극대 증가 부분서열을 찾는 선형 알고리즘

(Linear-time algorithms for computing a maximal increasing subsequence)

나중채*

(Joong Chae Na)

요 약

최장 증가 부분서열(longest increasing subsequence)은 컴퓨터 과학 분야에서 오랫동안 연구되어온 주요 문제이다. 본 논문에서는 최장 조건을 극대로 완화한 극대 증가 부분서열(maximal increasing subsequence) 문제를 고려한다. 본 논문에서는 두 가지 버전의 증가 개념(단조증가, 순증가)에 대해, 알파벳 Σ에 대한 서열의 극대 증가 부분서열을 구하는 선형시간 알고리즘을 제안한다. 극대 단조증가 부분서열을 구하는 알고리즘은 $O(1)$ 공간을 사용하고, 극대 순증가 부분서열을 구하는 알고리즘은 $O(|\Sigma|)$ 공간을 사용한다.

■ 중심어 : 극대 증가 부분서열 ; 증가 부분서열 ; 서열 분석 ; 문자열 알고리즘

Abstract

The longest increasing subsequence is a fundamental problem which has been studied for a long time in computer science. In this paper, we consider the maximal increasing subsequence problem where the constraint is released from the longest to the maximal. For two kinds of increasing (monotone increasing and strictly increasing), we propose linear-time algorithms computing a maximal increasing subsequence of an input sequence from an alphabet Σ. Our algorithm for computing a maximal monotone increasing subsequence requires $O(1)$ space and our algorithm for computing a maximal strictly increasing subsequence requires $O(|\Sigma|)$ space.

■ keywords : maximal increasing subsequences ; increasing subsequences ; sequence analysis ; string algorithms

Ⅰ. 서 론

컴퓨터 과학에서 최장 증가 부분서열(longest increasing subsequence, LIS)을 찾는 문제는 보안, 네트워크 등 다양한 분야에서 활용되는 근원적인 문제로 매우 오랫동안 연구됐다[1,2]. 길이 n인 서열의 LIS를 $O(n\log n)$ 시간에 찾는 일반적인 알고리즘이 널리 알려져 있고[3], 특수한 경우에 대해 LIS를 더 빠른 시간에 구하는 알고리즘도 다양하게 개발되었다[4,5].

LIS 문제는 두 서열의 최장 공통 부분서열(longest common subsequence, LCS)을 찾는 문제와 밀접한 관련이 있다[6,7]. 일례로 서열 S의 LIS는 S의 문자들을 정렬한 서열 S'과의 LCS를 구하여 얻을 수 있다. LCS는 k-오차(mismatch), 편집거리(edit distance)와 더불어 서열의 유사도를 측정하는 기준으로 오랫동안 연구되어온 주제로[7,8,9], 길이가 m과 n인 두 문자열의 LCS는 동적프로그래밍을 이용하여 $O(mn)$ 시간에 구할 수 있다[10]. LIS와 LCS의 개념을 결합한 최장 공통 증가 부분서열(longest common increasing subsequence)을 구하는 문제도 연구가 진행됐다[11,12].

최근 LCS의 최장 조건을 극대로 완화한 극대

* 정회원, 세종대학교 컴퓨터공학과
이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (2020R1F1A1068873).
이 논문은 2021년도 교내연구비 지원에 의한 논문임.
접수일자 : 2023년 06월 26일 게재확정일 : 2023년 07월 18일
교신저자 : 나중채 e-mail : jcna@sejong.ac.kr

공통 부분서열(maximal common subsequence)에 대한 연구가 활발히 진행되고 있다. 서열에서 극대(maximal) 개념은 서열을 더 이상 확장할 수 없음을 의미한다. 길이의 합이 n인 두 문자열의 극대 공통 부분서열을 $O(n\sqrt{\log n/\log\log n}$ 시간과 $O(n)$ 공간을 사용하여 찾는 이론적인 알고리즘이 제시되었고[13], 이를 개선하여 이론적 시간과 공간 효율성을 보장하면서도 길이가 긴 극대 공통 부분서열을 찾는 실용적인 알고리즘에 관한 연구도 활발히 진행되고 있다[14,15].

