한글에 대한 편집 거리 문제
(Edit Distance Problem for the Korean Alphabet)

노 강 호 † 김 진 욱 †† 김 은 상 † 박 근 수 ††† 조 환 규 ††††
(Kangho Roh) (Jin Wook Kim) (Eunsang Kim) (Kunsoo Park) (Hwan-Gue Cho)

요 약 문자열에 대한 편집 거리 문제는 하나의 문자열을 다른 문자열로 변환할 때 필요한 최소한의 연산의 개수를 구하는 문제이다. 편집 거리 문제는 오랫동안 연구가 진행되어 왔으며, 영어와 같이 1차원 문자열에 대해서는 최적해를 찾는 여러 가지 알고리즘이 개발되어 왔다. 그러나 한글 또는 한자와 같이 좀 더 복잡한 언어에 대한 편집 거리에 대해서는 많은 연구가 진행되지 못했다. 본 논문에서는 한글이 갖는 특징을 반영한 편집 거리를 정의하고, 한글 문자열에 대한 편집 거리를 구하는 알고리즘을 제안한다.
키워드 : 편집 거리, 한글, 알고리즘

Abstract The edit distance problem is finding the minimum number of edit operations to transform a string into another one. It is one of the important problems in algorithm research and there are some algorithms that compute an optimal edit distance for the one-dimensional languages such as the English alphabet. However, there are a few researches to find the edit distance for the more complicated language such as the Korean or Chinese alphabet. In this paper, we define the measure of the edit distance for the Korean alphabet and present an algorithm for the edit distance problem for the Korean alphabet.
Key words : edit distance, the korean alphabet, algorithm

1. 서 론

문자열 A와 B에 대한 편집 거리는 A를 B로 바꾸기 위하여 필요한 최소한의 연산(삽입, 삭제, 치환)의 개수를 나타낸다. 편집 거리 문제에 대해서는 동적 프로그래밍을 이용하여 $O(|A||B|)$ 시간 안에 최적해을 구하는 여러 방법들이 알려져 있다[1,2]. 그러나 대부분의 기존 연구는 알파벳 기반의 영어나 A, C, T, G 등으로 이루어진 유전자 서열등을 대상으로 하는 경우들이 많았고, 영어권의 언어가 아닌 좀 더 복잡한 형태의 언어들에 대해서는 많은 연구가 진행되지 않았다. 본 논문에서는 이러한 언어들 중에서 한국어, 즉 한글에 대한 편집 거리를 정의하고 그것을 구하는 알고리즘을 제시하였다.

1.1 한글의 특징 및 두 가지 Naive한 방법

영어가 알파벳의 1차원 배열 형태를 갖는 것과 달리 한글은 하나 이상의 음소(자음+모음)로 이루어진 음절의 배열로 이루어진다. 한글에는 모음(vowel, V)과 자음(consonant, C)의 2가지 종류의 음소가 존재한다. 한글의 음절은 V, CV, CVC의 3가지 형태가 존재하는데, V의 경우에는 무음가 "ㅇ"을 붙임으로써 실생활에서는 CV, CVC의 2가지 형태의 음절이 존재하게 된다. 본 논문에서는 이를 조금 확장하여 오타 등으로 인해 자주 보게 되는 자음 또는 모음 하나로만 이루어진 경우를 포함하여, C, V, CV, CVC의 4가지 형태의 음절을 고려하였다. 좀 더 나아가 언어학적으로 모음과 자음을 단모음/이중모음, 단자음/이중자음등으로 좀 더 자세히 구분하면, CCVC, CVCC, CCVCC, CVVC등의 복잡한 형태들이 존재할 수 있으나, 본 논문에서는 문제의 단순성

· This work was supported by Korea Research Council of Fundamental Science and Technology.

† 학생회원 : 서울대학교 전기컴퓨터공학부
kRoh@holybelief.com
eskim@theory.snu.ac.kr
†† 정 회 원 : 인하대학교 컴퓨터정보공학부 교수
gnugi@inha.ac.kr
††† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
kpark@theory.snu.ac.kr
†††† 정 회 원 : 부산대학교 정보.컴퓨터공학부 교수
hgcho@pusan.ac.kr
논문접수 : 2009년 10월 28일
심사완료 : 2009년 11월 20일

정보과학회논문지: 시스템 및 이론 제37권 제2호(2010.4)

을 위하여 이중자음/이중모음 등의 구분은 고려하지 않았다.

