복합키워드의 고속검색 알고리즘에 관한 연구

이진관* · 정규철* · 이태헌* · 박기홍*

A Study of High Speed Retrieval Algorithm of Long Component Keyword

Jin-Kwan Lee* · Kyu-cheol Jung* · Tae-hun Lee* · Ki-hong Park*

이 논문은 군산대학교 교내 연구비를 지원받았음.

요 약

효율적인 키워드 추출은 정보검색 시스템에서 중요하지만 많은 키워드 중 적당한 키워드를 결정하기 위한 방법들은 여러 가지가 있다. 그중 단일 키워드만을 검색하는 AC알고리즘을 해결하기 위한 DER구조는 복합키워드 검색이 가능하나 많은 검색시간이 걸린다는 문제점을 가지고 있다. 본 논문에서는 이러한 문제점을 해결하기 위해 DER구조의 검색방법을 기반으로 한 독립적인 검색테이블을 확장하여 EDER 구조라는 알고리즘을 구축하였다. 500개의 텍스트 파일을 실험한 결과 키워드의 포스팅 결과가 AC의 DER구조보다 EDER구조가 작았으며, 검색시간 또한 K5에서 DER구조가 0.6초, EDER구조가 0.2초로 더 빠른 검색을 보여주고 있어 제안 방법이 효과적임을 알 수 있었다.

ABSTRACT

Effective keyword extraction is important in the information search system and there are several ways to select proper keyword in many keywords. Among them, DER Structure for AC Algorithm to search single keyword, can search multiple keywords but it has time complexity problem.

In this paper, we developed a algorithm, "EDER structure" by expanding standalone search table based on DER structure search method to improve time complexity. We tested the algorithm using 500 text files and found that EDER structure is more efficient than DER structure for AC for keyword posting result and time complexity that 0.2 second for EDER and 0.6 second for DER structure,

키워드

키워드검색기법, AC알고리즘, DER구조, EDER구조

Ⅰ. 서 론

효율적인 키워드 추출은 문서 관리 시스템, 인터넷 정보검색에서 중요한 일이므로 많은 연구들이 이루어지고 있고, 여러 분야에 적용되고 있다.

키워드 추출 시스템에서 많은 복합어들은 자유롭게 키워드 후보들을 만들어낸다. 이러한 키워드 추출에서 키워드 구조는 두 가지 형태를 가진다. 하나는 단일 키워드(Single Component keyword)이고, 다른 하나는 복합 키워드(Long Component keyword)이다[1].

단일 키워드 처리에서 짧은 길이의 복합어는 우

*군산대학교

접수일자 : 2004. 7. 13

선순위가 주어지고 추출 매칭에 의해 키워드 검색이 쉽다. 그러나 단일 키워드는 복합 키워드에 의해 본래의 의미가 모호해진다. 예를 들면, 단일 키워드 "식물"은 복합 키워드 "동식물"의 의미를 표현 할 수 없다. 이에 비해 복합 키워드 "동식물"는 단일 키워드 "식물"을 원소로 포함하지만, 단일 키워드 검색시 복합 키워드를 추출 매칭에서 검색할 수 없다[2. 3].

또한, AC(Aho-Corasik) 알고리즘 키워드 검색은 키워드 "식물"을 검색했을 때, "식물"만을 검색한다. 그리고 키워드 "동식물"를 검색했을 때에만 "동식물"와 "식물"의 검색이 가능하다. 즉, AC 알고리즘에서는 "식물"을 가지고, "동식물"를 검색할 수 없다[4].

AC 알고리즘을 확장한 DER구조에서는 "식물"을 검색할 때, "동식물"와 "식물"의 검색이 가능하다. 즉 단일 키워드를 검색할 때, 복합 키워드까지 검색이 가능하다[3]. 그러나 이러한 검색을 위해서는 역(reverse)정보를 이용하여, 각 상태들을 추적하면서, 검색을 해야 하므로 많은 검색시간이 걸리는 문제점이 있다.