본 논문에서는 극대 개념을 증가서열 문제에 적용한 극대 증가 부분서열(maximal increasing subsequence, MIS)에 대해 다룬다. 길이 n인 서열 S가 주어졌을 때, S의 MIS W는 W를 포함하는 다른 증가 부분서열이 존재하지 않는 서열을 의미한다. 가장 짧은 MIS는 $O(n\log n)$ 시간과 $O(n)$ 공간을 사용하여 구할 수 있다[16].

본 논문에서는 두 가지 버전의 증가(단조증가, 순증가) 개념을 고려하여, 문자가 중복하여 나타날 수 있는 길이가 n인 서열 S의 MIS를 구하는 선형시간 알고리즘을 제안한다. 구체적인 기여는 다음과 같다.

- 극대 단조증가 부분서열을 구하는 알고리즘을 제안한다. 제안하는 알고리즘은 $O(n)$ 시간과 $O(1)$ 공간을 사용한다.
- 극대 순증가 부분서열을 구하는 알고리즘을 제안한다. 제안하는 알고리즘은 $O(n)$ 시간과 $O(|\Sigma|)$ 공간을 사용하거나($|\Sigma|$는 알파벳의 크기), $O(n\log n)$ 시간과 $O(1)$ 공간을 사용한다.

Ⅱ. 본 론

1. 관련 연구

가. 기본 영어 정의

서열은 알파벳 집합 Σ에서 추출된 문자들의 나열이다. 서열 S의 길이(서열을 구성하는 문자의 개수)는 $|S|$로 나타내며, i번째 문자는 $S[i]$로, i번째부터 j번째까지의 연속적인 문자들의 나열 $\langle S[i],S[i+1],...,S[j]\rangle$는 $S[i,j]$로 표기한다 ($1 \le i \le j \le |S|$). $S[1,i]$는 S의 접두사, $S[i,|S|]$는 접미사라 부른다($1 \le i \le |S|$). 부분서열(subsequence)은 원본 서열에서 임의 위치의 문자들을 0개 이상 삭제하여 얻은 서열이다. 즉, $1 \le l \le |S|$이고, $1 \le i_1 < i_2 < \cdots < i_l \le |S|$를 만족하는 $\langle S[i_1],S[i_2],...,S[i_l]\rangle$은 S의 부분서열이다. 어떤 서열 W가 S의 부분서열이면, S는 W를 포함한다고 말한다. 빈(empty) 문자는 Δ로 표시하고, Σ의 어떤 문자보다 작다고 가정한다. 기술을 간단히 하기 위해 $S[0]$은 빈 문자 Δ라고 가정한다. 두 서열 S_1과 S_2를 이어 붙인 서열은 $S_1 \circ S_2$로 표기한다.

본 논문에서는 서열의 증가 개념을 다음 두 가지 버전으로 나누어서 고려한다.

정의 1. 서열 $S=\langle S[1],S[2],...,S[n]\rangle$의 문자가 $S[1] \le S[2] \le \cdots \le S[n]$을 만족하면, S는 단조증가(monotone increasing)한다고 정의한다.

정의 2. 서열 $S=\langle S[1],S[2],...,S[n]\rangle$의 문자가 $S[1] < S[2] < \cdots < S[n]$을 만족하면, S는 순증가(strictly increasing)한다고 정의한다.

예를 들어, $\langle 2,2,5,7,9\rangle$는 단조증가 서열이지만, 순증가 서열은 아니다. $\langle 2,4,5,7,9\rangle$는 단조증가 서열이자 순증가 서열이다. 정의상 서열 S가 순증가 서열이면 S는 단조증가 서열이기도 하다. 하지만 역은 성립하지 않는다.

나. 극대 증가 부분서열

서열에서 극대(maximal) 개념은 서열을 더 이상 확장할 수 없음을 의미한다. 극대 증가 부분서열(maximal increasing subsequence, MIS)은 증가 개념의 정의에 따라 다음과 같이 두 가지 버전으로 정의한다.

