장착되고, WOWLiNUX Release 7.1이 운영체제로 설치된 컴퓨터에서 C++로 구현하였다.

4.1. 큐그램 트리의 크기와 생성 시간

큐그램의 길이	1	2	3	4	5	6
트리 크기	24	244	624	3k	15k	78k
생성 시간	58	66	73	80	91	102

큐그램의 길이	7	8	9	10	11
트리 크기	390k	1.9M	9M	48M	244M
생성 시간	113	148	184	237	303

1차원 배열로 구현한 카운트 큐그램 트리의 사용 공간과 생성 시간은 위의 표에 보여진다. 사용 공간의 정확한 값은 배열의 크기 $(D^{q+1}-1)/(D-1)$ 과 카운트의 데이타형 크기(정수, 4바이트)를 곱하여 계산할 수 있다. 생성 시간은 $O(qN)$으로 N이 고정되었을 때, q값에 비례한다.

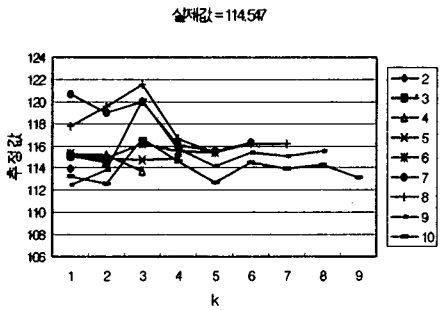


그림 6. 발생 빈도의 예측값

4.2. 큐그램 트리를 이용한 발생 빈도 예측

큐그램 트리를 이용한 발생 빈도 예측의 정확도를 평가하기 위해, 길이가 11인 질의 문자열 1000개를 임의로 생성하였고, q 값을 변화시키며 1-그램 트리에서 10-그램 트리까지 생성하였다. 그리고, 각 트리마다 예측의 정확성을 평가하였다.

그림 6은 각 트리의 발생 빈도를 예측한 결과를 보여준다. 범례는 q 값이 2부터 10까지인 카운트 큐그램 트리를 의미한다. 실제값과 추정값 모두 1000개의 질의 문자열에 대한 평균값이다. 큐그램 트리의 종류와 k 값에 따라 조금씩의 차이는 있었지만, 대부분의 경우에서 거의 실제값에 가깝게 추정하고 있음을 보여주었다.

5. 결론

수년 전부터 문자열 데이타에 대한 발생 빈도를 예측하는 문제가 연구되었다. 지금까지는 문자열 데이타로부터 카운트 서픽스 트리를 생성한 다음, 이 트리를 이용하여 발생 빈도를 예측하였다. 그런데, 카운트 서픽스 트리는 본질적으로 DNA 서열처럼 길이가 매우 긴 문자열 데이타들로 이루어진 데이타베이스에는 적합하지 않다. 수백 메가바이트에 달하는 서열에 대해 모든 서픽스를 삽입하는 과정 자체가 공간과 시간 모두 비효율적인 연산을 가져오게 된다.

이 문제를 해결하기 위해, 이 논문에서는 모든 서픽스를 삽입하는 대신, 길이가 q인 부분 문자열들만을 삽입하는 큐그램 트리를 제안하였다. 큐그램 트리는 길이가 q 이하인 모든 부분 문자열들의 정확한 등장 회수를 저장하고 있는 완전성을 지니고 있으며, 일정한 길이의 부분 문자열들만을 삽입하기 때문에 적은 메모리 한계 내에서도 빠른 생성이 가능하다. 또한, 1차원 배열에 의한 트리 구현이 가능한데, 문자열과 포인터를 저장할 필요가 없고 빈번한 동적 메모리 할당없이 공간을 절약할 수 있으므로 빠른 생성이 가능하였다.

큐그램 트리의 정확성을 보여주는 실험 결과, 도메인이 커지고 q가 커질수록 트리의 크기는 증가하였지만, 수메가 또는 수십메가 이내에서 DNA 서열들에 대한 충분한 통계적 정보를 저장할 수 있었고, 매우 정확하게 부분 문자열들의 발생빈도를 예측할 수 있음을 보여주었다.

참고 문헌

[1] P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. VLDB, pp 311-322, 1995

[2] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. ACM SIGMOD, 1996

[3] P. Krishnan, J. S. Vitter, and B. Iyer. Estimating alphanumeric selectivity in the presence of wildcards. ACM SIGMOD, pp 282-293, 1996

[4] H. V. Jagadish, R. T. Ng, and D. Srivastava. Substring selectivity estimation. ACM Symposium on Principles of Database Systems, June 1999

[5] H. V. Jagadish, O. Kapitskaia, R. T. Ng, and D. Srivastava. Multi-dimensional substring selectivity estimation. VLDB, 1999

[6] H. V. Jagadish, R. T. Ng, and D. Srivastava. On effective multi-dimensional indexing for strings. ACM SIGMOD, pp 403-414, 2000

[7] P. Ferragina, N. Koudas, S. Muthukrishnan, and D. Srivastava. Two-dimensional substring indexing. ACM Symposium on Principles of Database Systems, 2001

[8] Z. Chen, H. V. Jagadish, F. Korn, N. Koudas, S. Muthukrishnan, R. T. Ng, and D. Srivastava. Counting twig matches in a tree. ICDE, pp 595-604, 2001

[9] E. M. McCreight. A space-economical suffix tree construction algorithm. Journal of the ACM, Vol 23, pp 262-272, 1976

[10] R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. ACM Symposium on Theory of Computing, pp 397-456, 2000

[11] ftp://ncbi.nlm.nih.gov/genomes/H_sapiens/, 2001

[12] E. Ukkonen. Approximate string-matching with q-grams and maximal matches. Theoretical Computer Science, Vol 92, pp 192-211, 1992

[13] G. Navarro and R. Baeza-Yates. A practical q-gram index for text retrieval allowing errors. CLEI Electronic Journal, Vol 1, No 2, 1998