본 논문에서는 위와 같은 문제점을 없애기 위해 확장된 DER(Extended DER : EDER) 구조를 제안한다. 본 논문의 2장에서는 AC 알고리즘과 DER 구조에 대해 설명한다. 3장은 EDER 구조에 대해 설명한다. 4장에서는 실험 및 평가를 한다. 그리고 마지막으로 5장에서 결론을 맺는다.

II. AC 알고리즘과 DER구조

2.1 AC 알고리즘

AC 알고리즘은 단일 경로에서 문자열에 포함된 모든 키워드의 위치를 결정할 수 있는 알고리즘이다[4. 5. 6].

입력 심벌의 집합 I에서 AC 알고리즘의 동작을 다음 3개의 함수로 정의할 수 가 있다.

goto function $g : S \times I \rightarrow S \cup \{fail\}$,
failure function $f : S \rightarrow S$,
output function *Output* : $S \rightarrow A$, *K_SET*의 부분집합.

K_SET은 AC 알고리즘에 의해 검색될 수 있는 키워드의 전체집합이다. S는 상태들의 유한 집합이고, 출력함수에 의해 검색된 키워드의 집합을 A라 한다. AC 알고리즘은 상태들의 집합으로 구성된다. 각 상태는 숫자에 의해 표현된다. goto 함수 g는 상태 si에서 입력심벌 i를 보고, 다음 상태 sj로 이동하는 함수이다. 다음 상태로 가는 연결선이 없을 때, fail이 발생한다. fail이 발생하면 failure 함수 f가 사용된다. failure 함수 f는 상태 si에서 다음 상태 sj로 이동하는 함수이다. 출력함수 Output은 K_SET의 부분집합 Ai를 출력한다.

AC 알고리즘 상에서의 키 검색은 그림1과 같이 출력함수 Output에서 검색 가능한 키워드들을 보여준다. Output(3)={국립}이라는 것은 상태3에서 검색이 가능한 키워드가 "국립"이라는 것을 의미한다.

```
Output(3)={ 국립 }
Output(6)={ 국립동식물, 동식물, 국립, 식물 }
Output(7)={ 국립동식물원, 국립동식물, 동식물원, 동식물,
    국립, 식물, 원 }
Output(10)={ 동식물, 식물 }
Output(11)={ 동식물원, 동식물, 식물원, 식물, 원 }
Output(13)={ 식물 }
Output(14)={ 식물원, 식물, 원 }
Output(15)={ 식물학, 식물, 학 }
Output(19)={ 관엽식물, 식물 }
Output(20)={ 원 }
Output(21)={ 학 }
```

그림 1. 키워드집합 K_SET에 대한 AC 알고리즘
Fig 1. AC algorithm for keyword set K_SET

그림1의 AC알고리즘 키워드 검색은 키워드 "식물"을 검색했을 때, "식물"만을 검색한다. 그리고 키워드 "동식물"를 검색했을 때에만 "동식물"와 "식물"의 검색이 가능하다. 즉, AC 알고리즘에서는 "식물"을 가지고, "동식물"를 검색할 수 없다.

2.2 DER 구조

DER 구조의 경우, 키에 대한 레코드는 파일번호나 문장번호 등과 같은 포스팅(posting)이라고 불리는 참조 정보를 가지고 있다. g(1,x)=s 와 Output(s)∋x와 같은 키 x에 대한 포스팅을 P(s)라 한다. failure 함수에 의한 반대 방향으로의 상태 전이를 표현하기 위해 아래 집합을 소개한다[5]. 아래의 집합은 역(reverse)함수 f-1(s)에 의해 구해진다[3].

$$f^{-1}(s)=\{s' \mid f(s')=s\}$$

그림2는 DER 구조에서 사용되어진 goto 함수, failure 함수, reverse 함수에 의해 구성된 상태들의 관계를 보여준다. 상태1에서 상태2로 goto 함수에 의해 상태 전이가 가능하고, failure 함수에 의해 상태4에서 상태8로 상태 전이가 가능하다. 또한 상

태8에서 상태4로 reverse 함수에 의해 상태 전이가 가능하다.