정의 3. S의 단조증가 부분서열 W에 대해, W를 포함하는 S의 다른 단조증가 부분서열이 존재하지 않으면, W는 S의 극대 단조증가 부분서열(maximal monotone increasing subsequence, MMIS)이다.

정의 4. S의 순증가 부분서열 W에 대해, W를 포함하는 S의 다른 순증가 부분서열이 존재하지 않으면, W는 S의 극대 순증가 부분서열(maximal strictly increasing subsequence, MSIS)이다.

예를 들어, $S=\langle 2,5,7,4,7,5,4,7,5\rangle$가 주어졌을 때, $W_1=\langle 2,5,7,7\rangle$을 부분서열로 포함하는 단조증가 부분서열 $\langle 2,5,7,7,7\rangle$이 존재하므로, W_1은 MMIS가 아니다. 반면 $W_2=\langle 2,5,5,7\rangle$를 포함하는 다른 증가 부분서열이 존재하지 않으므로 W_2는 MMIS이다. 비슷하게 $W_3=\langle 2,4,7\rangle$은 $W_4=\langle 2,4,5,7\rangle$가 존재하므로 MSIS가 아니지만, W_4는 MSIS이다.

본 논문에서는 다음 문제를 해결한다.

문제 1. 서열 S가 주어졌을 때, S의 임의의 MMIS를 하나 구하라.

문제 2. 서열 S가 주어졌을 때, S의 임의의 MSIS를 하나 구하라.

서열의 모든 문자가 서로 다른(distinct) 경우에는 탐욕(greedy) 알고리즘을 이용하여 극대 증가 부분서열(MMIS이자 MSIS) W를 선형시간에 간단히 구할 수 있다. 원본 서열 S의 첫 문자를 W의 첫 문자로 설정한 후, S의 문자를 차례로 스캔하면서 현재 W의 마지막 문자보다 (같거나) 큰 문자가 나타나면 해당 문자를 W의 끝에 추가(append)하는 과정을 반복한다. 예를 들어, $S=\langle 3,2,5,9,7,4\rangle$가 주어졌을 때, 위 알고리즘에 의해 구해지는 $W=\langle 3,5,9\rangle$는 S의 극대 증가 부분서열이다. 문자가 한 번씩만 나타나기 때문에 위 알고리즘에 의해 구해지는 W가 S의 극대 증가 부분서열임은 쉽게 증명할 수 있다.

서열에 동일한 문자가 존재하는 경우를 고려해 보자. 위 탐욕 알고리즘에서 S의 문자와 W의 마지막 문자 비교 시 같은 경우를 허용하면 MMIS를 구할 수 있다. $S=\langle 2,5,7,4,7,5,4,7,5\rangle$가 주어졌을 때, 알고리즘은 $W=\langle 2,5,7,7,7\rangle$를 구한다. 이는 S에서 가장 큰 문자를 모두 포함하는 MMIS이다. 하지만, 위 탐욕 알고리즘은 MSIS는 제대로 구하지 못한다. 예를 들어, $S=\langle 2,5,7,4,7,5,4,7,5\rangle$가 주어졌을 때, 위 탐욕 알고리즘은 앞 세 문자로 구성된 증가 부분수열 $W=\langle 2,5,7\rangle$을 찾는다. 하지만 W는 S의 MSIS가 아니다. 이는 S의 후반부에 나타나는 문자 5와 7때문에, 서열 중간에 있는 4를 포함하는 증가 부분서열 $\langle 2,4,5,7\rangle$이 존재하기 때문이다. 따라서 MSIS를 구하기 위해서는 이러한 특징을 고려해야 한다.

Algorithm MMIS($S[1,n]$)
1: $W \leftarrow S[0]$ (Δ), $m \leftarrow 0$
2: **for** $i \leftarrow 1$ to n
3: **while** $W[m] > S[i]$
4: delete $W[m]$ from W, $m \leftarrow m-1$
5: $W \leftarrow W \circ S[i]$, $m \leftarrow m+1$
6: **return** W

그림 1. MMIS를 구하는 알고리즘

2. 극대 증가 부분서열 알고리즘

이 절에서는 MMIS와 MSIS를 선형시간에 구하는 알고리즘을 제시하고, 알고리즘의 정확성과 시간 및 공간복잡도를 분석한다. 앞 절의 탐욕 알고리즘을 이용하여 MMIS를 구할 수 있지만, 이 절에서는 MSIS를 구하는 알고리즘과 유사한 버전의 MMIS를 구하는 알고리즘을 제시한다.

