최장 공통 부분 서열과 극대 공통 부분 서열의 길이 비교 및 분석

이동엽, 나중채
세종대학교 컴퓨터공학과
dongyeoplee@sju.ac.kr, jcna@sejong.ac.kr

Comparison and Analysis of Lengths of Longest Common Subsequence and Maximal Common Subsequence

DongYeop Lee, Joong Chae Na
Dept. of Computer Science, Sejong University

요 약

최장 공통 부분 서열(Longest Common Subsequence, LCS)은 서열 유사도(Similarity)를 측정하기 위한 주요 지표 중 하나로 특별한 가정이 없는 한 두 문자열의 LCS 를 계산하기 위해서는 두 문자열의 길이의 곱에 비례하는 시간이 필요하다. 최근 최장(longest)이라는 조건을 극대(maximal)로 완화한 극대 공통 부분 서열(Maximal Common Subsequence, MCS)이 제시되었고, 두 문자열의 MCS 를 선형에 가까운 시간에 찾는 알고리즘이 개발되었다. 극대는 최장을 보장하지 않기 때문에 두 문자열의 MCS 길이는 LCS 길이와 달리 유일하지 않을 수 있고, LCS 길이가 매우 길어도 길이가 1 인 MCS 가 존재할 수도 있다. 본 논문에서는 기존 알고리즘에 의해 계산되는 MCS 의 효용성을 알아보기 위해, DNA 등 여러 종류의 실제 데이터와 랜덤 생성된 데이터에 대해 LCS 와 MCS 의 길이를 비교했다. MCS 길이는 LCS 길이 대비 실제 데이터에서 32.1 ~ 60.2%, 랜덤 데이터에서는 27.5 ~ 62.9%로 나타났다. 이 비율은 문자열을 이루고 있는 알파벳 수가 많을수록, 문자열의 길이가 길어질수록 감소했다.

1. 서론

최장 공통 부분 서열(Longest Common Subsequence, LCS 는 최대의 길이를 가지는 공통 부분 서열을 의미한다. 두 문자열의 LCS 의 길이는 두 문자열의 유사도를 측정하는 데 사용할 수 있고 문자열 알고리즘, 분자 생물학 등의 분야에서 응용이 가능하다 [1]. LCS 를 찾는 문제는 컴퓨터 과학의 고전 문제이고 널리 연구되었다. LCS 를 구하려는 문자열들의 길이의 합을 n 이라고 했을 때, Wagner 와 Fisher [2]는 두 문자열의 LCS 를 찾는 문제를 $O(n^2)$의 시간과 공간에 해결할 수 있는 알고리즘을 제시했다. Hirschberg [3]는 두 문자열의 길이에 대해 같은 시간에 선형 공간만을 사용하여 문제를 해결할 수 있는 알고리즘을 제시했다. 또한 strong exponential time hypothesis(SETH)가 옳다는 가정 하에 임의의 양의 상수 ϵ에 대해 $O(n^{2-\epsilon})$ 시간에 LCS 를 구하는 알고리즘은 존재하지 않는다 [4, 5]는 것이 증명되었다.

최근 최장이라는 조건을 완화하여 극대 개념을 사용한 극대 공통 부분 서열(Maximal Common Subsequence, MCS)가 제안되었다 [6]. 공통 부분 서열에 어떤 문자를 추가하여도 더 이상 공통 부분 서열이 발생하지 않는 경우를 극대라고 한다. Sakai [6]는 두 문자열의 MCS 중 하나를 선형에 가까운(sub-linear) $O\left(n\sqrt{\log n/\log\log n}\right)$ 시간과 선형 공간에 찾는 알고리즘을 제시했다.

극대는 최장을 보장하지 않기 때문에 두 문자열의 MCS 길이는 LCS 길이와 달리 유일하지 않을 수 있고, LCS 길이가 매우 길어도 길이가 1 인 MCS 가 존재할 수도 있다. LCS 는 가장 긴 공통 부분 서열이므로 MCS 중 하나이다. 본 논문에서는 기존 알고리즘에 의해 계산되는 MCS 의 효용성을 알아보기 위해, DNA 등 여러 종류의 실제 데이터와 랜덤 생성된 데이터에 대해 LCS 와 MCS 의 길이를 비교했다 실제 데이터인 DNA, 단백질, 영문 텍스트와 여러 가지 크기의 알파벳 집합을 사용하여 랜덤으로 생성된 텍스트 데이터에 대해서 실험했다.