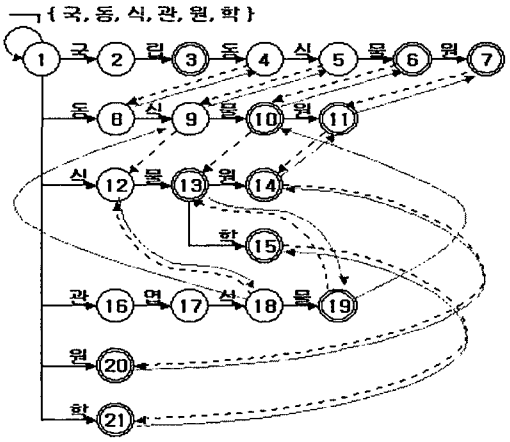

그림 2. DER 구조의 goto, failure와 reverse 함수
Fig 2. Goto and failure, reverse function of DER structure

그림3은 향상된 출력함수인 Output1에 그림1에 적용 시켜 나온 Output1 함수의 예를 나타낸 것이다.

```
Output1(3)={P(3)}
Output1(6)={P(3), P(6), P(10), P(13)}
Output1(7)={P(3), P(6), P(7), P(10), P(11),
            P(13), P(20)}
Output1(10)={P(10), P(13)}
Output1(11)={P(10), P(11), P(13), P(14), P(20)}
Output1(13)={P(13)}
Output1(14)={P(13), P(14), P(20)}
Output1(15)={P(13), P(15), P(21)}
Output1(19)={P(13), P(19)}
Output1(20)={P(20)}
Output1(21)={P(21)}
```

그림 3. 향상된 출력함수
Fig 3. Improved output function

그림1과 그림3에서 x="국립동식물"에서 Output1(6)= {P(3), P(6), P(10), P(13)}은 g(1,x)=6과 같은 상태6에서 P(3), P(6), P(10), P(13)이 검색 될 수 있다. x="식물"에 대해 Output1(13) ={P(13)}은 g(1,x)=13과 같은 상태13에서 P(13)이 검색될 수 있다. 그러나 Output1(13)에서는 "식물원", "동식물원", "동식물"과 같은 단일 키워드에 대한 포스팅을 검색하기는 불가능하다.

DER구조의 검색 기법은 y와 z를 공백이 아닌 문자열이라고 할 때, 아래 4개의 검색 방법을 정의한다[3].

EXACT(s)={ $P(s)$ | $g(1,x)=s$ 이고 $x \in Output(s)$ }
PREFIX(s)={ $P(t)$ | $g(1,x)=s$ 이고 $x \in Output(s)$을 만족하는 각 s에 대해, $g(s, y)=t$ 이고 $xy \in Output(t)$을 만족하는 상태들 }
SUFFIX(s)={ $P(t)$| $g(1,x)$=s 이고 $x \in Output(s)$인 각 s에 대해, 만약 상태t가 f-1(s)에 포함되고, $Output(t) \neq empty$이면, $Output(t)$는 $g(1,yx)=t$를 만족하는 yx를 갖는다. }
PROPER(s)={ $P(r)$ | $g(1,x)=s$ 이고 $x \in Output(s)$을 만족하는 각 s에 대해, 만약 $f^{-1}(s)$가 $g(t,y)=r$을 만족하는 상태t를 갖는다면, $P(r) \in Output(r)$이다. }

DER(Delayed Extraction and Retrieval) 구조에서는 "식물"을 검색할 때, "동식물"와 "식물"의 검색이 가능하다. 즉 단일 키워드를 검색할 때, 복합키워드까지 검색이 가능하다. 그러나 이러한 검색을 위해서는 reverse 정보를 이용하여, 각 상태들을 추적하면서, 검색을 해야 하므로 많은 검색시간이 걸리는 문제점이 있다.