영어에서는 두 문자열의 편집 거리를 구할 때, 삽입, 삭제, 치환의 3가지 편집 연산을 사용한다[1-3]. 한글에 대한 편집 거리를 구하는 간단한 방법으로 음절 단위로 위의 3가지 편집 연산을 적용하는 방법을 생각할 수 있는데, 이를 *SylED* (음절 기반의 편집 거리) 알고리즘이라고 부르도록 하자. 예제 1은 음절단위의 삽입, 삭제, 치환의 3가지 편집 연산의 비용을 1로 하였을 경우에 대한 예제이다. 문자열 A 와 B의 편집 거리를 $\delta(A,B)$라고 표현하면, $\delta(A,B)=\delta(A,C)=\delta(A,D)=1$이 되어 문자열 A와 B, C, D와의 거리는 모두 같게 된다. 그러나 단어의 의미 관점에서 접근해 보면 B는 A의 간단한 오타로 생각할 수 있을 정도로 가까운 단어가 되지만, C는 "파리에 실존하는 공원 이름"[1], D는 "울산대공원의 오타" 등으로 해석이 가능하여, A와는 완전히 다른 뜻의 단어가 될 수 있다. 따라서 영어에서 하나의 문자가 차이나는 것과 한글에서의 한 음절이 바뀌는 경우는 서로 같은 정도의 의미 변화를 반영하기 힘들다고 할 수 있으므로, 이러한 방법은 적합한 편집 거리에 대한 정의라고 말하기가 힘들다.

예제 1. 아래의 표는 A = "서울대공원"과 A와 유사한 몇 가지 단어들을 보여주고 있다. 음절 단위의 편집 거리를 사용하면 A와 B는 하나의 음절이 다르므로 편집 거리는 1이 된다. 마찬가지로 A와 C의 편집 거리도 1이 되며, B와 C의 경우는 2의 값을 갖게 된다.

A	서 울 대 공 원
B	서 울 대 고 원
C	서 울 - 공 원
D	- 울 대 공 원
E	서 - 대 공 원

□

또 다른 방법으로는 한글의 음절을 여러 음소로 구분한 후, 음소 단위에 대해 편집 거리를 구하는 방법을 생각해 볼 수 있다. 즉 배치되는 음절들을 비교하여 음소 단위로 발생하는 삽입, 삭제, 치환에 대해서 비용을 계산하는 방법이다. 이후부터는 이러한 방법을 *PhoED* (음소 기반의 편집 거리) 알고리즘이라고 부르도록 하겠다. 예제 2는 예제 1의 문자열에 *PhoED*을 적용했을 때의 예를 보여주고 있다.

예제 2. 예제 1의 문자열 A와 B에 대해서, *PhoED*에서의 편집 거리를 구하면, $\delta(A,B)=\delta(\text{"공"},\text{"고"})=1$이 된다. 동일한 방법으로 A, C간의 거리를 구하면, $\delta(A,C)=\delta(\text{"대"},-)=2$ 된다. □

1) 서울특별시와 프랑스 파리시의 자매결연 10주년(2001년 11월 12일)을 기념해서 한·불 우호의 상징으로 파리에 설립된 공원으로 2002년 3월 25일 파리의 아클리마타시옹 공원 안에 설립되었다.

위의 예제에서 보듯이 *PhoED*가 *SylED*에 비해서 조금 더 세밀하게 단어들을 구분할 수 있음을 알 수 있다. 그러나 이러한 방법을 사용할 경우에는 아래와 같은 문제가 발생한다. $\delta(A,D)$는 2인 반면에 A와 E는 "ㅇ", "ㅜ", "ㄹ"의 3개 음소가 다르기 때문에 $\delta(A,E)=3$이 된다. 그러나 위의 두 케이스는 모두 하나의 음절이 통째로 사라진 경우이기 때문에, $\delta(A,D)=\delta(A,E)$가 되는 것이 한글의 입장에서는 더 합리적이라고 말할 수 있다. 결론적으로 위의 2가지 알고리즘들은 적합하지 않은 방법이라고 생각된다.

2. 한글 편집 거리의 정의

앞의 분석으로부터 알 수 있듯이 한글은 음소보다는 음절의 영향력이 더 크기 때문에, 하나의 음절이 다른 경우에는 동일한 거리를 갖는 것이 더 적합하다고 할 수 있다. 그래서 두 음절 간의 거리를 음절을 구성하는 서로 다른 음소의 개수로 정의하되 대응되는 모든 음소들이 서로 다른 경우나 음절이 통째로 삽입, 삭제된 경우에는 더 큰 비용을 부여하여 음소보다 음절에 더 비중을 둔 음절 편집 거리를 정의해보았다. 표현의 편의를 위하여 초성, 중성, 종성이 모두 존재하지 않는 공백 음절과 공백 음소를 각각 Λ, λ로 표기하도록 하겠다.

