Succinct 표현의 효율적인 구현을 통한 압축된 써픽스 배열 생성[1)]

박치성° 조준하 김동규
부산대학교 컴퓨터공학과
{cspark°, jhjo, dkkim}@islab.ce.pusan.ac.kr

Constructing the Compressed Suffix Array via Efficient Implementation of Succinct Representation

Chi Seong Park° Junha Jo Dong Kyue Kim
Dept. of Computer Science & Engineering, Pusan National University, Pusan 609-735, Korea

요 약

대용량의 텍스트에 대해 빠른 패턴 검색의 필요성이 증가함에 따라 써픽스 트리, 써픽스 배열 등의 인덱스 자료구조에 대해 다양한 연구들이 진행되었다. 또한 써픽스 배열을 대용량의 인덱스 자료구조로 사용하기 위해 저장 공간을 $O(n\log n)$ 비트 이하로 줄이는 문제에 대한 연구들도 많이 수행되었다. 이들 중 Grossi & Vitter는 써픽스 배열을 압축하여, 기존의 써픽스 배열보다 작은 저장 공간을 사용할 수 있는 알고리즘을 제안하였다. Grossi & Vitter 알고리즘은 압축된 써픽스 배열에서 실제 써픽스 배열의 정보를 찾기 위하여, succinct 표현에서 기본적으로 사용되는 rank와 select 함수를 필요로 한다. 본 논문은 다양한 rank와 select 알고리즘을 각각 사용하는 압축된 써픽스 배열들의 성능 비교를 통해, succinct 표현의 효율적인 구현이 압축된 써픽스 배열의 성능에 미치는 영향을 실험적으로 보인다.

1. 서 론

대용량의 텍스트에 대해 빠른 패턴 검색의 필요성이 증가함에 따라 써픽스 트리[1, 2, 3, 4], 써픽스 배열[5, 6, 7, 8, 9] 등의 인덱스 자료구조에 대해 다양한 연구들이 진행되었다. 써픽스 배열은 써픽스 트리에 비해 구조가 간단하고 저장 공간을 적게 차지하는 장점에도 불구하고 문자집합 Σ 길이가 n인 스트링에 대하여 $O(n\log n)$ 비트의 저장 공간을 필요로 한다.

따라서, 써픽스 배열을 대용량의 인덱스 자료구조로 사용하기 위해 저장 공간을 줄이는 문제에 대한 연구들이 많이 수행되었으며 이들 중 Grossi & Vitter[10]는 써픽스 배열을 압축하여 $O(n\log n)$ 비트보다 작은 저장 공간을 사용할 수 있는 알고리즘을 제안하였다.

Grossi & Vitter 알고리즘은 $l=\log\log_{|\Sigma|}n$ 단계 동안 succinct representation을 사용하는 부가적인 자료구조들을 통해 이전 단계의 써픽스 배열을 절반 크기로 압축해 나가는 방법으로써, 실제 써픽스 배열의 정보를 찾기 위해 succinct 표현에서 기본적으로 사용되는 rank와 select 함수[11, 12, 13, 14, 15]를 필요로 한다. 이 때, 압축된 써픽스 배열을 기반으로 압축에 사용된 부가적인 자료구조들에 l 단계에 비례하는 횟수만큼의 rank와 select 함수를 사용하여 실제 써픽스 배열의 정보를 찾기 때문에 효율적인 rank와 select 함수의 구현은 압축된 써픽스 배열의 성능에 큰 영향을 미친다.

본 논문은 Kim et al.[15]에서 언급된 4가지 rank와 select 알고리즘을 각각 사용하는 Grossi & Vitter의 압축된 써픽스 배열들의 성능 비교를 통해, succinct 표현의 효율적인 구현이 압축된 써픽스 배열의 성능에 미치는 영향을 실험적으로 보인다. 이를 위해 비트 1의 비율과 비트 스트링의 길이에 따른 4가지 rank와 select 알고리즘의 성능을 비교한 후, real-world 스트링과 랜덤 스트링에 대하여 Grossi & Vitter의 압축된 써픽스 배열에 4가지 rank와 select 알고리즘을 각각 적용한 압축된 써픽스 배열 4가지의 성능을 비교한다.

본 논문은 총 3장으로 구성된다. 2장에서 succinct 표현에서 기본적으로 사용되는 rank와 select 함수와 Grossi & Vitter의 압축된 써픽스 배열을 살펴본 후, 3장에서 두 가지 실험을 통해 4가지 rank와 select 함수의 성능과, 4가지 rank와 select 함수를 각각 사용하는 압축된 써픽스 배열 4가지의 성능을 비교한다.