III. EDER 구조의 제안

3.1 EDER 구조

기존 DER 구조의 검색 방법은 failure 정보에 대한 reverse 정보를 이용하여 각 상태를 추적하면서 검색하므로 많은 시간이 걸리는 문제가 있다. 본 논문에서 제안한 EDER 구조는 DER의 검색방법을 이용하여 테이블 구조를 만들고, 검색 시에는 테이블을 이용하여 빠른 검색이 가능하다. 그러므로 DER 구조의 검색시간이 많이 걸리는 문제를 해결할 수 있다.

예를들어 "식물"을 그림2의 DER구조에서 검색할 때를 생각해보자. 먼저 EXACT 검색에서 "식물"을 검색하는 것은 DER와 EDER이 같다. 다음 PREFIX 검색에서 goto함수에 의해 "식물원"과 "식물학"을 찾을 수 있다. 그런데 "식물" 다음에 오는 글자가 하나 이상의 글자가 오는 키워드가 있다면 즉 "식물학과"가 있다면 "식물"까지 추적한 후 goto함수로 "학"까지 가고 다음 또 goto 함수로 "과"를 추적한다. 그리고 각 상태에서 output 함수로 키워드를 추출한다. 그러나 EDER은 EXACT 검색에서 키워드가 결정되면 그 키를 인덱스테이블

에서 인덱스를 찾고 이 인덱스를 PREFIX 검색 테이블의 인덱스로 사용하여 START와 OFFSET을 구한다. START는 다시 ARY테이블에서 인덱스로 사용되고 그 인덱스로부터 OFFSET수 만큼의 키워드를 추출할 수 있다. SUFFIX와 PROPER- SUB 검색 방법도 같은 방법으로 각 SUFFIX테이블과 PROPER-SUB테이블에서 검색할 수 있다. 그러므로 복합 키워드가 단일 키워드보다 단어의 수가 길수록 EDER구조가 빠른 검색을 할 수 있다는 것을 알 수 있다.

3.2 EDER 구조의 구성

DER 구조의 검색기법을 이용하여 검색될 수 있는 모든 키워드를 테이블을 이용하여 정렬하므로써 테이블을 이용한 빠른 검색이 가능하도록 하였다. 이 같이 하여 만들어진 테이블은 PREFIX 테이블, SUFFIX 테이블, PROPER 테이블이다. EXACT 테이블은 PREFIX 테이블에 포함되므로 EXACT 테이블은 만들지 않고, 검색시 EXACT 인덱스는 PREFIX 인덱스 테이블을 이용한다.

3.2.1 PREFIX 테이블의 구성

PREFIX 검색 테이블을 구성하는 부분으로, Prefix.IND과 Prefix.ARY, Prefix.START, Prefix.OFFSET 테이블을 생성한다.

```
Procedure PREFIX()
Step1     {Initialization}
          Index=0, prefix.IND=0, prefix.ARY=0;
          prefix.START=0, prefix.OFFSET=1;
Step2     if 단어가 키워드 아니면 goto Step4
          Index ++;
          if prefix[Index].START=0 then
          prefix[Index].START= Index; endif
          prefix[Index].IND=현재 문자의 위치;
          prefix[Index].ARY=현재 문자의 위치;
Step3     if 다음 문자가 없으면 goto Step4
          if 다음 문자까지가 키워드 아니면
             goto Step3;
          profix[Index].OFFSET ++;
          다음 문자를 현재 문자로 놓는다.
          goto Step3
Step4     if 다음 단어가 없으면 Step5
          다음 단어를 현재 단어로 놓는다.
          goto Step2
Step5     Return
```