편집 거리를 정의하기 위하여 먼저 아래와 같은 3개의 음소 단위의 편집 연산(phoneme edit operation, *PEO*)을 정의하였다. 음절이 최대 3개의 음소를 가지므로, 두 음절 간에는 최대 3개까지의 *PEO*가 발생하게 된다. 예를 들어 "강"→"감"의 경우는 종성의 "ㅇ"이 "ㅁ"으로 바뀌었으므로 1개의 음소-치환이 발생하며, "가"→"남"의 경우는 "ㄱ"이 "ㄴ"으로 바뀌고 "ㅁ"이 추가되었으므로, 음소-치환과 음소-삽입이 발생하게 된다. 또 다른 예로 "가"→Λ의 경우는 2개의 음소-삭제가 발생하게 된다. 단, 이때 *PEO*는 각각 대치되는 초성, 중성, 종성에 대해서만 정의하도록 하겠다. 즉 "강"→"고"의 경우 $P(\text{"ㅏ"},\text{"ㅗ"})$, $P(\text{"ㅇ"},\lambda)$는 가능하지만 $P(\text{"ㅏ"},\lambda)$, $P(\text{"ㅇ"},\text{"ㅗ"})$와 같이 되지는 않는다. 그리고 $P(\lambda,\lambda)$은 정의하지 않도록 한다.

- 음소-삽입 $P(\lambda,a)$: 음소 a 삽입
- 음소-삭제 $P(a,\lambda)$: 음소 a 삭제
- 음소-치환 $P(a,b)$: 음소 a를 b로 치환

다음에는 앞에서 정의한 *PEO*를 이용하여 음절 단위의 편집 연산 (syllable edit operation, *SEO*)을 정의하였다. 두 음절이 주어졌을 때, 각각을 구성하는 음소의 종류에 따라 다음과 같은 3가지 형태의 *SEO*가 가능하게 된다.

- 음절-삽입 $S(\lambda,A)$: 음절 A 삽입

• 음절-삭제 $S(A,\lambda)$: 음절 B 삭제

• 음절-치환 $S(A,B)$: 음절 A를 B로 치환

음절-삽입은 공백 음절 Λ에 음절이 삽입된 것을 뜻하며, 음절이 하나 이상의 음소를 갖기 때문에 하나 이상의 음소-삽입으로 이루어지게 된다. 반대로 음절-삭제는 하나 이상의 음소-삭제를 갖는다. 치환 *SEO*는 하나 이상의 *PEO*로 이루어지며 3종류의 *PEO*가 모두 나타날 수 있다.

단 *SEO*를 정의할 때, "ㄱ"와 같이 하나의 자음으로만 이루어진 음절은 예외적으로 해당 자음을 초성으로 생각하도록 하겠다. 하나의 자음으로 이루어진 음절은 초성만 있는 경우와 종성만 있는 경우가 모두 가능하지만, 자음 하나만 적은 경우는 초성을 의미할 경우가 많다고 생각할 수 있다. 영어에서도 영어 문장을 발음 중심의 음절로 나누어서 두 문장의 유사도를 비교하는 방법이 있는데, 각 음절의 첫 알파벳이 틀릴 확률이 다른 위치의 알파벳이 다를 경우보다 작다는 통계적인 연구 결과를 사용하고 있다[4]. 따라서 본 논문에서도 이와 비슷한 접근 방식을 사용하여 하나의 자음으로 이루어진 음절을 초성으로 한정하였다. 따라서 "ㄱ"→"낭"에 대해서 P("ㄱ","ㄴ")는 가능하지만 P("ㄱ","ㅇ")과 같은 *PEO*는 정의할 수 없다.

두 문자열의 편집 거리를 정의하기 위하여 먼저 두 음절 A, B간의 거리 $\delta_S(A,B)$를 다음과 같이 정의하였다. 먼저 음절-삽입, 음절-삭제에 대한 비용을 β로 정의한다. 즉 $\delta_S(\Lambda,A)=\delta_S(A,\Lambda)=\beta$가 된다. 음절-치환은 두 음절을 이루고 있는 음소의 종류에 따라 두 가지 방법으로 정의된다. 두 음절을 비교할 때에는 대치되는 초성, 중성, 종성에 따라 최대 3개의 *PEO*가 발생할 수 있는데, 각각의 *PEO*의 비용은 α로 정의하며, 음절간의 거리는 대응되는 모든 *PEO*의 비용의 합으로 정의한다. 즉 두 음절간의 *PEO*의 개수가 N이면 $\delta_S(A,B)=N\times\alpha$가 된다. 그러나 만약 대치되는 모든 음소가 서로 다른 경우에는 음절 전체가 다른 경우가 되므로 음절-삽입, 음절-삭제와 같이 β를 부여하였다. 이는 "가"→"너", "각"→"넌"과 같이 모든 음소가 달라지는 경우에 동일한 비용을 부여하기 위함이다. 그리고 음절을 비교할 때 최대 3개의 *PEO*가 발생할 수 있으므로 β는 $3\times\alpha$ 보다 크거나 같도록 하여, 음절이 달라지는 것이 음소가 달라지는 것보다 더 큰 거리를 갖도록 정의하였다. 이러한 내용을 정리하면 다음과 같다.