MCS 길이는 LCS 길이 대비 실제 데이터에서 32.1 ~ 60.2%, 랜덤 데이터에서는 27.5 ~ 62.9%로 나타났다. 이 비율은 문자열을 이루고 있는 알파벳 수가 많을수록, 문자열의 길이가 길어질수록 비율은 감소했다.

2. 배경지식

먼저 본 논문에서 사용할 문자열 관련 표기는 다음과 같다. 길이가 m인 문자열 X에 대해 각 문자를 $x_1, x_2, \dots, x_m$이라고 표시하고, 문자열의 길이는 $|X|$로 표시한다. $0 < i, j < |X|$인 i와 j에 대해 X의 i번째 문자부터 j번째 문자까지 연속하여 이어 붙인 경우 $X[i, j]$로 표기한다. 이때 i가 j보다 큰 경우 $X[i, j]$는 빈 문자열을 의미하는데 빈 문자열은 ε으로 표기한다. 위의 표현을 종합하여 문자열 X에 대해 표현하면 다음과 같다. $X = X[1, |X|] = X[1, m]$. 문자열의 i번째 문자까지의 접두사는 X_i로 표기하고 $X_i = X[1, i]$이다.

2. 1. LCS

부분 서열(subsequence)은 문자열에서 임의의 위치에서 임의의 문자들을 삭제하여 얻어지는 서열을 의미한다. 예를 들어 “ABCDE” 라는 문자열에서 ‘A’와 ‘C’ 두 개의 문자를 삭제하여 얻어낸 “BDE”는 “ABCDE”의 부분 서열이다. 두 문자열의 공통 부분 서열은 하나의 문자열의 부분 서열이 다른 문자열의 부분 서열도 되는 경우를 말하며 공통 부분 서열은 여러 개 존재할 수 있다. 예를 들어 “ABCDE”와 “ABDE”의 경우 “ABDE”, “ABE”, “AD” 등이 공통 부분 서열이 될 수 있다. 최장 공통 부분 서열은 최대의 길이를 가지는 공통 부분 서열을 의미하며 여러 개 존재할 수 있다. 앞에서 예를 들었던 “ABCDE”와 “ABDE”의 경우 공통 부분 서열 중 가장 긴 “ABDE”가 LCS 가 된다. 각각 길이가 m과 n인 두 문자열 X와 Y의 LCS 를 구한다고 했을 때 $LCS(X_i, Y_j)$는 X와 Y의 접두사 X_i와 Y_j의 LCS 의 길이라 하면 LCS 의 관계식은 다음과 같다.

$$LCS(X_i, Y_j) = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0 \\ LCS(X_{i-1}, Y_{j-1}) + 1, & \text{if } x_i = y_j \\ \max\left(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)\right), & \text{if } x_i \neq y_j \end{cases}$$

$LCS(X_m, Y_n)$이 X와 Y의 LCS 의 길이이고 이는 동적 프로그래밍을 이용하여 두 문자열의 길이의 곱에 비례하는 시간에 구할 수 있다. 두 문자열의 길이가 각각 n이라 하면 $O(n^2)$의 시간 복잡도를 가진다.

2. 2. MCS

두 문자열의 공통 부분 서열에 어떠한 문자를 추가하여도 더 이상 공통 부분 서열이 발생하지 않는 경우를 극대라 하고 이 조건을 만족하는 공통 부분 서열을 두 문자열의 MCS 라 한다. 두 문자열의 MCS 는 여러 개 있을 수 있다. 길이가 조건인 LCS 와 달리 MCS 는 최장을 보장하지 않기 때문에 두 문자열의 MCS 의 길이는 유일하지 않고, LCS 의 길이가 매우 길어도 길이가 1 인 MCS 가 존재할 수도 있다. 간단한 예를 들어 “ABBB”와 “BBBA” 라는 문자열이 있을 때 LCS 는 길이가 가장 긴 “BBB”가 되지만 MCS 는 “A”, “BBB” 두 가지 모두 극대 조건을 만족한다.