1) 이 논문은 교육인적자원부 지방연구중심대학육성사업(차세대물류IT기술연구사업단)의 지원에 의하여 연구되었음.

2. 배경 지식

2.1 succinct 표현(succinct representation)

succinct 표현은 n개의 이산객체를 $O(n)$ 비트로 표현하는 방법이다. 컴퓨터 상에서 하나의 정수로 표현해야 했던 하나의 객체를 1 비트로 표현함으로써 공간 효율성을 높이지만 표현만으로는 이산 객체의 개별적 위치를 상수 시간에 파악할 수 없다. 따라서 이를 상수 시간에 가능케 하는 rank와 select 함수가 필요하다. 비트 스트링 A에서 rank와 select 함수는 각각 다음과 같이 정의된다.

$rank_A(x)$: A의 처음부터 인덱스 x까지의 비트 1의 수
$select_A(y)$: A의 처음부터 y번째 비트 1의 인덱스

Clark[13]은 $O(\log n)$ 시간 복잡도의 rank와 select 함수를 제안했던 Jacobson[11, 12]의 2-level directory 구조에 $o(n)$ 비트의 lookup-table을 추가해 이론적으로 상수 시간 복잡도의 rank와 select 알고리즘(CK)를 제안하였으나, 실제 비트 1의 비율이 10% 이하인 경우 select의 성능이 매우 좋지 않았다. Kim et al.은 동일한 저장 공간 $o(n)$ 비트로 Clark의 select 함수의 단점을 개선한 알고리즘(A1)과, 이론적으로는 $n+o(n)$ 비트를 사용하지만 실제로는 A1보다 적은 저장 공간으로 더 좋은 성능을 보이는 알고리즘(A2), 그리고 범용 컴퓨터를 위해 A2의 비트 단위 연산을 바이트 단위 연산으로 변환한 알고리즘(A3)을 제안하였다.

2.2 압축된 써픽스 배열(compressed suffix array)

Grossi & Vitter 알고리즘은 문자집합 Σ 길이가 n인 스트링의 써픽스 배열을 입력으로 $(1+\log\log_{|\Sigma|}n/2)n\log|\Sigma|+O(n)$ 비트 또는 $(1+\varepsilon^{-1})n\log|\Sigma|+o(n\log|\Sigma|)$ 비트의 저장 공간을 사용하는 압축된 써픽스 배열을 구축한다. $O(n\log|\Sigma|)$ 구축 시간이 소요되는 첫 번째 압축 방법의 경우 $k(1\le k\le l=\log\log_{|\Sigma|}n)$ 단계 동안 각 단계마다 B_k 이진 배열[10]과 $Rank_k$. 이진 배열

[10], 그리고 별도의 압축이 필요한 Ψ_k 배열[10], 이렇게 3가지의 부가적인 자료구조를 통해 이전 단계의 써픽스 배열을 절반 크기로 압축해 나간다. 또한 압축된 써픽스 배열에서 i ($1 \le i \le n$)번째 써픽스 배열의 값을 찾는 lookup 함수는 $Rank_k$ 배열과 압축된 Ψ_k 배열에 rank와 select 함수를 사용하여 이전 단계의 값을 찾아간다. 이론적으로 상수 시간 복잡도의 rank와 select 함수가 사용된다면 lookup(i)는 $O(\log \log n)$ 시간 복잡도를 가진다.

3. 실험 및 결과

본 장에서는 앞서 언급된 4가지의 rank와 select 알고리즘과 Grossi & Vitter의 압축된 써픽스 배열을 사용하는 두 가지의 실험을 통해 succinct 표현의 효율적인 구현을 통한 압축된 써픽스 배열의 성능 향상을 실험적으로 보인다. 우선 비트 1의 비율과 비트 스트링의 길이에 따른 4가지 rank와 select 알고리즘의 성능을 비교 실험한 후, real-world 스트링과 랜덤 스트링에 대하여 Grossi & Vitter의 압축된 써픽스 배열에 4가지 rank와 select 알고리즘을 각각 적용한 압축된 써픽스 배열 4가지의 성능을 비교 실험하였다. 모든 실험은 펜티엄 IV 2.80GHz CPU, 2GB 메모리, Windows XP 환경에서 Microsoft Visual C++ 6.0로 구현된 코드를 사용하였다.