3.2.2 SUFFIX 테이블 구성

현재 위치의 reverse 정보를 이용하여 SUF FIX 검색 테이블의 인덱스 테이블(IND)을 생성한다.

```
Procedure SUFFIX()
Step1     {Initialization}
          Index=0,iIndex=1, suffix.IND=0,
          suffix.ARY=0;
          suffix.START=0, suffix.OFFSET=0,nIndex=0;
Step2     if 단어의 reverse가 없으면 goto Step7
          if suffix[all].IND에 단어위치정보가
          존재하면 goto Step7
Step3     if 현재 문자의 failure 위치정보가
          없으면 Step 4
          push(현재 문자 위치);
          failure 위치정보를 현재 문자 위치로
          놓는다.
          goto Step3
Step4     Index ++;
          suffix[Index].IND=현재 문자의 위치;
          if nIndex == 0 then nIndex = Index;
Step5     if 현재 문자의 reverse가 없으면 then
          nIndex = 0; goto Step6; endif
          reverse의 위치를 현재위치로 놓는다.
          if suffix[Index].START==0 then
          suffix[Index].START= nIndex; endif
          suffix[Index].OFFSET ++;
          if nIndex == 0 then
          suffix[iIndex].ARY=현재위치;
          iIndex ++; end if
          goto Step5;
Step6     if 스택이 비었으면 goto Step7
          현재 문자 위치 = pop();
          if suffix[nIndex].IND와
          현재 문자 위치가 같은 것이 있으면 then
          nIndex ++;
          else nIndex = 0; endif
          goto Step4;
Step7     if 다음 단어가 없으면 Step8
          다음 단어를 현재 단어로 놓는다.
          goto Step2
Step8     Return
```

Step3에서는 스택에 검색된 위치들을 저장한다. 왜냐하면, 가장 늦게 검색된 키워드의 위치가 앞으로 와야 하기 때문에 스택을 이용하여 위치를 바꾸어 준다. Step4,5,6는 스택의 위치 값을 순서대로 가져와 S-IND, S-ARY, S-OFF SET 테이블을 생성

하는 부분이다.

3.2.3 PROPER-SUB 테이블 구성

PS-IND, PS-ARY, PS-START, PS-OFF SET 테이블을 구성하는 부분이다.

```
Procedure PROPERSUB()
Step1   {Initialization}
        Index=0,iIndex=1, proper.IND=0,
        proper.ARY=0; proper.START=0,
        proper.OFFSET =0, nIndex=0;
Step2   if 단어의 reverse가 없으면 goto Step9
        if 현재 단어의 뒤로 붙는 문자가
          없으면 Step9
        if proper[all].IND에 단어의 위치정보가
          존재하면 Step9
Step3   if 현재 문자의 failure 위치정보가
          없으면 Step4
        push(현재단어위치);
        failure 위치정보를 현재 문자 위치로
        놓는다.
        goto Step3
Step4   Index ++;
        proper[Index].IND=현재 문자의 위치;
        if nIndex==0 then nIndex = Index;
Step5   reverse의 위치정보를 현재 위치로
        놓는다.
        if proper[Index].START == 0 then
        proper[Index].START = nIndex; endif
Step6   if 다음 문자가 없으면 goto Step8
        if 다음 문자까지가 키워드 아니면 Step7
        proper[Index].OFFSET ++;
        if nIndex==0 then
        proper[iIndex].ARY=현재위치;
        iIndex ++; endif
Step7   다음 문자위치를 현재문자위치로 놓는다.
        goto Step6
Step8   if 스택이 비었으면 goto Step9
        현재 문자 위치 = pop();
        if proper[nIndex].IND와 현재 문자위치가
        같은 nIndex값이 있으면 then nIndex ++;
        else nIndex = 0;
        goto Step4
Step9   if 다음 단어가 없으면 goto Step10
        다음 단어를 현재 단어로 놓는다.
        goto Step2
Step10  return
```

Step2는 키워드의 앞뒤로 확장되는 부분을 검색하고, failure 위치 정보를 갖을 때는 스택을 이용하여 가장 나중에 검색된 키워드을 인덱스의 처음에 위치시킨다. 이 스택의 값을 이용하여 PS-IND 테이블을 구성한다. Step5에서 PS-START 테이블을 구성하고 Step6에서 PS-ARY와 PS-OFFSET 테이블을 구성한다.

