접미사 배열 생성 과정에서 구간 최소값 위치를 상수 시간에 찾기 위한 효율적인 자료구조

(An Efficient Data Structure to Obtain Range Minima in Constant Time in Constructing Suffix Arrays)

박 희 진 †

(Heejin Park)

요 약 본 논문에서는 배열에서 구간 최소값 위치를 상수 시간에 찾기 위한 효율적인 자료구조를 제시한다. 최근의 생물 정보학 분야에서 빠른 DNA 서열의 검색을 위해 접미사 배열이 많이 사용되고 있는데 이 접미사 배열을 생성하는 문제는 구간 최소값 위치 문제를 포함하고 있다. 이 접미사 배열을 생성할 때는 구간 최소값 위치 문제를 빠르게 푸는 것뿐만 아니라 공간 효율적으로 해결하는 것도 중요하다. 그 이유는 DNA 서열이 수백만 개에서 수십억 개의 염기를 가진 굉장히 큰 데이타이기 때문이다. 배열의 구간 최소값 위치를 상수 시간에 찾기 위해 지금까지 알려진 가장 효율적인 자료구조는 배열의 구간 최소값 문제를 Cartesian 트리에서의 LCA(Lowest Common Ancestor) 문제로 바꾸고 이 트리에서의 LCA 문제를 다시 특수한 배열에서의 구간 최소값 문제로 바꾸어 푸는 방법을 이용한 자료구조이다. 이 자료구조는 이론적으로 $O(n)$ 공간을 사용하여 $O(n)$ 시간에 생성된다. 하지만 이 자료구조는 배열의 구간 최소값 문제를 두 번에 걸쳐 다른 문제로 변환하는 과정을 포함하고 있기 때문에 실제로 사용되는 공간은 상당히 큰 $13n$이며 또한 많은 시간이 요구된다. 본 논문에서 제시하는 자료구조는 배열의 구간 최소값 문제를 다른 문제로 변환하지 않고 직접 구하는 자료구조이다. 따라서 이론적으로 $O(n)$ 공간을 차지하며 $O(n)$ 시간에 생성될 뿐만 아니라 실제적으로도 $5n$의 적은 공간을 사용하며 빠른 시간에 생성된다.

키워드 : 구간 최소값 위치, LCA

Abstract We present an efficient data structure to obtain the range minima in an array in constant time. Recently, suffix arrays are extensively used to search DNA sequences fast in bioiformatics. In constructing suffix arrays, solving the range minima problem is necessary. When we construct suffix arrays, we should solve the range minima problem not only in a time-efficient way but also in a space-efficient way. The reason is that DNA sequences consist of millions or billions of bases. Until now, the most efficient data structure to find the range minima in an array in constant time is based on the method that converts the range minima problem in an array into the LCA (Lowest Common Ancestor) problem in a Cartesian tree and then converts the LCA problem into the range minima problem in a specific array. This data structure occupies $O(n)$ space and is constructed in $O(n)$ time. However since this data structure includes intermediate data structures required to convert the range minima problem in an array into other problems, it requires large space ($=13n$) and much time. Our data structure is based on the method that directly solves the range minima problem. Thus, our data structure requires small space ($=5n$) and less time in practice. As a matter of course, our data structure requires $O(n)$ time and space theoretically.

Key words : range minima, LCA

1. 서 론

구간 최소값 위치 문제는 다른 문제를 풀기위해 많이 사용되는 문제이다. 전통적인 계산 기하학 분야에서 다뤄온 최소 확장 트리 문제, 사각형 교차 문제, 그리고 사각형 포함 문제 등은 구간 최소값 위치 문제를 포함하고 있다. 최근의 생물 정보학 분야에서 DNA 서열의

· 이 논문은 2003년 한양대학교 교내연구비 지원으로 연구되었음

† 종신회원 : 한양대학교 정보통신대학 교수

hjpark@hanyang.ac.kr

논문접수 : 2003년 8월 1일

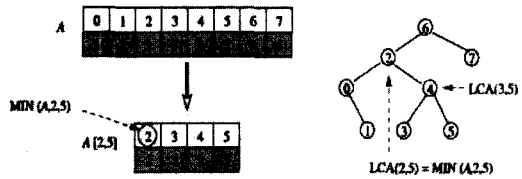
심사완료 : 2003년 11월 21일

검색을 위한 자료구조로 접미사 배열이 많이 사용되고 있는데 이 접미사 배열을 생성하는 문제[1]도 구간 최소값 위치 문제를 포함하고 있다. 접미사 배열을 생성하기 위해서 구간 최소값 위치 문제를 해결해야 할 때는 시간 효율적으로 해결하는 것뿐만 아니라 공간 효율적으로 해결하는 것도 중요하다. 그 이유는 DNA 서열이 수백만 개에서 수십억 개의 염기를 가진 굉장히 큰 데이타이기 때문이다.

이러한 문제들에 포함된 구간 최소값 위치 문제는 일반적으로 다음과 같은 특징을 가지고 있다. 그 특징은 임의의 배열 A는 처음에 주어진 후 변하지 않으며 이 배열 A의 서로 다른 여러 인덱스 구간에서 구간 최소값 위치 문제를 풀어야 한다는 것이다. 따라서 배열 A를 사전처리하여 구간 최소값 위치 문제를 빠르게 푸는 방법이 연구되어왔다.