• Case 1: 음절-삽입: $d_S(\Lambda,B)=\beta$

• Case 2: 음절-삭제: $d_S(A,\Lambda)=\beta$

• Case 3: 음절-치환: $d_S(A,B)=\beta$, A와 B의 대응되는 모든 음소들에 대하여 음소-삽입, 음소-삭제, 음소-치환이 발생한 경우

• Case 4: 음절-치환: $d_S(A,B)=N\times\alpha$, A와 B의 대응되는 초성, 중성, 종성 중에서 최소 하나 이상의 위치의 값이 같은 경우, N은 서로 다른 음소의 개수를 나타냄

예제 3. 음절-삽입, 음절-삭제의 정의에 의하여 "가"→Λ, Λ→"가"는 모두 β의 값을 갖는다(Case 1, 2). 다음으로 "가"→"나"의 경우는 하나의 *PEO* "ㄱ"→"ㄴ"이 발생하였으므로 δ_S("가","나")=α가 된다(Case 4). 비슷한 방법을 적용하여 δ_S("경","각")을 구하면, P("ㅕ","ㅏ"), P("ㅇ","ㄱ")이 발생하였으므로 2α가 되며, δ_S("ㄱ","가")는 α의 값을 갖게 된다(Case 4). δ_S("ㄱ","악")의 경우는 자음 하나로 이루어진 음절을 초성으로 간주하기로 하였으므로, "ㅇ"과 "ㅏ"가 삽입된 2α가 아니라 "ㄱ"→"ㅇ", λ→"ㅏ", λ→"ㄱ"이 되어 모든 음소가 서로 다른 음소로 치환되었으므로 β의 값을 갖게 된다(Case 3). □

두 문자열 A, B에 대해서 A를 B로 바꾸는 *SEO*들의 모음 $\sigma=\sigma_1,\ldots,\sigma_n$을 아래와 같이 정의하도록 하겠다. *SEO* σ_k를 적용하였을 때 $A\rightarrow B$가 된다면 σ_k를 $\langle\sigma_k : A\rightarrow B\rangle$와 같이 표현하도록 하겠다. $\sigma=\sigma_1,\ldots,\sigma_n$에 대해서, 문자열 $A_0,A_1,\ldots,A_n$이 주어졌을 때 $A=A_0$, $B=A_n$, $\langle\sigma_k : A_{K-1}\rightarrow A_K\rangle$를 만족하면 σ를 A를 B로 변환하는 ***SEO*의 서열**이라고 부르기로 하자. 그리고 σ_k가 $A[i]$를 $B[j]$로 변환하는 *SEO*라고 하면 σ_k의 거리 $\delta_S(\sigma_k)$는 $\delta_S(\sigma_k)=\delta_S(A[i],B[j])$와 같이 표기하도록 하겠다. 그리고 *SEO*의 서열 σ에 대한 비용은 모든 *SEO*의 비용의 합, 즉 $\delta_S(\sigma)=\Sigma_{k=1}^{n}\delta_S(\sigma_k)$로 정의하도록 하겠다.

A를 B로 변환하는 *SEO*의 서열은 여러 가지가 존재할 수 있다. 그 중에서 비용이 가장 작은 서열을 A와 B의 음절 편집 거리(*SED*)로 정의하고, *SED*(A,B)로 표기하도록 하겠다. *SEO*의 서열과 *SED*에는 아래와 같은 특징이 있다[2].

• **Property 1**: 문자열 A, B, C가 주어질 때, $SED(A,C)\leq SED(A,B)+SED(B,C)$이 성립한다.

• **Property 2**: 두 음절-치환 σ_i, σ_j가 각각 문자열 A, B에 대하여 $A[i_1]\rightarrow B[j_1]$, $A[i_2]\rightarrow B[j_2]$라고 할 때, $i_i<i_2\Leftrightarrow j_1<j_2$이 성립한다.

*SEO*의 서열은 다음과 같은 배치(alignment)로 표현할 수 있다. 아래 그림은 "일반통계학"을 "일방통행"으로 바꾸는 두 가지 배치를 보여주고 있다. 왼쪽의 배치는 "반"→"방", "계"→Λ, "학"→"행"의 3가지 *SEO*를 표

현하고 있으며, 그 비용은 δ_S("반","방")+δ_S("계",Λ)+δ_S("학","행")=$\alpha+\beta+2\alpha=3\alpha+\beta$가 된다. 오른쪽 그림이 보여주는 배치의 경우에는 "반"→Λ, "통"→"방", "계"→"통", "학"→"행"의 4가지 *SEO*가 발생하였으며, 비용은 $\beta+2\alpha+\beta+2\alpha=4\alpha+2\beta$가 된다.

```
일 반 통 계 학        일 반 통 계 학
| | | | |          | | | | |
일 방 통 - 행        일 - 방 통 행
```