3.1 실험 1: 임의의 비트 스트링에 대한 rank와 select 알고리즘의 성능

1, 10, 30, 50(Mbit)의 길이에 비트 1의 비율을 1, 3, 5, 10, 20, 25, 30, 40, 50, 60, 70, 75, 80, 90(%)의 14가지로 달리한 임의의 98개 비트 스트링을 입력으로 4가지 rank와 select 알고리즘의 자료구조 구축 시간과 각각 10^7번의 rank retrieval 시간과 select retrieval 시간을 측정하였다.

실험 결과 자료구조 구축 시간의 경우 스트링의 길이가 길수록, 비트 1의 비율이 높아질수록 느려지며 모든 경우 A3가 가장 빠르고 A2, A1, CK의 순이었다. rank retrieval 시간은 역시 모든 경우 A3가 가장 빠르고 같은 rank 알고리즘을 사용하는 CK와 A2의 성능이 유사하며 A1이 가장 느렸다. select retrieval 시간 역시 모든 경우에서 A3가 가장 빨랐으며 A2, A1, CK의 순이며, 특히 CK의 경우 비트 1의 비율이 커질수록 A1과 유사해지는 특징을 보였다. 그림 1, 그림 2, 그림 3은 각각 비트 1의 비율이 30%일 때 4가지 rank와 select 알고리즘의 자료구조 구축 시간과 rank retrieval 시간, select retrieval 시간에 대한 그래프이다. 그래프의 가로축은 스트링의 크기(단위 : Mbit), 세로축은 시간(단위 : 초)을 나타낸다.

3.2 실험 2: 사용되는 rank와 select 알고리즘에 따른 압축된 써픽스 배열의 성능

DNA, protein, real data 등 18개의 real-world 스트링과 1, 10, 30, 50(Mbit)의 길이에 $|\Sigma|=\{2,4,8,16,20,32,64,128\}$의 8가지 문자집합 크기를 가지는 32종류의 랜덤 스트링을 입력으로 Grossi & Vitter의 압축된 써픽스 배열에 4가지 rank와 select 알고리즘을 각각 적용한 4가지 압축된 써픽스 배열의 압축 시간과 10^6번의 lookup 시간을 측정하였다.

이 때 써픽스 배열의 압축시 rank와 select를 위한 자료구조가 구축되고 lookup시 rank와 select retrieval이 사용되므로 압축된 써픽스 배열의 구축 시간은 rank와 select를 위한 자료구조 구축 시간을 반영하고 lookup 시간은 rank retrieval 시간과 select retrieval 시간을 반영한다. 이를 보이기 위해 압축된 써픽스 배열의 구축 시간과 lookup 시간에서 실제 rank와 select 알고리즘과 관련된 시간을 별도 측정하였다.

실험 결과 압축 시간의 경우 real-world 스트링과 랜덤 스트링 모두에서 A3를 사용하는 Grossi & Vitter 알고리즘의 성능이 가장 우수했으며 A2, CK, A1의 순이었다. 실험 1의 rank와