구간 최소값 위치 문제는 0에서 $n-1$사이(0과 $n-1$ 포함)의 정수를 n개 저장하고 있는 배열에서 정의된다. 이와 같은 배열 A와 이 배열의 인덱스 구간 $[i,j]$ $(0 \le i < j \le n-1)$가 주어졌을 때 구간 최소값 위치 MIN(A,i,j)를 찾는 문제는 부분 배열 $A[i,j]$에서 최소값이 저장되어 있는 위치의 인덱스를 찾는 문제이다. 최소값이 저장되어 있는 위치가 여럿인 경우 가장 왼쪽 위치의 인덱스를 찾는다. 그림 1(a)는 8 개의 원소를 가진 배열 A에서 구간 최소값 위치 MIN(A,2,5)를 구하는 과정을 보여주고 있다. 부분 배열 $A[2,5]$에서 최소값 2가 저장되어 있는 가장 왼쪽 위치의 인덱스가 2이므로 MIN(A,2,5)값은 2가 된다.

배열의 구간 최소값 위치를 찾는 문제는 그 배열의 Cartesian 트리[2]에서 LCA(Lowest Common Ancestor)를 찾는 문제로 변환될 수 있다. 어떤 배열의 Cartesian 트리는 그 배열의 인덱스가 노드로 표현되는 트리이며 다음과 같이 재귀적으로 정의된다: (1) 루트 노드는 배열의 최소값이 저장된 위치의 인덱스이고 (2) 루트 노드의 왼쪽(오른쪽) 서브트리는 루트 노드보다 작은 (큰) 인덱스들로 이루어진 구간에 대한 부분 배열의 Cartesian 트리이다. 그림 1(b)는 그림 1(a)의 배열 A의 Cartesian 트리를 보여주고 있다. 이 트리의 루트 노드는 배열 A의 최소값 0을 저장하고 있는 위치의 인덱스 6이며 왼쪽 서브트리는 6보다 작은 인덱스들의 구간에 대한 부분 배열 $A[0,5]$의 Cartesian 트리이고 오른쪽 서브트리는 6보다 큰 인덱스들의 구간에 대한 부분 배열 $A[7]$의 Cartesian 트리이다. LCA 문제는 트리에서 두 개의 노드가 주어졌을 때 이 두 노드의 공통 조상이 되는 노드 중 루트 노드에서 가장 멀리 떨어져 있는 노드를 찾는 문제이다. 그림 1(b)의 트리에서 노드 3과 노드 5의 LCA는 노드 4가 된다. 배열의 구간 최소값 위치 문제 MIN(A,2,5)는 Cartesian 트리에서 노드 2와 노드 5의 LCA를 찾는 문제가 된다. n개의 원소를 가진 배열을 Cartesian 트리로 변환하기 위해서는 $4n$의 공간이 필요하다. 왜냐하면 Cartesian 트리의 각 노드가 4개의 공간(2개의 자식 포인터, 1개의 부모 포인터, 인덱스)을 필요로 하기 때문이다.



(a) (b)

그림 1

배열의 구간 최소값 위치를 상수 시간에 찾기 위한 자료구조 중 가장 단순한 자료구조는 $n \times n$ 이차원 정수 배열이다. 이 배열의 i번째 행, j번째 열에는 구간 최소값 위치 MIN(A,i,j)가 저장되어 있으며 따라서 이 이차원 배열을 이용하면 모든 경우의 구간 최소값 위치를 상수 시간에 구할 수 있다. 하지만 이 이차원 배열은 $O(n^2)$ 공간을 사용하고 이 배열을 생성하기 위해서는 $O(n^2)$ 시간이 걸리므로 비효율적이다. Gabow, Bently, 그리고 Tarjan[3]은 배열을 Cartesian 트리로 변환하고 Cartesian 트리에서 LCA를 구하는 자료구조[4]를 이용함으로써 배열의 구간 최소값 위치를 상수 시간에 구하는 방법을 제시하였다. 이 방법은 배열을 Cartesian 트리로 변환하는 과정에서 $O(n)$ 공간과 $O(n)$ 시간 그리고 트리에서 LCA를 구하는 자료구조를 만들기 위해 $O(n)$ 공간과 $O(n)$ 시간을 사용하므로 전체적으로 $O(n)$ 공간을 사용하며 $O(n)$ 시간에 생성되는 자료구조이다. 하지만 이 방법은 이론적으로는 $O(n)$ 공간과 $O(n)$ 시간을 요구하지만 시간·공간 복잡도 속에 숨겨진 상수가 크고 구현 방법도 복잡해서 실제적으로 사용되기에는 어려운 방법이다. 이 방법은 Schieber와 Vishkin[5]에 의해서 개선되었으며 Bender와 Farach-Colton[6]에 의해서 더욱 개선되었다. 개선된 자료구조는 시간·공간 복잡도 상으로는 같은 $O(n)$ 이지만 구현 방법이 좀 더 간단하고 시간·공간 복잡도 속에 숨겨진 상수를 개선한 자료구조이다. Bender와 Farach-Colton의 방법은 배열의 구간 최소값 문제를 Cartesian 트리에서의 LCA 문제로 변환하고 이를 다시 특수한 배열에서의 구간 최소값 문제로 변환하여 해결하는 방법이다. 이 방법의 경우 $13n$의 공간을 사용한다. 본 논문에서는