3. 편집 거리 알고리즘

다음에는 두 문자열의 *SED*를 구하는 점화식(recurrence relation)을 정의하도록 하겠다. 두 음절 간의 거리는 앞에서의 정의에 따르면 0, α, 2α, β ($\geq 3\alpha$) 중 하나의 값을 갖게 된다. 따라서 음절들을 영어의 문자로 생각하면, 대응되는 알파벳의 종류에 따라 서로 다른 비용을 부여하는 편집 거리 문제와 유사한 문제가 된다. 우리는 *SED*를 동적 프로그래밍 기법을 이용하여 구하였으며, 그에 해당하는 점화식을 아래와 같이 정의하였다. 아래 식에서 두 문자열 A에 대해서 $A(i)$는 길이가 i인 A의 접두사(prefix)를 뜻한다.

한글 편집 거리 알고리즘

Base Case

$SED(\Lambda,B(j))=j\times\beta$ for $|B|\geq j\geq 0$

$SED(A(i),\Lambda)=i\times\beta$ for $|A|\geq i\geq 0$

General Case

$$SED(A(i),B(j)) = \min(SED(A(i),B(j-1))+\beta,\ SED(A(i-1),B(j))+\beta,\ SED(A(i-1),B(j-1))+\delta_S(A[i],B[j]))$$

예제 4. "일반통계학"과 "일방통행"간의 *SED*를 위의 점화식을 이용하여 구해보자. 간단한 예로 $SED(A(1),B(1))$를 구하면, $SED(A(1),B(1))=\min(\beta+\beta,\beta+\beta,SED(0,0)+\delta_S(\text{"일"},\text{"일"})=\min(2\beta,2\beta,0)=0$이 된다. 모든 부분 문자열들에 대한 *SED*를 구하면 아래와 같이 되며, 두 문자열의 *SED*는 $3\alpha+\beta$가 된다. □

	Λ	일	방	통	행
Λ	0	β	2β	3β	4β
일	β	0	β	2β	3β
반	2β	β	α	$\alpha+\beta$	$\alpha+2\beta$
통	3β	2β	$\alpha+\beta$	α	$\alpha+\beta$
계	4β	3β	$\alpha+2\beta$	$\alpha+\beta$	$\alpha+\beta$
학	5β	4β	$\alpha+3\beta$	$\alpha+2\beta$	$3\alpha+\beta$

위의 표를 이용하여 배치를 구하면 다음과 같은 결과를 얻을 수 있다. 즉 음절-삭제 1회와 음소 1개, 2개가 바뀌는 음절-치환이 각각 1회씩 발생하였음을 알 수 있다.

```
일 반 통 계 학
| | | | |
일 방 통 - 행
```

3.1 편집 거리 알고리즘 증명

다음에는 위의 점화식이 우리가 정의한 *SED*값을 올바르게 구할 수 있음을 증명하도록 하겠다. 여기서는 동적 프로그래밍 증명 방법을 따르며, $i+j$에 대한 귀납법을 사용하였다. $i+j=0$ 즉, $i=j=0$일 경우는 $SED(A(0),B(0))=SED(\Lambda,\Lambda)=0$이 된다. Base case는 두 문자열 중 하나가 공백 문자열인 경우로, 공백의 모든 위치에서 삭제($B=\Lambda$) 또는 삽입($A=\Lambda$)이 발생하므로 base case에 해당하는 값을 갖게 된다.

General case에 대한 증명을 위해 *SEO*의 서열 $\sigma=\sigma_1,\sigma_2,\cdots,\sigma_n$를 $A(i)$를 $B(j)$로 변환하는 최소값을 갖는 배치라고 하고, σ_n을 $X\rightarrow Y$에 해당하는 *SEO*라고 하자. Property 2에 의하여 σ_n과 $A[i]$, $B[j]$는 반드시 아래의 4가지 중 하나의 관계를 갖는다.

- σ_n이 음절-삽입이고 $Y=B[j]$인 경우: 음절 $B[j]$가 삽입된 경우이다. 따라서 $\sigma_1,\sigma_2,\cdots,\sigma_{n-1}$은 $A[i]$, $B[j-1]$의 배치가 된다. 이 경우는 점화식의 첫 번째 조건에 해당한다.
- σ_n이 음절-삭제이고 $X=A[i]$인 경우: $A[i]$에 대해서 음절-삭제가 발생한 경우이다. 따라서 $\sigma_1,\sigma_2,\cdots,\sigma_{n-1}$은 $A[i-1]$, $B[j]$의 배치가 된다. 이 경우는 점화식의 두 번째 조건에 해당한다.
- σ_n이 음절-치환이고 $X=A[i]$, $Y=B[j]$인 경우: σ_n이 $A[i]\rightarrow B[j]$가 되는 치환인 경우로, $\sigma_1,\sigma_2,\cdots,\sigma_{n-1}$이 $A[i-1]$, $B[j-1]$의 배치를 나타내게 된다. 점화식의 세 번째 조건에 해당한다.
- 위의 3가지 경우에 포함되지 않는 경우: σ_n이 $A[i]$, $B[j]$와 관계된 *SEO*가 아닌 경우이며, $A[i]=B[j]$이 성립된다. 따라서 $\delta_S(A[i],B[j])=0$이 되며, 점화식의 세 번째 조건에 해당한다.