가. 알고리즘 MMIS

알고리즘 MMIS(그림 1)는 길이 n인 서열 S가 입력으로 주어질 때, S의 MMIS 중 하나를 구한다. 알고리즘은 총 n개의 단계로 구성되고, 단계 i에서 $S[1,i-1]$의 MMIS로부터 $S[1,i]$의 MMIS를 구한다. 즉, 그림 1의 의사코드에서 최초에 W는 빈 문자열이고(라인 1), 단계 i가 시작될 때(라인 2) W는 $S[1,i-1]$의 MMIS이고, 종료될 때(라인 5) W는 $S[1,i]$의 MMIS이다. 단계 i에서 구하는 W를 W_i로 표기하자. 알고리즘은 단계 i에서 W_{i-1}의 문자 중 $S[i]$보다 큰 문자들을 삭제하고(라인 3-4) 마지막에 $S[i]$를 추가하여(라인 5) W_i를 생성한다.

예를 들어, $S=\langle 2,5,5,4,5,4,7,5\rangle$가 주어졌을 때, 단계 진행에 따라 알고리즘은 $W_1=\langle 2\rangle$, $W_2=\langle 2,5\rangle$, $W_3=\langle 2,5,5\rangle$, $W_4=\langle 2,4\rangle$, $W_5=\langle 2,4,5\rangle$, $W_6=\langle 2,4,4\rangle$, $W_7=\langle 2,4,4,7\rangle$, $W_8=\langle 2,4,4,5\rangle$을 구한다. 단계 진행에 따라 W의 길이는 늘어날 수도 있고 줄어들 수도 있다.

다음으로 알고리즘 MMIS의 단계 i에서 구하는 W_i가 $S[1,i]$의 MMIS임을 보인다. $S[i]$를 제외한 W_i의 가장 마지막 문자(즉, 마지막에서 두 번째 문자)가 단계 k에서 추가되었다고 하자. 즉, $W_i=W_k\circ S[i]$이고, W_i의 마지막 두 문자는 $S[k]$와 $S[i]$이다. (만약, $|W_i|=1$이면, $k=0$이다.)

보조정리 1. 단계 i에서 구한 W_i의 마지막 두 번째 문자가 단계 k에서 추가된 문자라고 할 때, $S[k+1,i-1]$의 문자는 모두 $S[i]$보다 크다.

증명. 귀류법을 사용하여 보조정리를 증명한다. $S[k+1,i-1]$에 $S[i]$보다 작거나 같은 문자가 존재한다고 가정하고, 이러한 문자 중 가장 작은 문자를 $S[j](k<j<i)$라 하자. $S[j]$의 값에 따라 다음 두 가지 경우로 나누어 생각해볼 수 있다.

- $S[j]<S[k]$인 경우: $S[k]$가 단계 j(혹은 그 이전 단계)에서 W에서 제외되므로, $S[k]$가 W_i의 문자가 될 수 없다.
- $S[k]\leq S[j]\leq S[i]$인 경우: 단계 j에서 $S[j]$가 W에 포함되고 $S[j+1,i]$의 모든 문자가 $S[j]$보다 크거나 같으므로 $S[j]$는 단계 i까지 W에서 제외되지 않는다. 이는 $S[k]$가 W_i에서 $S[i]$ 바로 앞 문자라는 가정에 모순이 된다.

그러므로 $S[k+1,i-1]$에 $S[i]$보다 작거나 같은 문자는 존재하지 않는다. □

보조정리 2. 알고리즘 MMIS의 단계 i에서 구한 W_i는 $S[1,i]$의 MMIS이다.