구간 최소값 문제를 Cartesian 트리의 LCA문제로 변환하지 않고 직접 해결하는 방법에 기반한 자료구조를 제시한다. 이 자료구조는 배열을 Cartesian 트리로 변환하는 과정 없이 직접 구간 최소값을 구하는 방법에 기반하고 있기 때문에 구현 방법도 단순하고 생성하는데 걸리는 시간도 짧다. 무엇보다도 이 자료구조는 $5n$의 적은 공간만을 요구하기 때문에 기존의 방법과 비교하여 더 효율적이다.

본 논문에서는 배열에서 상수 시간에 구간 최소값 위치를 구하기 위해 자료구조 L, LL, E를 사용한다. 자료구조 L, LL, E는 각각 특정 인덱스 구간$[i,j]$에서 구간 최소값 위치 MIN(A,i,j)를 상수 시간에 구하기 위한 자료구조이며 이 자료구조를 모두 사용하면 일반적인 인덱스 구간$[i,j]$에서 구간 최소값 위치를 상수 시간에 구할 수 있다. 이 자료구조 L, LL, E가 차지하는 공간은 $5n$이며 이 자료구조를 $O(n)$ 시간에 모두 생성 할 수 있다. 본 논문에서 제시하는 자료구조는 $O(n)$ 공간을 사용하며 $O(n)$ 시간에 생성되고 트리의 LCA 문제로 변환하는 과정을 거치지 않고 배열에서 직접 구간 최소값 위치를 상수 시간에 구하기 위한 최초의 자료구조이다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 구간 최소값 위치를 상수 시간에 구하기 위한 기본적인 사전처리 방법 α와 β를 소개한다. 방법 α는 배열 A는 미리 알려지지 않고 배열 A의 원소의 개수와 배열의 원소가 가질 수 있는 값의 경우의 수만 미리 알려졌을 경우 사용하는 사전처리 방법이며 방법 β는 임의의 값을 갖는 n개의 정수가 저장된 배열 A가 미리 주어졌을 경우 사용하는 사전처리 방법이다. 3장에서는 0에서 $n-1$ 사이의 값을 갖는 n개의 정수가 저장된 배열 A가 미리 주어졌을 때 사용하는 자료구조인 L, LL, E를 소개하고 이 자료구조들을 방법 α와 β를 사용하여 생성하는 과정을 설명한다. 4장에서는 이 자료구조 L, LL, E를 이용하여 구간 최소값 위치를 상수 시간에 구하는 방법을 설명한다. 5장에서는 기존의 방법과 본 논문의 방법을 비교하며 6장에서는 결론을 맺는다.

2. 기본적인 사전처리 방법

2.1 방법 α

이 방법은 배열 A는 미리 알려지지 않고 배열 A의 원소의 개수 n과 배열의 원소가 가질 수 있는 값의 경우의 수 m이 미리 알려졌을 경우 사용하는 사전처리 방법이다. 배열 A가 미리 주어지지 않으므로 모든 가능한 (A,i,j)에 대해서 구간 최소값 위치 MIN(A,i,j)을 계산해서 저장해야 어떤 (A,i,j)에 대해서도 상수 시간에 구간 최소값 위치를 구할 수 있다. 모든 가능한 배열 A의 종류는 m^n개이고 모든 가능한 인덱스 구간 $[i,j]$의 종류는 n^2개이므로 모든 가능한 (A,i,j)의 종류는 $m^n n^2$개이다. 따라서 이 방법은 $m^n n^2$ 공간 자료구조를 이용하는 방법이다. 이 $m^n n^2$ 공간 자료구조는 $O(m^n n^2)$ 시간에 생성할 수 있다. 본 논문에서는 이 방법을 $m=n$인 경우에 사용하며 이 경우 모든 가능한 (A,i,j)의 종류는 n^{n+2}개이다.

2.2 방법 β

이 방법은 임의의 값을 갖는 n개의 정수가 저장된 배열 A가 미리 주어졌을 경우 사용하는 방법이다. 배열 A가 미리 주어지므로 배열 A를 사전처리한 자료구조를 만들고 이 자료구조를 이용하여 주어진 인덱스 구간 $[i,j]$에 대해서 구간 최소값 위치 MIN(A,i,j)를 상수 시간에 구한다. 이 방법에서 사용하는 자료구조는 이차원 배열 P와 S이며 이 배열들은 $n\lfloor\lfloor\log n\rfloor\rfloor$개의 공간을 사용하며 $O(n\log n)$ 시간에 생성된다. 먼저 배열 P와 S에 대해서 설명한 후 이 배열을 이용하여 상수 시간에 구간 최소값 위치를 구하는 방법을 설명한다.

