DNA 시퀀스 데이터베이스를 위한 저장-효율적인 Trie 인덱싱 기법

김강모°, 서남호*, 원정임*, 윤지희*, 박상현**, 김상욱***
*한림대학교 정보통신공학부, **연세대학교 컴퓨터과학과, ***한양대학교 정보통신학부
{hallym,kara1115,jiwon,jhyoon}@hallym.ac.kr, sanghyun@cs.yonsei.ac.kr, wook@ihanyang.ac.kr

A Storage-Efficient Trie Indexing Method for DNA Sequence Databases

Kang-Mo Kim*, Nam-Ho Seo*, Jung-Im Won*, Jee-Hee Yoon*, Sang-Hyun Park**, Sang-Wook Kim***
*Div. of Information Engineering and Telecommunications, Hallym University
**Dept. of Computer Science, Yonsei University
***Div. of Information and Communications, Hanyang University

요 약

대규모 DNA 시퀀스를 대상으로 하여 서브시퀀스를 고속으로 검색하기 위한 인덱싱 방법으로서 접미어 트리가 유용하다. 그러나 접미어 트리는 데이터 크기의 약 100배에 해당하는 방대한 저장 공간을 필요로 한다. 본 논문에서는 기존 접미어 트리의 검색 성능을 유지하며, 저장 공간을 획기적으로 감소시킬 수 있는 새로운 인덱스 구조를 제안한다. 제안된 인덱싱 방안에서는 DNA 시퀀스 내의 모든 염기 위치에 고정 길이의 슬라이딩 윈도우를 위치시켜, 윈도우 크기에 해당하는 연속된 서브시퀀스를 추출한 후, 이들을 대상으로 트라이를 구성한다. 트라이는 저장 공간 감소를 위하여 각 문자를 최소 비트 정보로 표현하며, 저장 구조로서 포인터를 사용하지 않는 디스크 기반의 이진 트라이 구조를 사용한다. DNA 서브시퀀스 검색을 효율적으로 처리하기 위한 인덱스 기반의 질의 처리 알고리즘을 제안하고 실험을 통하여 그 유용성을 보인다. 제안된 인덱스는 접미어 트리의 약 10분의 1의 저장 공간을 필요로 하며, 데이터 크기 증가에 거의 영향을 받지 않는 안정된 고속 검색 성능을 지원한다.

1. 서 론

DNA 시퀀스 데이터베이스는 매우 큰 용량을 가진다. DNA 시퀀스 데이터베이스의 하나인 GenBank[1]는 초기인 1992년 약 1억의 염기들로 구성된 7만 8천여 개의 DNA 시퀀스들을 가졌으나, 2003년 현재 약 365억 염기들로 구성된 3천만여 개의 DNA 시퀀스들을 가지는 거대한 규모로 성장하였다. 이와 같은 DNA 시퀀스 데이터베이스 규모의 급격한 증가 추세를 고려할 때, DNA 시퀀스 검색 연산을 보다 효과적으로 지원하기 위한 인덱싱 기술이 요구된다.

접미어 트리(suffix tree)[2]는 DNA 시퀀스 검색을 위한 좋은 인덱스 구조로 알려져 왔다. 접미어 트리는 주어진 시퀀스의 접미어에 해당되는 서브시퀀스들을 트리 형태로 구성한 것으로서 서브시퀀스 검색의 효율이 우수하다. 그러나 접미어 트리는 그 구조적 특성으로 인하여 저장 공간의 오버 헤드가 매우 큰 단점을 갖는다. 참고 문헌 [3]에 의하면, 286M 염기를 가지는 DNA 시퀀스를 대상으로 구성된 접미어 트리의 크기는 19G 바이트로 나타났다.

본 논문에서는 이와 같은 저장 공간 문제를 해결하는 DNA 시퀀스 검색을 위한 새로운 인덱스 구조를 제안한다. 제안된 인덱스는 트라이(trie)[2]를 기본 구조로 사용하며, DNA 시퀀스 내의 모든 염기 위치에 고정 길이의 슬라이딩 윈도우를 위치시켜, 윈도우 크기에 해당하는 연속된 서브시퀀스를 추출한 후, 이들을 대상으로 트라이를 구성한다. 또한 저장 공간 감소를 위하여 각 문자를 최소 비트 정보로 표현하며, 저장 구조로서 포인터를 사용하지 않는 디스크 기반의 이진 트라이 구조를 갖는다. 질의 처리는 인덱스 검색 과정과 착오 채택(false alarm) 제거를 위한 후처리 과정으로 이루어진다. 실험 결과에 의하면, 제안된 인덱스 구조는 기존 접미어 트리에 비하여 약 10분의 1의 저장 공간(데이터 크기의 약 10배의 저장 공간)을 필요로 하며, 데이터 크기 증가와 질의 길이 변화에 거의 영향을 받지 않는 안정된 고속 검색 성능을 지원하는 것으로 나타났다.