이상으로부터 우리가 제시한 점화식이 σ_n과 $A[i]$, $B[j]$ 사이에 가능한 4가지 경우를 모두 포함하므로, 우리의 알고리즘이 올바른 *SED*값을 구할 수 있음을 알 수 있다.

4. 구현 및 실험 결과

4.1 한글 인코딩 방법

한글을 컴퓨터에 저장할 때에는 하나의 음절을 2바이트로 인코딩하여 저장하게 된다. 인코딩 방법은 여러 가

지가 있을 수 있으나, 여기서는 유니코드를 기준으로 하겠다. 한글 유니코드는 완성형 한글을 표현하는 방법으로 초성, 중성, 종성을 각각 아래 표에 해당하는 숫자로 변환한 후, "초성번호"×21×28 + "중성번호"×28 + "종성번호" + 44032의 수식을 이용하여 값을 구하게 된다. 단 초성, 중성, 종성 1개로 이루어진 음절은 특정 유니코드 영역(0x3130 - 0x3082)의 값으로 표현한다. 이러한 방법으로 완성형 한글의 모든 음절을 2바이트 유니코드 값으로 변경할 수 있으며, 반대로 유니코드 값에 해당하는 한글 음절을 구하는 작업도 어렵지 않게 수행할 수 있다. 먼저 code 값이 "0×3130 - 0×3082" 영역에 속하는 값이면 그로부터 바로 하나의 음소로 이루어진 음절을 찾을 수 있으며, 속하지 않는 값이면 2개 이상의 음소로 이루어진 음절이므로 위의 수식으로부터 초성, 중성, 종성에 해당하는 번호를 계산하여 음절을 찾으면 된다.

• 초성

ㄱ	ㄲ	ㄴ	ㄷ	ㄸ	ㄹ	ㅁ	ㅂ	ㅃ	ㅅ	ㅆ
0	1	2	3	4	5	6	7	8	9	10
ㅇ	ㅈ	ㅉ	ㅊ	ㅋ	ㅌ	ㅍ	ㅎ			
11	12	13	14	15	16	17	18			

• 중성

ㅏ	ㅐ	ㅑ	ㅒ	ㅓ	ㅔ	ㅕ	ㅖ	ㅗ	ㅘ	ㅙ
0	1	2	3	4	5	6	7	8	9	10
ㅚ	ㅛ	ㅜ	ㅝ	ㅞ	ㅟ	ㅠ	ㅡ	ㅢ	ㅣ	
11	12	13	14	15	16	17	18	19	20	

• 종성

ㄱ	ㄲ	ㄳ	ㄴ	ㄵ	ㄶ	ㄷ	ㄹ	ㄺ	ㄻ
0	1	2	3	4	5	6	7	8	9	10
ㄼ	ㄽ	ㄾ	ㄿ	ㅀ	ㅁ	ㅂ	ㅄ	ㅅ	ㅆ	ㅇ
11	12	13	14	15	16	17	18	19	20	21
ㅈ	ㅊ	ㅋ	ㅌ	ㅍ	ㅎ					
22	23	24	25	26	27					

예제 5. "가"에 대한 유니코드 값을 구하면 초성 "ㄱ", 중성 "ㅏ", 종성 "공백"이 모두 0이므로 "가"의 유니코드 값은 44032 = 0xAC00이 된다. 비슷한 방법으로 "울"에 대한 값을 구하면 초성, 중성, 종성의 값이 각각 11, 13, 8이 되므로 유니코드 값은 50872 = 0x6CB8이 된다. □

4.2 실험 결과

한글의 편집 거리를 구하기 위해서는 알고리즘 구현에 필요한 메모리와 유니코드 값으로부터 한글을 추출하기 위한 메모리가 필요하게 된다. 그러나 유니코드를 사용하는 경우는 위의 표로부터 쉽게 음절을 찾을 수 있으므로 컴퓨터로부터 음절을 구분하는 데에는 상수 크기의 메모리만을 추가로 필요로 하게 된다. 그 외에 알고리즘 구현은 동적 프로그래밍을 사용하기 때문에 두 문자열 A, B에 대하여 $O(|A||B|)$의 시간 안에 $O(|A||B|)$의 메모리를 사용하여 편집 거리를 구할 수 있다. 또한 동적 프로그래밍 영역에서 잘 알려진 방법을 사용하면 $O(\min(|A|,|B|))$에 해당하는 선형 크기의 메모리를 사용하여 $O(|A||B|)$ 시간 안에 편집 거리를 구할 수 있다[5].