3.3 EDER 구조에서의 키워드 검색

그림4(a)는 검색 테이블의 인덱스(IND)를 나타낸다. (b)는 PREFIX 함수에 의해 구해진 PREFIX 검색 테이블이고. (c)는 SUFFIX 함수에 의해 구해진 SUFFIX 검색 테이블이다. (d)는 PROPER 함수에 의해 구해진 PROPER- SUB 검색 테이블이다.

이 방법의 중점은 검색 결과의 포괄적인 관계이다. 그림4로부터 EXACT와 PREFIX의 검색 결과는 EXACT(6)∪PREFIX(6)=P(6), EX ACT(7)∪PREFIX(7)=P(7)와 같은 포함관계를 가진다. 그러므로 EXACT함수의 검색 결과는 PREFIX 함수의 검색 결과에 포함된다는 것을 나타낸다. 또한 이 관계의 사용으로 P(s)는 1차원 배열 ARY로 겹쳐 놓는다.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
E-IND P-IND			1			2	3			4	5		6	7	8				9	10	11
S-IND										2	5		1	4						3	6
PS-IND										2			1								

(a) 검색 테이블의 인덱스
(a) Index of retrieval table

	1	2	3	4	5	6	7	8	9	10	11
P-ARY	P(3)	P(6)	P(7)	P(10	P(11	P(13	P(14	P(15	P(19	P(20	P(21

	1	2	3	4	5	6	7	8	9	10	11
P-START	1	2	3	4	5	6	7	8	9	10	11
P-OFFSET	3	2	1	2	1	3	1	1	1	1	1

(b) PREFIX 검색 테이블
(b) PREFIX retrieval table

	1	2	3	4	5	6	7
S-ARY	P(19)	P(10)	P(6)	P(14)	P(11)	P(7)	P(15)

	1	2	3	4	5	6
S-START	1	3	4	5	6	7
S-OFFSET	3	1	3	2	1	1

(c) SUFFIX 검색 테이블
(c) SUFFIX retrieval table

	1	2
PS-ARY	P(11)	P(7)

	1	2
PS-START	1	2
PS-OFFSET	2	1

(d) PROPER-SUB 검색 테이블
(d) PROPER-SUB retrieval table
그림 4. 검색 테이블
Fig 4. Retrieval tables

START는 ARY의 시작 인덱스를 나타내고, OFFSET은 검색 엔트리의 수이고, IND는 상태번호부터 START의 인덱스로 맵핑한다. 그리고 t는 상태번호를 나타낸다.

J=IND(t)에 대해 I=START(J)이고 K=OFF SET(J)라고 가정하면, 결과는 START(J)=I, OFFSET(J)=K이기때문에 ARY(I), ARY(I+1), … ,ARY(I+K-1)이 된다.

PREFIX, SUFFIX, EXACT, PROPER 테이블을 구분하기 위해 IND, ARY, START, OFF SET에 E-, P-, S-, PS-를 붙인다. EXACT 검색테이블는 E-IND 데이터만 나타나고, E- ARY 데이터는 P-ARY에 합병된다.

그림4는 그림2을 EDER구조로 구성한 예이다.