증명. 수학적 귀납법을 이용하여 증명한다. 우선 W_1이 $S[1]$의 MMIS임은 자명하다. 다음으로 모든 $1\leq j<i$에 대해, W_j가 $S[1,j]$의 MMIS이면, W_i가 $S[1,i]$의 MMIS임을 보인다. W_i가 $S[1,i]$의 문자들로 구성되고(라인 5) 이 문자들이 단조증가임은 명백하다(라인 3). W_i가 극대조건을 만족시킨다는 점은 귀류법을 이용하여 다음과 같이 증명한다.

증가 부분서열 W_i가 극대 조건을 만족시키지 못한다고 가정하면, $W_i=W_k\circ S[i]$의 어떤 위치에 문자 하나 c를 추가한 단조증가 부분서열이 존재한다. (W_i의 마지막에서 두 번째 문자가 추가된 단계를 단계 k라고 표기함을 상기하자.) 귀납적 가설에 따라 W_k는 $S[1,k]$의 MMIS이다. 추가되는 문자 c의 위치에 따라 다음 두 가지 경우로 나누어 고려한다.

- 문자 c가 W_k과 $S[i]$ 사이에 추가되는 경우: 귀납적 가설에 따라 $W_k\circ c$는 $S[1,k]$의 MMIS가 될 수 없으므로, 문자 c는 $S[k+1..i-1]$에 존재해야 한다. 하지만 보조정리 1에 의해 $c>S[i]$이므로 $W_k\circ c\circ S[i]$는 단조증가 서열이 아니다.
- 문자 c가 W_k의 앞 또는 안에 추가되는 경우: W_k의 앞 또는 안에 c가 추가된 서열을 W'라 표기하자. 보조정리 1에 의해 $S[k+1,i-1]$의 모든 문자는 $S[k]$보다 크므로($\because S[k]\leq S[i]$), W'은 $S[1,k]$의 부분서열이어야 한다. 이는 W_k가 $S[1,k]$의 MMIS라는 귀납적 가설에 어긋난다.

W_i를 포함하면서 W_i보다 긴 $S[1,i]$의 단조증가 서열은 존재할 수 없으므로 W_i는 $S[1,i]$의 MMIS이다. □

알고리즘 MMIS의 시간복잡도와 공간복잡도를 분석한다. 알고리즘 MMIS가 선형시간에 수행된다는 것은 쉽게 보일 수 있다. 라인 3의 while 문이 참이 되는 횟수는 W에서 삭제되는 문자의 수와 동일하다. W에는 S의 각 문자가 많아야 한 번씩만 추가되므로, W에서 삭제되는 문자의 수 역시 최대 n이다. 라인 3의 while 문이 거짓이 되는 경우를 포함하여 다른 문장들은 최대 $n+1$번 수행된다. 메모리 공간의 경우, 입력과 출력에 사용되는 메모리 이외에는 상수 크기의 메모리만을 사용한다. 따라서 다음 정리를 얻을 수 있다.

```
Algorithm MSIS( S[1,n] )
1: W←S[0] (Δ), m←0
2: for i←1 to n
3:   if S[i] is not equal to any character in W
4:     while W[m] > S[i]
5:       delete W[m] from W, m←m−1
6:     W←W∘S[i], m←m+1
7: return W
```

그림 2. MSIS를 구하는 알고리즘

정리 3. 길이 n인 서열 S의 MMIS는 $O(n)$ 시간과 $O(1)$ 작업 공간을 사용하여 구할 수 있다.

나. 알고리즘 MSIS

알고리즘 MSIS(그림 2)는 길이 n인 서열 S가 입력으로 주어질 때, S의 MSIS 중 하나를 구한다. 순증가 조건을 만족시키기 위해 W가 $S[i]$와 값이 동일한 문자를 포함하면 해당 문자를 무시한다는 점을 제외하면(라인 3), 알고리즘 MSIS는 알고리즘 MMIS와 동일하다. 두 알고리즘의 의사코드 차이는 단 한 줄이지만, 동작 과정은 많이 달라진다. 예를 들어, $S=\langle 2,5,2,4,5,4,7,5\rangle$가 주어졌을 때, $W_1=\langle 2\rangle$, $W_2=\langle 2,5\rangle$이다. 단계 3에서 $S[3]$은 $W_2[1]$과 같으므로 아무 변화 없이 $W_3=\langle 2,5\rangle$이다. 이후 단계 진행에 따라 $W_4=\langle 2,4\rangle$, $W_5=W_6=\langle 2,4,5\rangle$, $W_7=W_8=\langle 2,4,5,7\rangle$이다.