두 문자열의 MCS 는 [6]에서 증명한 보조정리 1.을 이용하여 찾는다. 다음은 보조정리 1. 그리고 그 이해에 필요한 정의 세가지이다.

정의 1. 공통 부분서열 W를 부분 서열로 가지는 문자열 X와 Y의 가장 짧은 접두사의 각 문자열에서의 마지막 인덱스를 g_W와 h_W라 한다.

정의 2. 공통 부분서열 W의 마지막 글자 $W[|W|]$를 포함하는 문자열 X와 Y의 가장 짧은 접미사의 각 문자열에서의 첫 인덱스를 i_W와 j_W라 한다.

정의 3. 문자열 X와 Y가 ε을 제외한 공통 부분 서열을 하나도 가지지 않는 경우 문자열 X와 Y는 서로소라고 한다.

보조정리 1. 문자열 X와 Y의 ε이 아닌 임의의 공통 부분 서열 W에 대해, $X[g_W + 1, |X|]$와 $Y[h_W + 1, |Y|]$가 서로소인 경우, $W[1, |W| - 1]$ 을 포함하는 $X[1, i_W - 1]$과 $Y[1, j_W - 1]$의 MCS 에 $W[|W|]$을 이어 붙인 것은 극대이다.

$X[g_W + 1, |X|]$와 $Y[h_W + 1, |Y|]$가 서로소인 X와 Y의 공통 부분 서열 W를 찾고 마지막 글자 $W[|W|]$를 MCS 의 마지막 글자로 추가한다. 이후 $X[1, i_w - 1]$과 $Y[1, j_w - 1]$의 MCS 를 찾는다. 이 과정을 MCS 를 찾는 문자열이 빈 문자열이 될 때까지 반복한다. Beame 과 Fich [7]의 자료구조를 사용했을 때 두 문자열의 길이가 모두 n이라 하면 $O\left(n\sqrt{\log n / \log\log n}\right)$의 시간에 찾을 수 있다.

3. 실험 및 결과

본 논문에서는 [2]에서 제시한 두 문자열의 LCS 를 찾는 알고리즘과 [6]에서 제시한 두 문자열의 MCS 를 찾는 알고리즘을 사용하여 프로그램을 작성했다. 프로그램은 C++로 작성되었으며 컴파일러는 VC++17 최적화 단계는 O2 로 컴파일 했다. Intel Core i7-11700 2.5Ghz CPU, 16GB ram 기기에서 Windows 10 환경으로 실험했다.

실험에는 실제 데이터와 랜덤 데이터를 사용했다. 실제 데이터로는 문자열 관련 알고리즘 연구에서 성능 테스트를 위해 많이 사용되는 말뭉치(corpus)인 Pizza & Chili corpus [8]의 텍스트 컬렉션 중 DNA, 단백질, 영문 텍스트 데이터를 사용했다.

<표 1> 실험에 사용한 실제 데이터 정보

Data	$\|\Sigma\|$	length	Inv prob.
DNA	16	1,184,051,855	3.91
Protein	27	403,927,746	17.02
Text	239	2,210,395,553	15.25

<표 1>은 실제 데이터로 사용한 텍스트 컬렉션에 대해 [8]에서 제공한 정보이다. $|\Sigma|$는 알파벳 집합의 크기이고 Inv prob.은 무작위로 두 개의 원소를 뽑았을 때 일치할 확률이다. 데이터에 대한 더 자세한 설명은 다음과 같다.

- DNA 는 염기 정보가 하나의 영문 알파벳 대문자로 표현된 데이터이다. 대부분의 데이터가 A, G, C, T 4 개의 문자로 이루어져 있으며, 추가로 12 개의 거의 나타나지 않는 특별 케이스 처리용 문자를 포함 총 16 개의 문자로 구성된다.
- 단백질 데이터는 대부분 아미노산을 표현하는 20 개의 알파벳 대문자로 구성되어 있으며, 추가로 7 개의 거의 나타나지 않는 특별 케이스 처리용 문자를 포함 총 27 개의 문자로 구성되어 있다.
- 영문 텍스트 데이터는 실제 문서로 사용된 텍스트를 모은 데이터로 239 개의 문자로 구성된다.