2. 인덱싱 방안

2.1 트라이

트라이는 다수의 시퀀스들을 인덱싱하기 위하여 사용되며, 주어진 질의 시퀀스와 일치하는 시퀀스 내의 서브시퀀스 위치를 신속하게 찾는데 유용하다. 트라이를 구현하는 효율적인 방법 중 하나로 포인터 없는 이진 트라이(pointerless binary trie)[4][5]를 생각해 볼 수 있다. 포인터 없는 이진 트라이는 알파벳 Σ를 {0,1}로 제한하여 각 노드가 최대 2개의 에지를 가지도록 하며, 0의 값을 가지는 에지는 노드의 왼편에, 1의 값을 가지는 에지는 노드의 오른편에 연결하는 규칙을 적용하여 에지 정보를 생략 표현한다. 즉, 노드 당 두 비트를 할당하여 그 값이 '10'이면 노드에 왼쪽 에지만이 연결된 형태를 표현하고, '01'은 노드에 오른쪽 에지만이 연결된 형태를 표현하고, '11'은 노드에 왼쪽 에지와 오른쪽 에지가 모두 연결된 형태를 표현하고, '00'은 에지가 연결되지 않은 단말 노드의 형태를 표현한다.

2.2 인덱스 구성

DNA 시퀀스의 고속 검색을 지원하기 위하여 대상이 되는 DNA 시퀀스의 모든 접미어를 추출하여 이들을 인덱싱 대상으로 하는 접미어 트리(접미어 트라이)를 생

※ 본 연구는 한국과학재단 목적기초연구 (과제번호: R04-2003-000-10048-0) 지원으로 수행되었음.

각할 수 있다. 접미어 트리는 인덱싱 대상이 되는 접미어들이 많은 공통 접두어 서브시퀀스를 가질 때 좋은 압축 효과를 갖는다. 그러나 DNA 시퀀스에서는 이 들 접미어 시퀀스들 사이의 공통 접두어 서브시퀀스의 길이가 비교적 짧다. 즉, 대부분의 접미어 시퀀스는 일정 길이(6-8 염기) 이상의 공통 접두어 서브시퀀스를 거의 가지지 않으므로 접미어 트리의 압축 효율이 좋지 못하다. 본 연구에서는 DNA 시퀀스 내의 모든 염기 위치에 고정 길이의 슬라이딩 윈도우를 위치시켜, 윈도우 크기에 해당하는 연속된 서브시퀀스를 추출한 후, 이 들을 인덱싱 대상으로 한다. 이 후 DNA 시퀀스에서 각 윈도우 크기에 해당하는 연속된 서브시퀀스를 윈도우 시퀀스라고 칭한다. 윈도우 크기는 DNA 시퀀스 상의 대부분의 접미어 시퀀스가 가지는 공통 접두어의 최대 길이를 가정한다. 이와 같은 가정은 제안된 인덱스 구조가 접미어 트리가 가지는 기본적인 특징을 그대로 유지할 수 있음을 의미한다. 또한 인덱스 구조로서 포인터를 사용하지 않는 이진 트라이의 기본 개념을 활용하며, 보다 높은 인덱스 압축 효율을 구현하기 위하여 DNA 시퀀스가 소수의 문자만으로 구성된 시퀀스라는 점에 착안하여 시퀀스 내에 출현하는 각 문자를 8비트가 아닌 최소의 비트량으로 표현한다. DNA 시퀀스로부터 얻어진 모든 윈도우 시퀀스를 위에서 정의한 이진 트라이 구조에 삽입하여 디스크 기반의 인덱스를 생성하는 과정을 그림 1에 보인다.

단계 1: 윈도우 시퀀스 추출

DNA 시퀀스의 모든 윈도우 시퀀스를 추출한다.