알고리즘 MSIS의 정확성을 분석한다. 순증가 부분서열에서는 같은 문자가 중복으로 선택될 수 없으므로 알고리즘 MMIS의 증명과 다른 방식으로 증명한다.

보조정리 4. 알고리즘 MSIS에서 생성하는 최종서열 W의 임의의 어떤 문자가 단계 k에서 추가된 $S[k]$라고 할 때, $S[k+1,n]$에서 W_k의 문자와 값이 다르면서 가장 작은 문자 c는 W에 포함된다.

증명. $S[k+1,n]$에서 c가 처음 나타나는 위치를 j라 하면, $c=S[j]$이고 $S[j]>S[k]$이다. ($S[j]<S[k]$이면 $S[k]$는 단계 j에서 삭제된다.) 알고리즘 MSIS는 단계 j에서 W에서 $S[j]$보다 큰 문자들을 삭제하고 $S[k]$ 다음 문자로 $S[j]$를 추가한다. $S[j]$는 $S[k+1,i]$에서 W_k의 문자가 아니면서 가장 작은 문자이므로, 이후 단계에서 삭제되지 않는다. □

보조정리 5. 알고리즘 MSIS에서 구하는 W는 S의 MSIS이다.

증명. W가 S의 순증가 부분서열임은 자명하다. W가 극대조건을 만족시킴을 귀류법을 이용하여 증명한다. W의 어떤 위치에 문자 c를 추가하여 S의 순증가 부분서열을 만들 수 있다고 가정하자. 문자 c가 추가된 증가서열에서 c 바로 앞 문자가 단계 k에서 추가된 $S[k]$라고 하자. $S[1,k]$는 W_k를 포함하는 가장 짧은 접두사이므로 c는 $S[k+1,n]$에 나타나야 하고, c는 W_k의 문자와 달라야 한다. 문자 c의 값에 따라 다음 두 가지 경우로 나누어 고려한다.

- 문자 c가 $S[k+1,n]$에서 W_k의 문자와 값이 다르면서 가장 작은 문자이면, 보조정리 4에 의해 c는 W에 포함되어야 한다.
- 문자 c가 $S[k+1,n]$에서 W_k의 문자와 값이 다르면서 가장 작은 문자(이를 a라 하자)보다 크면, 보조정리 4에 의해 문자 a는 W에 포함되므로 $S[k]$는 c의 바로 앞 문자가 될 수 없다.

그러므로, W를 포함하는 S의 다른 순증가 부분서열은 존재하지 않는다. □

알고리즘의 수행시간과 사용하는 메모리 공간은 $S[i]$가 W의 문자와 같은지 검사(라인 3)를 어떻게 구현하느냐에 따라 달라진다. $|\Sigma|$ 크기의 배열(각 문자가 W에 포함되었는지 여부를 저장)을 사용하면 알고리즘 수행에는 $O(n)$ 시간과 $O(|\Sigma|)$이 필요하다. 이진 탐색으로 $S[i]$를 W에서 찾는 방식을 사용하면, 알고리즘은 $O(n\log n)$ 시간과 $O(1)$ 공간을 사용한다. 따라서 다음 정리를 얻을 수 있다.

정리 6. 길이 n인 서열 S의 MSIS는 $O(n)$ 시간과 $O(|\Sigma|)$ 작업 공간, 또는 $O(n\log n)$ 시간과 $O(1)$ 작업 공간을 사용하여 구할 수 있다.

Ⅲ. 결 론

본 논문에서는 극대 증가 부분서열을 단조증가와 순증가로 구분하여 정의하고, 극대 단조증가 부분서열(MMIS)과 극대 순증가 부분서열(MSIS)을 구하는 선형 알고리즘을 각각 제시하였다. MMIS를 구하는 선형시간 알고리즘은 상수 크기의 공간을 사용하는 반면, MSIS를 구하는 선형시간 알고리즘은 알파벳 크기에 비례하는 공간을 사용한다. 이론적 측면에서 상수 크기의 공간을 사용하여 MSIS를 선형시간에 구할 수 있는지는 향후 연구되어야 할 문제이다.