다음에는 실생활에서 헷갈리기 쉬운 몇몇 유사 단어들에 대한 거리를 구해보았다. 아래 표에서 볼 수 있듯이 모든 예제에서 거리가 $2\alpha+\beta$이하로 나타났으며, 이로부터 사람들이 쉽게 헷갈리는 단어들은 많은 경우 2개 이하의 음절에서 서로 다른 값이 나타나는 것을 확인할 수 있었다.

A	B	$SED(A,B)$
종묘제례악	종묘제례악	α
교양학단	교향악단	2α
아지랭이	아지랑이	α
반드시	반듯이	2α
알레르기	알러지	$2\alpha+\beta$
비타민	바이타민	$\alpha+\beta$
고양이	괭이	$2\alpha+\beta$

다음에는 우리가 제안한 알고리즘을 구현하여 앞에서 언급한 *SylED*, *PhoED*와 비교하는 실험을 진행하였다. 표현의 편의를 위하여 우리가 제안한 방법을 *KorED*라고 부르도록 하겠다. 실험 데이터로는 문화관광부의 문서로부터 얻은 헷갈리기 쉬운 한글 단어에 대한 데이터를 사용하였다[6]. 이 데이터는 (가까이,가까히), (강낭콩,강남콩), (덥다,더웁다)와 같이 일상생활에서 쉽게 헷갈릴 수 있는 689쌍의 단어쌍을 포함하고 있다. 이 데이터를 이용하여 다음과 같은 두 가지 종류의 단어쌍의 집합을 구성하였다.

X: 유사단어의 집합, 위에서 찾은 689개의 단어쌍으로 이루어진 집합. 앞에서 언급한 (가까이,가까히), (강낭콩,강남콩), (덥다,더웁다)등이 X에 포함된다.

Y: 비유사단어의 집합, 위에서 찾은 689개의 단어쌍 중에서 문법적으로 올바른 단어(위의 표에서의 A에 해당하는 단어)들에 대한 모든 조합(약 23만여개)으로 구성된 집합. (가까이,강낭콩), (강낭콩,덮다)와 같은 데이터가 Y에 해당된다.

X와 Y의 정의에 의하여 X에 속한 단어쌍에 대해서는 유사하다고 판단하고, Y에 속한 단어쌍에 대해서는 유사하지 않다고 판단하는 비율이 높을수록 좋은 알고리즘이라고 할 수 있다. X에 속한 단어쌍을 유사하

다고 판단할 확률을 P_X, Y에 속한 단어쌍을 잘못 판단하여 유사하다고 판단할 확률을 P_Y으로 표기하도록 하자. 세 가지 알고리즘에서 사용한 파라미터는 *SylED*의 경우에는 음절이 다른 경우에 대해서 3을 할당하고, *PhoED*는 서로 다른 음소의 개수에 1의 값을 부여하였으며, *KorED*에 대해서는 $\alpha=1$, $\beta=3$을 사용하였다.

다음의 표 1은 세 알고리즘의 실험 결과를 보여준다. 제일 왼쪽열의 값은 각각의 알고리즘이 두 문자열을 유사하다고 판단하는 기준 값을 나타낸다. 먼저 *SylED*의 결과값을 살펴보자. 기준 비용이 3인 경우, 즉 하나의 음절이 다른 경우를 보면, 유사한 단어를 올바르게 판단할 확률은 75.9%이고, 유사하지 않은 단어를 유사하다고 잘못 판단할 확률은 0.3%이다. 기준값을 6으로 했을 경우에는 각각 96.2%, 6.9%가 된다.

다음에는 실험 결과로부터 세 알고리즘을 비교하도록 하겠다. 먼저 *SylED*와 *PhoED*의 실험 결과를 살펴보자. *SylED*에서 비용이 6인 경우와 *PhoED*에서 비용이 5인 경우를 살펴보자. 둘 다 유사한 단어를 정상적으로 판단할 확률은 96.2%이지만, 유사하지 않은 단어를 유사하다고 잘못 판단할 확률은 각각 6.9%, 5.2%로 나타났다. 또한 유사단어를 99.3%확률로 구분하는 기준 비용이 9, 7의 경우에는, 유사하지 않은 단어를 잘못 판단할 확률이 각각 45.2%, 25.2%가 되었다. 이는 *PhoED*가 *SylED*에 비하여 더 좋은 결과를 보여준다.