실제 데이터가 모두 하나의 긴 텍스트 파일로 되어 있기 때문에 실험에 알맞게 일부를 추출한 데이터를 여러 개 만들어 실험했다. DNA, 단백질 데이터는 하나의 파일 내에서 줄 바꿈 부호로 구분되어 있다. 실험에 사용할 데이터는 구분된 여러 라인에서 하나의 라인을 랜덤으로 선택하여 100, 200, …, 1000 까지 총 10 개의 길이를 추출한 데이터를 사용했다. 추출한 데이터 두 개를 한 쌍으로 하여 각 길이 마다 총 100 쌍의 데이터를 랜덤으로 추출했다. 영문 텍스트 데이터의 경우 모든 공백과 줄 바꿈 부호를 제거하고 랜덤 위치에서 시작하여 마찬가지로 100, 200, …, 1000 까지 총 10 개의 길이에 대해 각 100 쌍의 데이터를 추출했다.

랜덤 데이터로는 알파벳 크기에 따른 변화를 비교하기 위해 알파벳 집합의 크기를 여러가지로 하여 랜덤 생성한 데이터를 사용했다. 알파벳 집합의 크기를 4, 8, 16, 64, 128 로 하여 실제 데이터와 마찬가지로 100, 200, …, 1000 까지 총 10 개의 길이에 대해 각 100 쌍의 데이터를 만들어 실험했다.

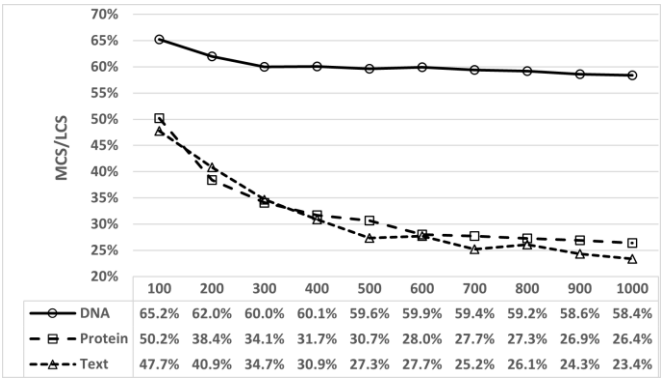
3. 1. 실제 데이터 실험결과

<표 2>는 각 실제 데이터들에 대해 LCS 알고리즘과 MCS 알고리즘을 수행한 결과이다. 표에서 Entropy 를 제외한 나머지 값들의 단위는 모두 % 이다. MCS 와 LCS 를 구하는 각 문자열 쌍을 문자열 X와 Y라고 할 때 MCS/$|X|$는 MCS 길이를 MCS 를 구한 데이터 쌍 중 하나의 길이인 $|X|(=|Y|)$로 나눈 값의 평균이다. LCS/$|X|$도 LCS 에 대해 이와 같은 방법으로 구한다. MCS/LCS 는 각 문자열 쌍에서 구한 MCS 길이를 LCS 의 길이로 나눈 값의 평균이다. Entropy 는 실험에 사용된 각 데이터의 정보 엔트로피의 평균값으로 본 실험에서는 Shannon 엔트로피 [9]를 사용했다.

<표 2> 실제 데이터에 대한 실험 결과

Data	(%) MCS/$\|X\|$	(%) LCS/$\|X\|$	(%) MCS/LCS	Entropy
DNA	37.1	61.7	60.2	1.939
Protein	11.4	35.9	32.1	4.037
Text	17.5	41.9	38.5	4.654