본 논문에서 제시한 알고리즘은 극대 증가 부분서열의 길이는 보장해주지 못한다. 극단적인 예로 서열의 맨 마지막 문자가 가장 작으면 알고리즘은 길이가 1인 서열을 답으로 제시한다. 따라서 실용적 측면에서는 수행시간은 많이 증가시키지 않으면서 충분히 긴 길이의 극대 증가 부분서열을 구하는 알고리즘 개발이 필요하다.

REFERENCES

[1] C. Schensted, “Longest increasing and decreasing subsequences,” *Canadian Journal of Mathematics*, 13, pp. 179-191, 1961.

[2] I. Lee, “Mining Regular Expression Rules based on q-grams,” *Smart Media Journal*, Vol. 8, No. 3, pp. 17-22, 2019.

[3] M. L. Fredman, “On computing the length of longest increasing subsequences,” *Discrete Mathematics*, Vol. 11, Issue 1, pp. 29-35, 1975.

[4] J.W. Hunt and T.G. Szymanski, “A fast algorithm for computing longest common subsequences,” *Communications of ACM*, Vol. 20, Issue 5, pp. 350-353, May 1977.

[5] D. Aldous and P. Diaconis, “Longest increasing subsequences: from patience sorting to the Baik-Deift-Johansson theorem,” *Bulletin of the American Mathematical Society*, Vol. 36, No. 4, pp. 413-432, Jul. 1999.

[6] M. Crochemore and E. Porat, “Fast computation of longest increasing subsequences and application,” *Information and computation*, Vol. 208, Issue 9, pp. 1054-1059, Sep. 2010.

[7] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, Cambridge University Press, 1997.

[8] 이인복, “k개의 오차를 허용하는 순위 패턴 매칭,” *스마트미디어저널*, 제9권, 제2호, 33-38쪽, 2020년 6월

[9] 박승현, 이은지, 김판구, “한글 편집거리 알고리즘을 이용한 한국어 철자오류 교정방법,” *스마트미디어저널*, 제6권, 제1호, 16-21쪽, 2017년 3월

[10] R.A. Wagner and M.J. Fischer. “The string-to-string correction problem,” *Journal of the ACM*, Vol. 21, Issue 1, pp. 168 - 173, Jan. 1974.

[11] I.-H. Yang, C.-P. Huang, and K.-M. Chao, “A fast algorithm for computing a longest common increasing subsequence,” *Information Processing Letters* 93 (5), pp. 249 - 253, 2005.

[12] A. Agrawal and P. Gawrychowski, “A Faster Subquadratic Algorithm for the Longest Common Increasing Subsequence Problem,” *Proc. of International Symposium on Algorithms and Computation (ISAAC)*, 4, Hong Kong, China, Dec. 2020.

[13] Y. Sakai, “Maximal common subsequence algorithms,” *Theoretical Computer Science*, vol. 793, pp. 132-139, 2019.

[14] 이동엽, 나중채, “유사도 측정에 대한 극대 공통 부분 서열의 효용성,” *한국차세대컴퓨팅학회 논문지*, vol. 18, no. 3, 53-61쪽, 2022년 6월

[15] 이동엽, 나중채, “더 긴 극대 공통 부분 서열을 찾기 위한 알고리즘,” *정보과학회논문지*, vol. 49, no. 7, 507-513쪽, 2022년 7월

[16] M.J. Atallah and S.R. Kosaraju, “An Efficient Algorithm for Maxdominance, with Applications,” *Algorithmica*, 4, pp. 221-236, 1989.

저 자 소 개

나중채(정회원)

1998년 서울대학교 컴퓨터공학과 학사 졸업.
2000년 서울대학교 컴퓨터공학과 석사 졸업.
2005년 서울대학교 전기컴퓨터공학부 박사 졸업.

<주관심분야 : 알고리즘, 생물정보학, 컴퓨터이론>