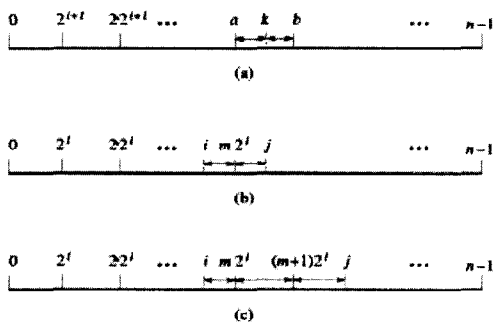


그림 2

2.2.1 이차원 배열 P와 S

먼저 임의의 인덱스 구간 $[a,b]$에 대해서 접두 인덱스 구간과 접미 인덱스 구간을 정의한 후 이차원 배열 P와 S 에 대해서 설명한다. 인덱스 구간 $[a,b]$의 접두 인덱스 구간들은 a에서 시작하는 $[a,b]$의 부분 구간들이며 $[a,x](a\le x\le b)$의 형태로 표현된다. 인덱스 구간 $[a,b]$의 접미 인덱스 구간들은 b에서 끝나는 $[a,b]$의 부분 구간들이며 $[y,b](a\le y\le b)$의 형태로 표현된다. 인덱스 구간 $[a,b]$의 모든 접두 인덱스 구간의 개수는 인덱스 구간 $[a,b]$의 길이인 $b-a+1$개이며 모든 접미 인덱스 구간의 개수도 $b-a+1$개이다.

이제 이차원 배열 P와 S에 대해서 설명한다. 배열 P

는 $\lfloor |\log n| \rfloor$ 개의 일차원 배열 P_0, P_1, ..., $P_{\lfloor \log n \rfloor -1}$로 구성되며 배열 S도 $\lfloor |\log n| \rfloor$ 개의 일차원 배열 S_0, S_1, ..., $S_{\lfloor \log n \rfloor -1}$로 구성되어 있다. 일차원 배열 P_i와 S_i $(0 \le i \le \lfloor \log n \rfloor -1)$에 저장되어 있는 값은 다음과 같다. 먼저 전체 인덱스 구간 $[0, n-1]$을 길이가 2^{i+1}인 구간으로 분할한다(그림 2(a)). 분할된 구간 하나를 $[a, b]$라 하자. 분할된 구간 $[a, b]$의 원소 k $(a \le k \le b)$마다 k에서 끝나는 접두 인덱스 구간 $[a, k]$와 k에서 시작하는 접미 인덱스 구간 $[k, b]$가 존재한다. 접두 인덱스 구간 $[a, k]$의 최소값 위치 MIN(A,a,k)는 배열 P_i의 원소 $P_i[k]$에 저장되며 접미 인덱스 구간 $[k, b]$의 최소값 위치 MIN(A,k,b)는 배열 S_i의 원소 $S_i[k]$에 저장된다. 예를 들어, 8개의 원소를 가진 배열 A = [1,3,2,7,2,3,6,0]가 주어졌을 때 $P_1[2]$와 $S_1[2]$에 저장되는 값을 계산해 보자. 먼저 인덱스 구간 [0,7]을 길이가 2^{1+1}인 구간으로 분할하면 인덱스 구간 [0,3]과 [4,7]이 생성된다. $P_1[2]$에는 인덱스 구간 [0,3]의 접두 인덱스 구간 [0,2]에 대한 구간 최소값 위치 MIN(A,0,2)= 0이 저장되고 $S_1[2]$에는 접미 인덱스 구간 [2,3]에 대한 구간 최소값 위치 MIN(A,2,3)=2가 저장된다.

이차원 배열 P와 S는 $|\log n| \times n$ 배열이므로 $n|\log n|$개의 공간을 사용하는 것은 자명하며 배열 P와 S를 $O(n \log n)$ 시간에 생성하는 방법은 다음과 같다. 배열 P의 경우 배열 P_0, P_1, ..., $P_{\lfloor \log n \rfloor -1}$의 순서로 생성한다. 배열 S도 마찬가지이다. 배열 P_{i+1}의 원소 $P_{i+1}[k]$는 배열 P_i를 이용하여 상수 시간에 생성할 수 있으며 생성하는 방법은 다음과 같다. 원소 $P_{i+1}[k]$는 $\lfloor k/2^{i+1} \rfloor$이 짝수이냐 홀수이냐에 따라서 구하는 방법이 달라진다. $\lfloor k/2^{i+1} \rfloor$이 짝수이면 $P_{i+1}[k]$의 값이 $P_i[k]$의 값과 동일하다. $\lfloor k/2^{i+1} \rfloor$이 홀수이면 $P_i[(2j+1)2^{i+1}-1]$에 저장된 인덱스와 $P_i[k]$에 저장된 인덱스 중 더 작은 값을 저장하고 있는 위치의 인덱스가 $P_{i+1}[k]$가 된다. 비슷한 방법으로 배열 S의 원소 $S_{i+1}[k]$도 상수 시간에 구할 수 있다. 이 방법을 사용하면 배열 $P_{i+1}(S_{i+1})$을 $O(n)$ 시간에 생성할 수 있으며 따라서 전체 배열 P와 S를 $O(n \log n)$ 시간에 모두 생성할 수 있다.