DNA 는 세 데이터 중 LCS 와 MCS 의 길이가 가장 길고 LCS/MCS 비율 또한 60.2 로 가장 크다. 단백질의 경우 LCS 와 MCS 의 길이가 가장 짧게 나타나고 MCS/LCS 비율 또한 32.1 로 가장 작다. 영문 텍스트 데이터의 경우 239 개의 가장 많은 알파벳으로 구성되었음에도 불구하고 단백질보다 LCS, MCS 모두 더 길게 나타나고, MCS/LCS 비율 또한 38.5 으로 높게 나타난다. 대체로 알파벳 집합의 크기가 커져 문자의 종류가 늘어나고 각 문자가 나타날 확률이 줄어들 경우 MCS 와 LCS 의 길이가 줄어들게 된다. DNA 와 단백질을 비교한 경우 알파벳 집합의 크기가 16 에서 27 로 늘어났을 때 엔트로피는 각각 1.939 에서 4.037 로 큰 차이를 보인다. 하지만 영문 텍스트 데이터의 엔트로피는 알파벳 집합 크기가 훨씬 큰 239 임에도 비교적 높지 않은 4.654 이다. 구성하는 알파벳은 많으나 그 중 특정 문자가 훨씬 자주 나타나기 때문이다.



(그림 1) 실제 데이터에서 각 데이터별 MCS 와 LCS 길이의 비

(그림 1)은 각 데이터에서 LCS 와 MCS 를 구하는 문자열의 길이 변화에 따라서 MCS/LCS 비율의 변화 양상을 표현한 그래프다. 가로축은 LCS 와 MCS 를 구한 문자열의 길이이고 세로축은 MCS/LCS 비율이다. 전체적으로 문자열이 길어질수록 MCS/LCS 비율이 점점 작아지는 경향을 보인다. 이는 LCS 와 MCS 를 구하려는 문자열의 길이가 길어질수록 LCS 에 비해 길이가 짧은 MCS 를 알고리즘에서 찾게 될 확률이 높아지기 때문이다. DNA 데이터의 경우 대부분이 A, G, C, T 4 글자로 이루어져 있기 때문에 데이터가 길어져도 비교적 MCS/LCS 비율이 완만하게 감소하지만 알파벳 집합 크기가 비교적 큰 단백질이나 영문 텍스트의 경우 빠르게 감소하는 것을 볼 수 있다.

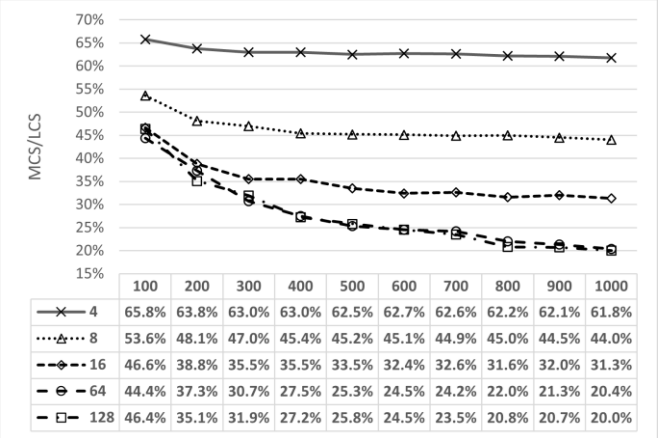
3. 2. 랜덤 데이터 실험결과

<표 3>은 알파벳 집합의 크기를 서로 다르게 랜덤으로 생성한 데이터에 대한 실험 결과이다. |Σ|는 랜덤 생성하는데 사용한 알파벳 집합의 크기를 의미한다. MCS/$|X|$, LCS/$|X|$, MCS/LCS, Entropy 는 모두 표 2 와 동일한 지표이고 동일한 단위를 사용한다.

<표 3> 랜덤 데이터에 대한 실험 결과

|Σ|	(%) MCS/$\|X\|$	(%) LCS/$\|X\|$	(%) MCS/LCS	Entropy
4	40.2	64.0	62.9	1.933
8	23.1	50.0	46.2	2.985
16	13.3	38.2	34.9	3.968
64	5.7	20.8	27.7	5.855
128	4.0	15.0	27.5	6.704

랜덤 데이터에서는 알파벳 집합이 커질수록 MCS/$|X|$, LCS/$|X|$, MCS/LCS 모두 감소하는 것을 볼 수 있다. 이는 알파벳 집합이 커지기 때문에 각 문자가 문장 내에서 나타나는 빈도가 감소하기 때문이다. 각 문자의 등장 빈도 감소는 엔트로피의 증가를 통해서도 알 수 있다.