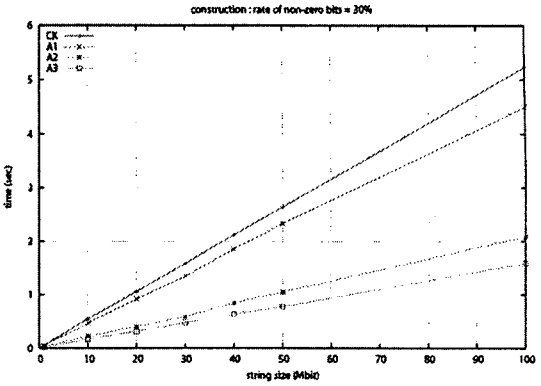

그림 1. 비트 1의 비율이 30%일 때 rank와 select 자료구조 구축 시간

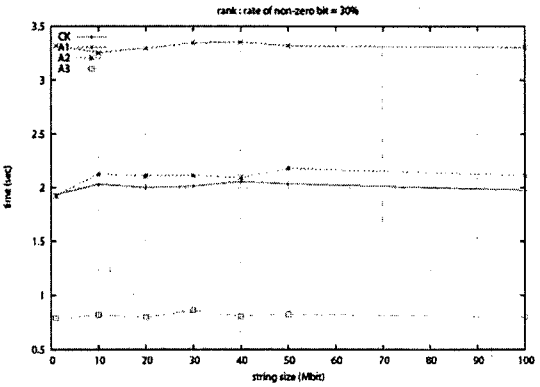

그림 2. 비트 1의 비율이 30%일 때 rank retrieval 시간

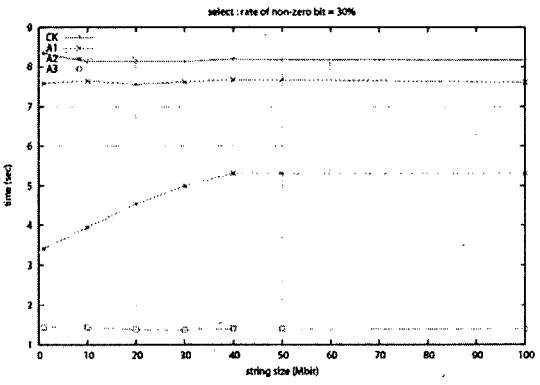

그림 3. 비트 1의 비율이 30%일 때 select retrieval 시간

select 자료구조 구축 시간과 달리 CK보다 A1이 느린 이유는 비트 1의 분포 차이 때문이다. 실제 실험 1에서 사용하였던 비트 스트링의 경우 비율 내에서 비트 1이 스트링 내에 고르게 분포되어 있던 것과 달리, 압축된 Ψ_k 배열의 경우 특성 상 비트 1이 스트링의 뒤쪽에 몰려있는 형태이다.

lookup 시간의 경우 역시 real-world 스트링과 랜덤 스트링 모두에서 A3를 사용하는 Grossi & Vitter 알고리즘의 성능이 가장 우수했으며 CK와 A2가 유사하고 A1이 가장 느렸다. 이는 lookup시 select retrieval보다 rank retrieval의 사용 빈도가 높기 때문이다. 또한 Grossi & Vitter 알고리즘 내에서 rank와 select가 필요한 이진 배열 B_k의 비트 1의 비율은 매 단계 50%이며 압축된 Ψ_k 배열의 비트 1의 비율은 20-50% 사이이므로

select retrieval의 경우 CK와 A1의 성능 차이는 그림 3과 같이 크지 않다.

그림 4, 5는 각각 real-world 스트링 중 $|\Sigma|=5$, 압축된 Ψ_k 배열의 비트 1의 비율이 34.92%인 31,031,233 바이트 길이의 DNA 스트링 month.est_mouse에 대한 압축 시간과 lookup 시간이며 그림 6, 7은 각각 $|\Sigma|=128$, 압축된 Ψ_k 배열의 비트 1의 비율이 42.57%인 50Mbit의 랜덤 스트링에 대한 압축 시간(단위 : 초)과 lookup 시간(단위 : 초)이다. 그래프 막대의 회색 부분이 실제 rank와 select 알고리즘과 관련된 시간이며 흰색 부분이 그 외의 시간이다. 따라서 두 부분을 합친 전체 막대가 실제 Grossi & Vitter 알고리즘의 성능이다.

4. 결론

본 논문은 succinct 표현의 효율적인 구현을 통해 압축된 써픽스 배열의 성능 향상이 가능함을 실험적으로 보였다. 압축된 써픽스 배열은 실제 써픽스 배열의 정보를 얻기 위해 반복된 rank와 select retrieval로 이루어지는 lookup 함수를 사용하며 실험을 통해 rank와 select 함수의 성능이 곧 압축된 써픽스 배열의 성능에 직결됨을 알 수 있었다. 저장 공간을 줄이기 위해 succinct 표현을 사용해야 하는 경우 알고리즘의 성능 향상을 위해 succinct 표현의 효율적인 구현을 고려할 필요가 있다.