단계 2: 이진 윈도우 시퀀스로 변환

추출된 모든 윈도우 시퀀스에 대해 시퀀스를 구성하는 각 문자당 최소 비트를 할당하여 시퀀스를 이진 비트열로 변환한다. 이를 이진 윈도우 시퀀스라 정의하고, 이를 인덱싱 대상으로 한다. 다음, 변환된 이진 윈도우 시퀀스를 오름차순으로 정렬한다.

단계 3: 트라이 구성

정렬된 이진 윈도우 시퀀스를 다음 과정에 의하여 순서대로 트라이에 삽입하여 이진 윈도우 트라이를 구성한다.

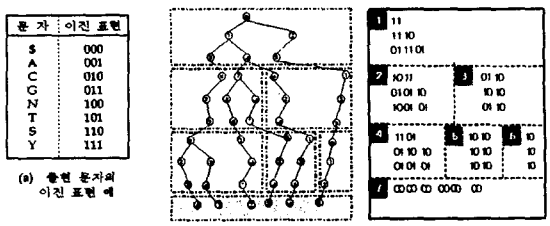
① 시퀀스가 삽입되는 과정에 의하여 새로이 생성 또는 변경되는 노드 구조를 2비트 노드 정보로 해당 페이지 영역에 기록한다. 이 때 한 페이지에 저장할 최대 트리 레벨 수와 최대 노드 수에 의하여 페이지 크기가 결정된다.

② 새로이 삽입되는 노드에 의하여 해당 페이지 영역 내에 오버플로우가 발생할 가능성이 있으면, 이 노드를 제외한 나머지 해당 페이지 영역을 디스크에 기록하고, 기록되지 않은 노드 정보로 해당 페이지 영역을 재구성한다. 디스크에 해당 페이지 영역을 기록할 때, 페이지 정보를 함께 기록한다.

③ 이진 윈도우 시퀀스가 트라이 구조에 삽입된 후, 그 단말 정보를 단말 정보 테이블에 별도로 저장한다.

그림 1. 인덱스 생성 알고리즘

다음의 그림 2는 이와 같은 과정에 의하여 생성된 인덱스의 구성 요소를 보인다. 예를 들어 S1='ACGT'와 S2='ACT'의 두 개의 DNA 시퀀스에 대한 인덱스를 구성하는 경우, 각 시퀀스로부터 모든 윈도우 시퀀스를 추출하여 그림 2-(a)의 방식에 의하여 각 문자당 3비트를 할당하여 이를 이진 시퀀스로 변환하면 그림 2-(b)와 같은 결과를 얻을 수 있다. 이 예에서는 윈도우 크기를 3으로 가정한다. 여기에서 사용된 '$' 문자는 일정 길이의 윈도우 시퀀스를 생성하기 위하여 사용된 특수 문자를 나타낸다. 그림 2-(b)의 각 이진 시퀀스를 정렬하여 이진 트라이 구조에 삽입하면 그림 2-(c)와 같은 이진 트라이 구조와 이진 데이터를 얻을 수 있다. 여기에서 보이는 점선은 디스크 내의 페이지 분할 저장 상황을 나타낸다. 이와 같이 트라이 인덱스는 디스크 페이지에 분할되어 저장되므로, 이 인덱스를 이용하여 임의의 경로를 검색하기 위하여는 노드와 노드 사이의 페이지 연결 상태를 나타내는 정보가 필요하다. 페이지 연결 정보는 페이지 번호(#Page), 현재 페이지 레벨의 이전 페이지까지 유입된 에지의 총 수(Top), 현재 페이지 레벨의 이전 페이지들로부터 나간 에지의 총 수(Bottom) 등으로 이루어지며, 그림 2-(d)는 페이지 정보를 나타내는 페이지 테이블을 나타낸다. 다음, 그림 2-(e)는 트라이에 삽입된 각 윈도우 시퀀스의 단말 정보(시퀀스 번호, 오프셋)를 저장하고 있는 단말 노드 테이블을 나타낸다.