2.2.2 상수시간에 구간 최소값을 구하는 방법

배열 P와 S를 이용하여 인덱스 구간 $[i, j]$의 최소값 위치 MIN(A,i,j)를 상수 시간에 구하는 방법은 다음과 같다. 먼저 다음의 lemma를 소개한 후 구간 최소값 위치 MIN(A,i,j)을 상수 시간에 구하는 방법을 설명한다. 설명을 쉽게 하기 위해서 $\lfloor \log(j-i) \rfloor$을 l로 $\lfloor j/2^l \rfloor - \lfloor i/2^l \rfloor$을 l'으로 정의한다.

Lemma 1. 인덱스 구간 $[i, j]$는 2 개의 인접한 2^l 길이 구간에(그림 2(b)) 걸쳐 있거나 3 개의 인접한 2^l 길이 구간에(그림 2(c)) 걸쳐 있다.

Proof > $l = \lfloor \log(j-i) \rfloor$이므로 다음의 부등식 $2^l + 1 \le j - i + 1 < 2^{l+1} + 1$이 성립한다. 부등식 $2^l + 1 \le j - i + 1$에 의해서 인덱스 구간 $[i, j]$는 하나의 2^l 길이 구간보다 길다. 따라서 인덱스 구간 $[i, j]$는 두 개 이상의 인접한 2^l 길이 구간에 걸쳐 있게 된다. 또 부등식 $j - i + 1 < 2^{l+1} + 1$에 의해서 인덱스 구간 $[i, j]$는 두개의 인접한 2^l 길이의 구간의 합보다 길지 않다. 따라서 4 개 이상의 인접한 2^l 길이 구간에 걸쳐 있을 수 없고 3 개 이하의 인접한 2^l 길이 구간에 걸쳐 있게 된다.

Lemma 2. 인덱스 구간 $[i, j]$이 2 개의 인접한 2^l 길이 구간에 걸쳐 있는 경우는 $l' = \lfloor j/2^l \rfloor - \lfloor i/2^l \rfloor$이 1인 경우이고 3개의 인접한 2^l 길이 구간에 걸쳐 있는 경우에는 l'이 2인 경우이다.

Proof > 인덱스 구간 $[i, j]$이 2 개의 인접한 2^l 길이 구간에 걸쳐 있는 경우는 어떤 m에 대해서 $\lfloor i/2^l \rfloor$이 m이고 $\lfloor j/2^l \rfloor$이 $m+1$인 경우이다. 따라서 $l' = m+1-m = 1$이 성립한다. 3 개의 인접한 2^l 길이 구간에 걸쳐 있는 경우에는 $\lfloor i/2^l \rfloor$이 m이고 $\lfloor j/2^l \rfloor$이 $m+2$가 되므로 $l' = m+2-m = 2$가 성립한다.

인덱스 구간 $[i, j]$가 그림 2(b)처럼 2개의 인접한 2^l 길이 구간에 걸쳐 있는 경우에 MIN(A,i,j) 값은 인덱스 MIN(A,i,m2^l-1) $(= S_{l-1}[i])$과 인덱스 MIN(A,m2^l,j) $(= P_{l-1}[j])$ 중 더 작은 값을 저장하고 있는 위치의 인덱스가 MIN(A,i,j) 값이 된다.

인덱스 구간 $[i, j]$가 그림 2(c)처럼 3개의 인접한 2^l 길이 구간에 걸쳐 있는 경우에는 $m2^l$와 $(m+1)2^l$ 중 하나만이 2^{l+1}의 배수가 되며 인덱스 구간 $[i, j]$가 2 개의 인접한 2^{l+1} 길이 구간에 걸쳐 있게 된다. 따라서 이 경우 인덱스 $S_l[i]$과 인덱스 $P_l[j]$ 중 더 작은 값을 저장하고 있는 위치의 인덱스가 MIN(A,i,j) 값이 된다.

3. 자료구조 L, LL, E

이 자료구조들은 0에서 $n-1$사이의 값 (0과 $n-1$포함)을 갖는 n개의 정수로 이루어진 배열 A를 사전처리하여 인덱스 구간 $[i, j]$가 주어졌을 때 구간 최소값 위치 MIN(A,i,j)를 상수 시간에 구하기 위한 자료구조

이다. 각각의 자료구조 하나만을 이용하면 어떤 특정한 인덱스 구간 $[i,j]$의 최소값 위치를 상수 시간에 구할 수 있고 자료구조 L, LL, E를 모두 사용하면 임의의 인덱스 구간 $[i,j]$의 최소값 위치를 상수 시간에 구할 수 있다. 본 장에서는 각각의 자료구조 L, LL, E에 대해 구간 최소값 위치를 상수 시간에 구할 수 있는 인덱스 구간을 소개하고 그 자료구조를 생성하는 방법을 설명한다. 자료구조 L, LL, E를 모두 사용하여 임의의 인덱스 구간 $[i,j]$의 최소값 위치를 상수 시간에 구하는 방법은 다음 장에서 설명한다. 설명을 쉽게 하기위해서 n이 $\log n$으로 나누어 떨어지고 $\log n$이 $\log\log n$으로 나누어 떨어진다고 가정한다. 이렇게 가정해도 일반성을 잃지 않는다.