표 1 실험 결과

비용	*SylED*		*PhoED*		*KorED*	
	P_X	P_Y	P_X	P_Y	P_X	P_Y
0	0	0	0	0	0	0
1	-	-	68.4	0	68.4	0
2	-	-	83.5	0.1	82.6	0
3	75.9	0.3	89.6	0.3	89.1	0.3
4	-	-	94.2	1.5	93.8	1.1
5	-	-	96.2	5.2	95.8	3.7
6	96.2	6.9	98.4	13.0	98.0	10.0
7	-	-	99.3	25.2	95.5	18.6
8	-	-	99.7	43.1	99.6	32.5
9	99.3	45.2	99.9	58.9	99.9	52.0
10	-	-	100.0	72.5	100.0	63.0
11	-	-	100.0	83.7	100.0	75.6
12	100.0	83.3	100.0	89.7	100.0	86.8
13	-	-	100.0	93.6	100.0	91.0
14	-	-	100.0	96.3	100.0	94.4
15	100.0	95.3	100.0	97.9	100.0	96.7
16	-	-	100.0	99.2	100.0	98.2
17	-	-	100.0	99.8	100.0	99.4
18	100.0	100.0	100.0	99.9	100.0	99.9
19	-	-	100.0	100.0	100.0	100.0

이제 *PhoED*와 본 논문에서 제안한 *KorED*를 비교해보도록 하겠다. 기준 비용이 5인 경우를 보면 유사한 단어를 제대로 판단할 확률은 각각 96.2%, 95.8%로 거의 비슷하지만 유사하지 않은 단어를 잘못 판단할 경우는 각각 5.2%, 3.7%가 되어 *KorED*가 더 나은 결과를 보여주고 있다. 또한 기준 비용이 9인 경우를 보면 두 알고리즘 모두 유사 단어는 99.9%의 확률로 구분하지만, 잘못 판단할 확률은 *KorED*가 *PhoED*보다 약 6.9%정도 낮은 결과를 보여주고 있다.

이상의 결과로부터 *KorED*가 *SylED*, *PhoED*보다 더 좋은 결과를 보여줌을 확인할 수 있었다. 추가적으로 *KorED*에 대해서 α를 1로 고정하고 β를 4, 5, ... 로 증가시키는 실험을 진행하였으나, $\beta=3$일 때에 가장 나은 결과를 얻을 수 있었다. 또한 *KorED*의 실험 결과에서 기준 값이 5에서 6으로 올라갈 때, 잘못 판단하는 확률이 3.7%에서 10.0%로 급증하는 현상을 확인할 수 있다. 이상의 실험결과로 볼 때, 변수를 $\alpha=1$, $\beta=3$로 설정하고 유사한 단어를 판단하는 기준값을 5로 할 때, 가장 적절한 결과를 얻을 수 있다고 생각할 수 있다.

참 고 문 헌

[1] Gusfield, D.: Algorithms on strings, trees, and sequences : computer science and computational biology, Cambridge Univ. Press, January 2007.

[2] Wagner, R. A., Fischer, M. J.: The String-to-String Correction Problem, *J. ACM*, 21(1), pp.168-173, 1974.

[3] Navarro, G.: A guided tour to approximate string matching, *ACM Computing Surveys*, 33(1), pp.31-88, 2001.

[4] Gong, R., Chan, T. K.: Syllable Alignment: A Novel Model for Phonetic String Search, IEICE - Trans. Inf. Syst., E89-D(1), pp.332-339, 2006.

[5] Hirschberg, D. S.: A linear space algorithm for computing maximal common subsequences, Commun. *ACM*, 18(6), pp.341-343, 1975.

[6] Sowon Chang, Seong-kyu Kim, Seung-chul Jung, This slip of the tongue that slip of the pen: official documents, Ministry of Culture and Tourism, 2000 (in korean).

노 강 호

2001년 서울대학교 컴퓨터공학부 학사 2003년~현재 서울대학교 전기·컴퓨터공학부 박사과정. 2007년~현재 삼성전자 DS부문 책임연구원. 관심분야는 알고리즘, Flash Memory Software

김 진 욱
정보과학회논문지 : 시스템 및 이론
제 37 권 제 1 호 참조

김 은 상
2005년 서울대학교 컴퓨터공학부 학사
2005년~현재 서울대학교 전기·컴퓨터 공학부 박사과정. 관심분야는 컴퓨터이론, 알고리즘, 웹검색, 암호 및 보안

박 근 수
정보과학회논문지 : 시스템 및 이론
제 37 권 제 1 호 참조

조 환 규
1984년 서울대 계산통계학과 학사. 1986, 1990년 KAIST 전산학과 석사, 박사. 1990년~현재 부산대학교 정보컴퓨터공학과. 관심분야는 알고리즘 설계과 분석, 그래프이론, 생물정보학