(a) 출현 문자와 이진 표현 예

문 자	이진 표현
$	000
A	001
C	010
G	011
N	100
T	101
S	110
Y	111

(c) 이진 트라이 구조와 내부표현의 예

(b) 추출된 윈도우 시퀀스의 예

윈도우 시퀀스	이진 변환된 윈도우 시퀀스
S1: ACG	001010011
CGT	010011101
GT$	011101000
T$$	101000000
S2: ACT	001010101
CT$	010101000
T$$	101000000

(d) 페이지 테이블

#Page	Top	Bottom	Node	Addr
1	0	0	6	84
2	0	0	8	24
3	2	3	6	108
4	0	0	8	0
5	2	3	6	54
6	4	5	3	132
7	0	0	6	159

(e) 단말노드 테이블

#Seq	Offset
1	0
2	0
1	1
2	1
1	2
1	3
2	2

그림 2. 인덱스 구성 예

3. 질의 처리

그림 3에 트라이 인덱스 T를 이용하여 질의 Q를 만족하는 서브시퀀스들을 검색하는 알고리즘 Search-Trie을 보인다. 알고리즘에서는 페이지 내의 트라이 구조를 파악하기 위하여 다음과 같은 4개의 변수를 이용한다. 다음 레벨의 노드 수를 파악하기 위하여 size 변수를 사용하며, 각 레벨의 마지막 노드 위치를 파악하기 위하여 last 변수를, 다음 레벨의 어느 노드로 이동할 지를 결정하기 위한 srch와 next 변수를 사용한다.

```
Algorithm Search_Trie
Input : Trie index T, Query Q, Page Info P
Output : set of answers
srch := 0, next := 0, size := 1, last := 0;
For (pageLevel:=0; pageLevel < p_Height; pageLevel++) do
   if (pageLevel > 0) then { Page_Change(),
                             reset size, last, next, srch } ;
   For (nodeLevel:=0; nodeLevel<n_Height; nodeLevel++) do
      while (isBefore(next)) do { increase srch, size } ;
      if (!match(node(next), query_bit)) return('No Match') ;
      if (isLast(query_bit)) return(find_answers()) ;
      get(qery_bit), increase srch, size ;
      while (isBefore(last)) do { increase size } ;
      if (nodeLevel < n_Height - 1) then { reset next, srch,
                                           last } ;
```

4. 성능 평가

성능 평가는 다음 두 가지의 서로 다른 기법을 대상으로 한다. Ours는 본 논문에서 제안한 윈도우 시퀀스 기반의 이진 트라이 인덱스 검색 방식이다. 성능 비교를 위한 기법으로서 윈도우 시퀀스 기반의 트리 검색 방식(Tree)을 사용한다. Tree는 DNA 시퀀스로부터 얻어진 모든 윈도우 시퀀스를 기존의 트리 구조에 삽입하여 디스크 기반의 인덱스를 생성하는 방식으로서, 트리 생성에는 참고 문헌 [6]에서 제안된 incremental disk-based 트리 생성 방식을 사용한다. 실험 데이터로는 GenBank[7]로부터 다운 받은 Human Chromosome 18번의 3가지 DNA 시퀀스(2.17Mbp, 14.8Mbp, 32.4Mbp)를 사용하였다. 시퀀스 내에 출현하는 서로 다른 문자수는 7개이다. 실험을 위한 하드웨어 플랫폼으로는 Windows 2000 Server를 운영체제로 사용하고, 1GB의 주기억장치, 40GB 디스크를 갖는 Pentium IV 2GHz의 PC를 사용한다.

실험 1 (인덱스 크기 비교) : 표 1에 인덱스의 크기를 구성 요소별로 분리, 측정한 결과를 보인다. 본 실험에서는 페이지 크기로 4K를 가정하였다. Ours는 이진 트라이 인덱스(페이지 테이블 포함)와 단말 테이블로 이루어지며, 이 중 이진 트라이는 윈도우 크기에 직접적인 영향을 받는다. 여기에서는 윈도우 크기(|w|)가 15인 경우를 나타낸 것이다. 각 기법의 인덱스 크기를 비교하기 위하여 데이터 크기에 대한 인덱스 크기 비율을 그림 4에 보인다.

표 1. 인덱스 크기 비교 (|w|=15의 경우)

Data Size	Ours		Tree
	Trie Index	Leaf Table	
2.17M	6.82M	16.72M	106M
14.8M	30.45M	112.17M	679M
32.4M	54.47M	245.37M	1.41G

실험 결과에 의하면, Ours는 |w|=15의 경우에 데이터 크기의 약 10배의 크기를 가지며, |w|=30의 경우에 데이터 크기의 약 20배의 크기를 가지고, Tree는 |w|=15의 경우에 약 50배의 크기를 가지며, |w|=30의 경우에 약 60배의 크기를 가지는 것으로 나타났다.