자료구조 L은 i와 $j+1$이 $\log n$의 배수가 되는 인덱스 구간 $[i,j]$에서 최소값 위치를 상수 시간에 구하기 위한 자료구조이다. 한 예로 자료구조 L을 이용하면 인덱스 구간 $[2\log n, 5\log n-1]$에서 최소값 위치를 상수 시간에 구할 수 있다. 자료구조 L을 생성하는 방법은 다음과 같다. 먼저 전체 인덱스 구간 $[0,n-1]$을 그림 3 (a)와 같이 길이가 $\log n$인 인덱스 구간 $[0,\log n-1]$, $[\log n, 2\log n-1]$, ..., $[n-\log n, n-1]$으로 나누고 나누어진 각각의 인덱스 구간에서 최소값 위치를 찾는다. 이렇게 구한 $n/\log n$ 개의 구간 최소값 위치에 대해서 β 방법을 적용하여 자료구조 L을 만든다. 자료구조 L은 β 방법을 사용하여 생성하기 때문에 P와 S 배열을 생성하기 위해 $2\cdot n/\log n\cdot\log(n/\log n)$ $<2n$ 공간이 필요하며 $O(n)$ 시간이 필요하다. 이 자료구조 L을 이용하면 i와 $j+1$이 $\log n$의 배수가 되는 인덱스 구간 $[i,j]$에서 최소값 위치를 상수 시간에 구할 수 있다.

자료구조 LL은 길이가 $\log n$인 분할된 인덱스 구간

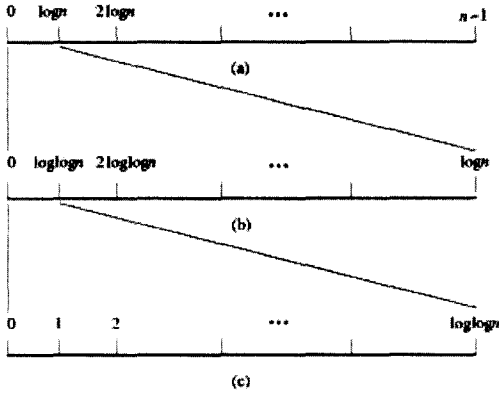


그림 3

에 포함되는 인덱스 구간 $[i,j]$ 중에서 i와 $j+1$이 $\log\log n$의 배수가 되는 인덱스 구간 $[i,j]$의 최소값 위치를 상수 시간에 구하기 위한 자료구조이다. 자료구조 LL을 생성하는 방법은 다음과 같다. 먼저 전체 인덱스 구간 $[0,n-1]$을 길이가 $\log n$인 인덱스 구간으로 나눈다. 길이가 $\log n$인 인덱스 구간마다 그림 3 (b)처럼 다시 길이가 $\log\log n$인 인덱스 구간으로 나누고 나누어진 각각의 인덱스 구간에서 최소값 위치를 찾는다. 이렇게 구한 $\log n/\log\log n$ 개의 구간 최소값 위치에 대해서 β 방법을 적용하여 $2\cdot\log n/\log\log n\cdot\log(\log n/\log\log n)<2\log n$ 공간을 사용하는 자료구조를 만든다. 길이가 $\log n$인 인덱스 구간은 모두 $n/\log n$ 개가 있으므로 길이가 $\log n$인 인덱스 구간마다 생성되는 자료구조를 모두 합하면 $2\cdot\log n\cdot n/\log n=2n$ 공간이 된다. 이 자료구조 LL을 이용하면 길이가 $\log n$인 분할된 인덱스 구간에 포함되는 인덱스 구간 $[i,j]$ 중에서 i와 $j+1$이 $\log\log n$의 배수가 되는 인덱스 구간 $[i,j]$의 최소값 위치를 상수 시간에 구할 수 있다.

자료구조 E는 길이가 $\log\log n$인 분할된 인덱스 구간에 포함되는 인덱스 구간 $[i,j]$의 최소값 위치를 상수 시간에 구하기 위한 자료구조이다. 자료구조 E를 생성하는 방법은 다음과 같다. 먼저 0에서 $\log\log n-1$사이(0과 $\log\log n-1$포함)의 값을 갖는 $\log\log n$개의 정수로 구성된 모든 배열의 모든 구간에 대해서 방법 α를 적용하여 구간 최소값 위치를 구하여 저장한다. $\log\log n^{\log\log n+2}<n$이므로 이 자료구조는 n 공간을 사용하며 $O(n)$ 시간에 생성된다. n이 16인 경우의 예를 들어보자. 이 경우 $\log\log n$이 2이고 따라서 0과 1로 구성된 길이가 2인 모든 배열 $[0,0]$, $[0,1]$, $[1,0]$, $[1,1]$을 생성하고 이 배열들의 모든 구간에 대해서 구간 최소값 위치를 구하여 저장한다. 그 다음 전체 인덱스 구간 $[1,n-1]$을 그림 3(b)와 같이 길이가 $\log\log n$인 인덱스 구간으로 분할한다. 각각의 분할된 $\log\log n$ 길이의 인덱스 구간에 해당하는 부분 배열 $A[i,j]$를 앞에서 생성한 배열 중 하나에 대응시킨다. 대응되는 배열을 B라 하자. 배열 B는 다음의 조건을 만족시켜야한다. 그 조건은 배열 B의 원소들 간의 크기 순서가 부분배열 $A[i,j]$의 원소들 간의 크기 순서가 같아야 한다는 조건이다. 예를 들어 부분 배열이 $[100,101]$이면 배열 B는 $[0,1]$이 된다. 또 부분 배열이 $[7,9]$일 경우에도 배열 B는 $[0,1]$이 된다. 이 조건을 만족하면 부분 배열 $A[i,j]$에서의 구간 최소값 위치 MIN(A,i+p,i+q) $(0\le p<q<\log\log n)$는 배열 B에서의 구간 최소값 위치 MIN(B,p,q)와 일치하게 된다. 배열 B에서의 구간