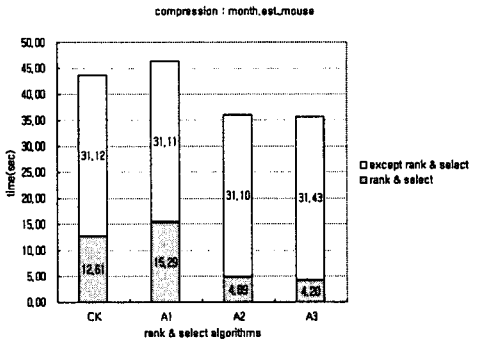

그림 4. month.est_mouse 스트링에 대한 써픽스 배열 압축 시간

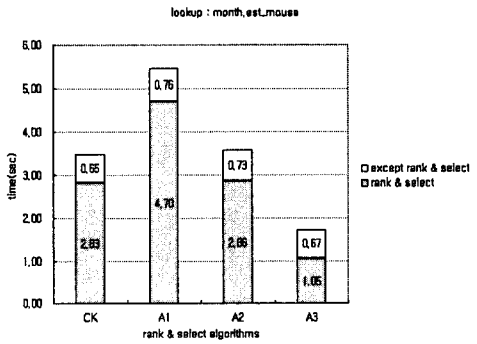

그림 5. month.est_mouse 스트링에 대한 압축된 써픽스 배열 lookup 시간

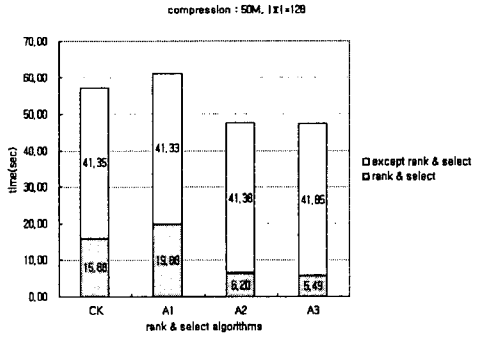

그림 6. $|\Sigma|=128$, 50Mbit 랜덤 스트링에 대한 써픽스 배열 압축 시간

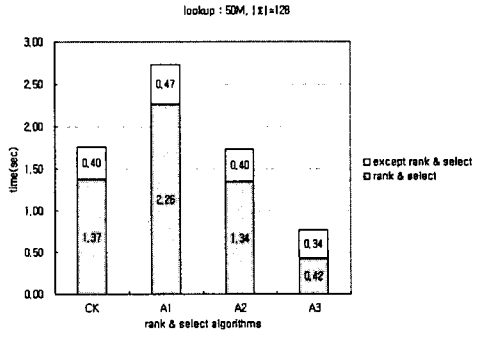

그림 7. $|\Sigma|=128$, 50Mbit 랜덤 스트링에 대한 압축된 써픽스 배열 lookup 시간

참고문헌

[1] P. Weiner, Linear pattern matching algorithms, Proceedings of the 14th IEEE Symposium on Switching and Automata Theory, pp.1-11, 1973

[2] E. M. McCreight, A Space-Economical Suffix Tree Construction Algorithm , Journal of the ACM, vol.23, pp.262-272, 1976

[3] E. Ukkonen, On-Line Construction of Suffix Trees, Algorithmica, vol.14, no.3, pp.249-260, 1995

[4] M. Farach, Optimal suffix tree construction with large alphabets, Proceedings of the 38th Annual Symposium on Foundations of Computer Science, pp.137-143, 1997

[5] 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

[6] D. Gusfield, An "Increment-by-one" approach to suffix arrays and trees, Report. CSE-90-39, Computer Science Division, University of California, Davis, 1990.

[7] J. Kärkkäinen and P. Sanders, Simple Linear Work Suffix Array Construction, Proceedings of the 13th International Conference on Automata, Languages and Programming, pp.186-199, 2003

[8] P. Ko and S. Aluru, Space Efficient Linear Time Construction of Suffix Arrays, Lecture Notes in Computer Science, vol.2676, pp.200-210, 2003

[9] D. K. Kim, J. S. Sim, H. Park and K. Park, Linear-Time Construction of Suffix Arrays, Lecture Notes in Computer Science, vol.2676, pp.186-199

[10] R. Grossi and J. S. Vitter, Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching, Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, pp.397-406, 2000

[11] G. Jacobson, Succinct Static Data Structures, PhD thesis, Carnegie Mellon University, Pittsburgh, 1988

[12] G. Jacobson, Space-efficient static trees and graphs, Proceedings of the IEEE Symposium on Foundations of Computer Science, pp.549-554, 1989

[13] D. R. Clark, Compact Pat Trees, PhD thesis, University of Waterloo, Waterloo, 1988

[14] J. I. Munro and V. Raman, Succinct representation of balanced parentheses and static trees, SIAM Journal of Computing, vol.31, no.3, pp.762-776, 2001

[15] D. K. Kim, J. C. Na, J. E. Kim and K. Park, Efficient Implementation of Rank and Select Functions for Succinct Representation, Lecture Notes in Computer Science, vol.3503, pp.315-327, 2005