	100	200	300	400	500	600	700	800	900	1000
4	65.8%	63.8%	63.0%	63.0%	62.5%	62.7%	62.6%	62.2%	62.1%	61.8%
8	53.6%	48.1%	47.0%	45.4%	45.2%	45.1%	44.9%	45.0%	44.5%	44.0%
16	46.6%	38.8%	35.5%	35.5%	33.5%	32.4%	32.6%	31.6%	32.0%	31.3%
64	44.4%	37.3%	30.7%	27.5%	25.3%	24.5%	24.2%	22.0%	21.3%	20.4%
128	46.4%	35.1%	31.9%	27.2%	25.8%	24.5%	23.5%	20.8%	20.7%	20.0%

(그림 2) 랜덤 데이터에서 알파벳 집합 크기 별 MCS 와 LCS 길이의 비

(그림 2)는 랜덤 데이터에서 알파벳 크기에 따라 MCS 를 LCS 의 길이로 나눈 비의 변화 양상을 표현한 그래프다. 실제 데이터와 비슷하게 전체적으로 감소하는 추세를 보인다. 또한 알파벳 집합 크기가 적은 경우는 완만하게 감소 추세를 보이지만 알파벳 집합 크기가 클수록 더 빠르게 감소하는 것을 볼 수 있다. 이는 실제 데이터와 마찬가지로 데이터 길이가 길어지면서 길이가 낮은 MCS 를 알고리즘에서 찾게 될 확률이 높아지기 때문이다.

4. 결론

MCS 길이는 LCS 길이 대비 실제 데이터에서 32.1 ~ 60.2%, 랜덤 데이터에서는 27.5 ~ 62.9%로 나타났다. 본 논문에서 사용한 MCS 알고리즘은 두 문자열의 MCS 중 단 하나만을 찾는 알고리즘이다. 따라서 실험 결과에서 나타난 대로 MCS 중 비교적 짧은 것을 찾게 될 경우 LCS 와의 길이 차이가 많이 날 수 있다. 이는 문자열이 길어질수록 알파벳 집합의 크기가 클수록 높은 확률로 발생한다.

LCS 를 구하는 데는 일반적으로 $O(n^2)$ 시간이 필요한 반면 MCS 는 선형에 가까운 시간이 필요하다. 이는 특히 큰 데이터를 처리할 때 두드러지게 나타난다. 따라서 현재의 MCS 알고리즘 보다 LCS 에 더 가까운 길이의 MCS 를 선형시간에 가깝게 찾을 수 있는 방법에 대한 연구의 필요성이 있다.

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (2020R1F1A1068873).

참고문헌

[1] L. Bergroth, H. Hakonen, and T. Raita, “A Survey of Longest Common Subsequence Algorithms,” Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000, pp. 39-48, 2000.

[2] Robert A. Wagner and Michael J. Fischer, “The String-to-String Correction Problem,” Journal of the ACM, Vol. 21, No. 1, pp. 168-173, 1974.

[3] D. S. Hirschberg, “A linear space algorithm for computing maximal common subsequences,” Communications of the ACM, 18, 6, pp. 341-343, 1975.

[4] A. Abboud, A. Backurs, and V. V. Williams, “Tight Hardness Results for LCS and Other Sequence Similarity Measures,” 2015 IEEE 56th Annual Symposium on Foundations of Computer Science, pp. 59-78, 2015.

[5] K. Bringmann and M. Künnemann, “Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping,” 2015 IEEE 56th Annual Symposium on Foundations of Computer Science, pp. 79-97, 2015

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

[7] P. Beame and F. E. Fich, “Optimal Bounds for the predecessor problem and related problems,” Journal of Computer and System Sciences, Vol. 65, No. 1, pp. 38-72, 2002.

[8] P. Ferragina and G. Navarro, Pizza&Chili Corpus http://pizzachili.dcc.uchile.cl/texts.html

[9] C. E. Shannon “A mathematical theory of communication,” The Bell System Technical Journal, Vol. 27, No. 3, pp. 379-423, 1948