최소값 위치 MIN(B,p,q)는 방법 a를 사용하여 만든 자료구조를 이용하면 상수 시간에 구할 수 있으므로 MIN(A,i+p,i+q)도 상수 시간에 구할 수 있다. 이제 길이가 $\log\log n$인 부분 배열 $A[i,j]$에 대응되는 배열 B를 찾는 방법을 설명한다. 부분 배열 $A[i,j]$에 대응되는 배열 B를 찾기 위해서는 $A[i,j]$를 정렬시키면 된다. 길이가 $\log\log n$인 부분 배열 $A[i,j]$을 정렬할 때 $n/\log\log n$개의 모든 부분 배열을 한꺼번에 n개의 버킷을 사용하여 $O(n)$ 시간에 정렬할 수 있다[7].

4. 일반적인 인덱스 구간 $[i,j]$의 최소값 위치를 상수 시간에 구하는 방법

자료구조 L, LL, E를 사용하여 일반적인 인덱스 구간 $[i,j]$의 최소값 위치 MIN(A,i,j)을 상수 시간에 구하는 방법은 다음과 같다. 먼저, 인덱스 구간 $[i,j]$를 최대 5개의 특정한 인덱스 구간으로 분할한다(그림 4). 분할하는 방법은 다음 lemma에서 자세하게 설명한다. 분할된 각각의 특정한 인덱스 구간에서 L, LL, E를 사용하여 최소값 위치를 상수 시간에 찾는다. 찾은 최소값 위치 중에서 전체 인덱스 구간 $[i,j]$의 최소값 위치를 찾는다. 이 때 분할된 구간의 개수가 최대 5개이므로 분할된 구간의 최소값 위치에서 전체 구간의 최소값 위치를 찾는 것은 상수 시간에 가능하다.

Lemma 3. 임의의 인덱스 구간 $[i,j]$는 자료구조 L, LL, E를 사용하여 구간 최소값을 구할 수 있는 최대 5개의 인덱스 구간으로 분할할 수 있다.

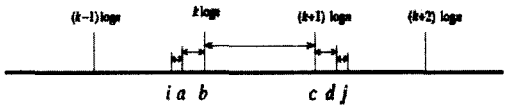


그림 4

Proof > 임의의 인덱스 구간 $[i,j]$는 다음과 같이 최대 5개의 특정한 인덱스 구간으로 분할될 수 있다(그림 4). 먼저 인덱스 구간 $[i,j]$에서 $\log n$으로 나누어지는 인덱스 중 가장 작은 인덱스 b와 가장 큰 인덱스 c를 찾는다. 또 인덱스 구간 $[i,j]$에서 $\log\log n$으로 나누어지는 인덱스 중 가장 작은 인덱스 a와 가장 큰 인덱스 d를 찾는다. 부등식 $i<a<b<c<d<j$ 가 성립될 경우 인덱스 구간 $[i,j]$는 다음과 같은 5개의 인덱스 구간 $[i,a-1]$, $[a,b-1]$, $[b,c-1]$, $[c,d-1]$, $[d,j]$로 나누어진다. 이 구간들 중 구간 $[i,a-1]$과 $[d,j]$는 각각 길이가 $\log\log n$인 분할된 인덱스 구간에 포함되므로 E를 사용하여 최소값 위치를 찾을 수 있는 특정한 인덱스 구간이다. 구간 $[a,b-1]$과 $[c,d-1]$은 각각 길이가 $\log n$인 분할된 인덱스 구간에 포함되며 a,b,c,d가 모두 $\log\log n$의 배수이므로 LL을 사용하여 최소값 위치를 상수 시간에 찾을 수 있는 특정한 인덱스 구간이다. 마지막으로 구간 $[b,c-1]$은 a,c가 $\log n$의 배수이므로 L을 사용하여 최소값 위치를 상수 시간에 찾을 수 있는 특정한 인덱스 구간이다.

부등식 $i<a<b<c<d<j$ 가 성립되지 않는 경우(예를 들어 $i=a$ 혹은 $b=c$ 혹은 b나 c가 존재하지 않은 경우) 에는 5개 이하의 구간으로 분할되며 이 경우에도 각각의 경우마다 부등식 $i<a<b<c<d<j$가 성립되는 경우와 비슷하게 5개 이하의 구간으로 분할되는 것을 보일 수 있다.