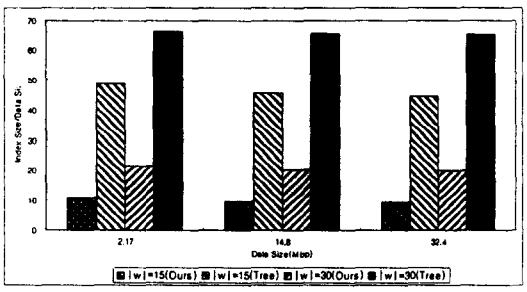


그림 4. 데이터 크기에 대한 인덱스 크기의 비율

실험 2 (질의 처리 시간 비교) : Ours의 질의 처리 시간을 Tree 방식과 비교한 결과를 그림 5와 그림 6에 보인다. 질의 처리 시간은 인덱스 검색 시간과 후처리 시간을 합한 총시간을 의미한다. 그림 5는 질의 길이(|Q|) 변화에 따르는 질의 처리 시간의 변화를 나타낸다. 실험 결과로부터 Tree는 거의 일정한 질의 처리 시간을 나타내고 있으나, 질의 길이가 짧은 경우(|Q|=6)에는 비교적 질의 처리 시간이 증가하고 있음을 알 수 있다. 그러나 Ours는 질의의 길이 변화에 거의 영향을 받지 않는 일정한 질의 처리 시간을 나타내고 있다. 또한 Ours와 Tree에서 윈도우 크기가 30인 경우에 비하여 15의 경우에 오히려 질의 처리 성능이 우수한 것으로 나타났다. 그림 6은 데이터 크기 변화에 따르는 질의 처리 시간의 변화를 나타낸다. Tree는 질의 길이가 짧은 경우(|Q|=6), 데이터 크기가 증가함에 따라 질의처리 시간이 급격히 증가하고 있다. 그러나, Ours는 모든 질의 길이에 대하여 데이터 크기가 증가함에 따라 매우 완만한 질의 처리 시간의 증가 현상을 나타냄을 알 수 있다.

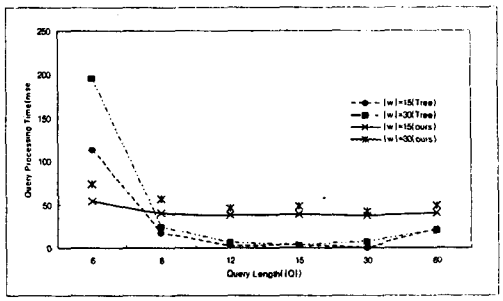


그림 5. 질의 길이 변화에 따른 질의 처리 시간 비교 (데이터 크기: 32.4Mbp)

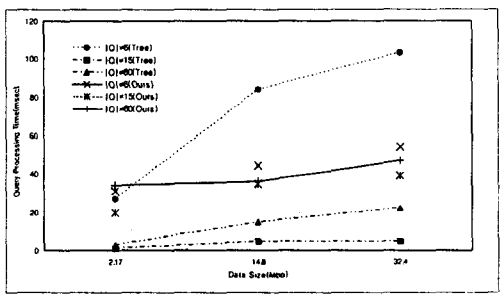


그림 6. 데이터 크기 변화에 따른 질의 처리 시간 비교 (|w|=15의 경우)

참고 문헌

[1] D. A. Benson, M. S. Boguski, D. J. Lipman, J. Ostell, and B. F. Quellette, "Genbank," Nucleic Acids Research, Vol. 26, No. 1, pp. 1-7, 1998.

[2] G. A. Stephen, String Searching Algorithms, World Scientific Publishing, 1994.

[3] E. Hunt, M. P. Atkinson and R. W. Irving, "Database indexing for large DNA and protein sequence collections," The VLDB Journal, Vol. 11, No. 3, pp. 256-271, 2002.

[4] H. Shang and T. H. Merrett, "Tries for approximate string matching," IEEE Trans. on Knowlege and Data Engineering, Vol. 8, No. 4, pp. 540-547, 1996.

[5] 원정임, 박용일, 윤지희, 박상현, "트라이 인덱스를 이용한 DNA시퀀스 검색", 정보과학회 추계학술 발표 논문집, pp. 4-6, 2003.

[6] P. Bieganski, J. Riedl, J. V. Carlis, "Generalized suffix trees for biological sequence data: applications and implementation," In Proccedings of Hawaii International Conference on System Sciences, 1994.

[7] http://www.ncbi.nlm.nih.gov