5. 기존 방법과의 비교

Bender와 Farach-Colton의 방법은 배열의 구간 최소값 문제를 Cartesian 트리에서의 LCA 문제로 변환하고 이를 다시 특수한 배열에서의 구간 최소값 문제로 변환하여 해결하는 방법이다. 이 방법의 경우 $13n$의 공간을 사용하는데 $13n$의 공간 중 $4n$ 공간은 배열을 Cartesian 트리로 변환하기 위해 필요하고 $5n$의 공간은 이 트리를 다시 특수한 배열로 변환하기 위해 필요하고 나머지 $4n$ 공간은 특수한 배열에서 구간 최소값을 구하기 위해서 필요하다. 본 논문에서는 구간 최소값 문제를 Cartesian 트리의 LCA문제로 변환하지 않고 자료구조 L, LL, E를 사용하여 직접 해결한다. 자료구조 L은 각각 이차원 배열 P와 S로 구성되어 있으며 최대 $2n$의 공간을 차지하며 자료구조 LL도 비슷하게 최대 $2n$의 공간을 차지한다. 자료구조 E는 최대 n의 공간을 차지한다. 따라서 전체 자료구조는 총 $5n$의 공간을 차지한다.

시간적인 측면에서 기존의 방법과 본 논문의 방법을 비교하기 위하여 두 가지 방법을 모두 구현하여 수행시간을 측정하였다. 구현 언어는 C언어를 사용하였고 실험 환경은 Linux가 설치되고 Pentium IV 2.0Ghz CPU가 장착된 PC이다. 먼저 원소의 개수가 1,000개인 배열 1,000개를 무작위로 생성하여 자료구조를 생성하고 각각 자료구조를 생성하는 데 걸리는 시간을 측정한 후 평균을 구하였다. 그리고 원소의 개수가 10,000 개, 100,000 개, 그리고 1,000,000 개일 때의 경우에 대해서도 배열 1,000 개를 무작위로 생성하여 같은 실험을 반복하였다. 표 1은 실험의 결과로 얻어진 측정된 시간의 평균을 보여주고 있다. 실험 결과를 정리하면 본 논문의 방법은 원소의 개수가 비교적 적은 경우(1,000 개 그리고 10,000 개)에는 기존의 방법보다 25-30% 정도의 빠르

게 수행되며 원소의 개수가 많은 경우(100,000 개 그리고 1,000,000 개)에는 약 50% 정도 빠르게 수행된다.

표 1

원소의 개수	1,000	10,000	100,000	1,000,000
Bender와 Farach의 방법	0.67ms	6.31ms	91.1ms	953ms
본 논문의 방법	0.44ms	4.62ms	45.9ms	478ms

6. 결 론

본 논문에서는 배열의 구간 최소값 위치를 Cartesian 트리의 LCA 문제로 변환하지 않고 상수 시간에 구할 수 있는 효율적인 자료구조를 제시하였다. 이 자료구조는 배열에 들어있는 원소의 개수를 n이라 할 때 $5n$ 공간을 차지하는데 이는 기존의 가장 효율적인 자료구조가 필요로 하는 $13n$보다 절반이하의 적은 공간을 차지하는 것이다. 또한 본 논문의 자료구조를 생성하는 데 걸리는 시간도 기존의 방법에 비해서 배열의 원소의 개수가 적은 경우에는 약 25%-30% 단축되고 배열의 원소가 많은 경우에는 약 50% 단축된다.

참 고 문 헌

[1] D. Kim, J. Sim, H. Park, and K. Park, Linear-Time Construction of Suffix arrays, Combinatorial Pattern Matching (2003), 186-199.

[2] Vuillemin, A Unifying Look at Ddata Structures, Comm. ACM Vol. 24, (1980), 229-239.

[3] H. Gabow, J. Bently, and R. Tarjan, Scaling and Related Techniques for Geometry Problems, ACM Symp. Theory of Computing (1984), 135-143.

[4] D. Harel and R Tarjan, Fast Algorithms for Finding Nearest Common Ancestors, SIAM J. Computing 13 (1984), 338-355.

[5] B. Schieber and U. Vishkim, On Finding Lowest Common Ancestors: Simplification and Parallelization, SIAM J. Computing 17 (1988), 1253-1262.

[6] M. Bender and M. Farach-Colton, The LCA Problem Revisited, In Proceedings of LATIN 2000, LNCS 1776, (2000), 88-94.

[7] A.Aho, J.Hopcroft, J.Ullman, Data structures and algorithms, Addisson-Wesley, 1983.

[8] O. Berkman and U. Vishkin, Recursive Star-Tree Parallel Ddata Structure, SIAM J. Computing 22 (1993), 221-242.

[9] B. Wang, G. Chen, Cost-Optimal Parallel Algorithms for Constructing 2-3 Trees, J. Parallel and Distributed Computing, 11(3), (1991), 257-262.

박 희 진

1990년 3월~1994년 2월 서울대학교 컴퓨터공학과 학사. 1994년 3월~1996년 2월 서울대학교 컴퓨터공학과 석사. 1996년 3월~2001년 2월 서울대학교 컴퓨터공학과 박사. 2001년 3월~2003년 2월 서울대학교 컴퓨터연구소 전문연구원. 2003년 2월~2003년 8월 이화여자대학교 컴퓨터학과 BK연구교수. 2003년 9월~현재 한양대학교 정보통신대학 전임강사