상태13을 가지고 PREFIX 검색 테이블에서 검색한다면, P-IND (13)=6으로부터 P-IND 테이블에서 상태 13에 의해 인덱스 6을 구할 수 있다. 인덱스 6은 P-START(6)=6과 P-OFF SET(6)=3을 구하고, P-START(6)=6에 의해 P-ARY(6)을 구하고, P-OFFSET(6)=3에 의해 P-ARY(6), P-ARY(7), P-ARY(8)과 같이 3개의 엔트리를 구할 수 있다. 즉 P-ARY(6)= P(13), P-ARY(7)=P(14), P-ARY(8)=P(15)가 검색되어 진다.

상태13을 가지고 SUFFIX 검색 테이블과 PROPER-SUB 검색 테이블에서 검색한다.

IV. 실험 및 평가

코퍼스에서 체언을 추출한 후, EDER 구조를 이용하여 키워드 사전을 구성하였다. 그리고 EDER 구조로 구성된 키워드 사전에 있는 키워드로 문서들에 대한 포스팅 정보를 구성하였다.

사용한 코퍼스는 KAIST에서 구성한 100만 어절의 품사 부착 코퍼스을 사용하여 구성하였다[7].

제안한 방법의 평가를 위해 CD 데이터로 제공된 한국정보처리학회 13회 춘계학술발표 논문집과 한국정보과학회 27회 추계 학술발표논문집 중 500개의 파일을 사용하여 포스팅한 결과, 1,722 복합 키워드가 생성되었다. 사용한 파일의 개수는 500개로 제공된 CD의 문서 600개에서 키워드사전과 관련된 주제의 문서로 선별하였다.

본 평가는 펜티엄 III 800MHz, 128MByte 메모리의 PC에서, 윈도우98, Visual C6.0에서 구현 및 실험을 하였다.

표 1. 실험 결과
Table 1. Simulation Result

		K1	K2	K3	K4	K5
전체키워드 수		350	1,249	1,849	2,747	3,347
복합 키워드 수		0	350	597	932	1,022
전체 포스팅 수	AC	6,128	8,451	9,917	11,383	12,186
	DER와 EDER	742	1,550	2,115	2,645	2,759
사전 크기 (Kbyte)	AC	23.4	32.2	37.8	43.4	46.5
	DER	31.2	42.9	50.4	57.8	62
	EDER	18.5	34.5	45.7	61.4	70.6
검색 시간 (sec)	DER	0.1	0.2	0.2	0.4	0.6
	EDER	0.1	0.1	0.1	0.2	0.2

표1은 복합 키워드의 수에 따른 다양한 시뮬레이션 결과를 보여준다.

검색시간의 비교평가는 AC와 DER에서 검색되는 키워드가 다르고 DER와 EDER에서는 같은 키워드를 검색한다. 그러므로 DER와 EDER의 검색시간 만을 평가한다.

K1, K2, K3, K4, K5는 EDER 구조를 구성하기 위한 키워드의 집합을 의미한다. 각 키워드의 집합은 100개씩의 문서로 구성되었다. 각 문서는 특별한 가중치는 없고 임의의 문서로 구성되었다.

표1에서 보는 바와 같이, K1에서는 키워드가 350개일 때, AC의 사이즈는 23.4Kbytes, DER 구조는 31.2Kbytes이고, EDER은 18.5Kbytes로 DER 구조가 사전의 크기에서는 더 늘어나는 것을 확인할 수 있다. K5에서는 키워드가 3,347개일 때, AC은 46.5Kbytes, DER구조는 62Kbytes, EDER은 70.6Kbytes로 구성된다. 검색시간은 K5에서 DER 구조가 0.6초, EDER구조가 0.2초로 나타났다.

이 결과로 DER 구조, EDER 구조가 AC보다 사전 크기에서는 증가하지만, 복합 키워드들을 검색할 수 있고, EDER 구조가 DER구조에 Table이 추가되지만, 검색시간에 총 키워드의 수가 증가할수록 빠른 검색 시간을 보여주고 있다는 것을 확인할 수 있다. 사전의 크기가 증가하는 이유는 사전의 구조가 변경되었기 때문이다. AC는 base, check와 failure의 세 배열로 이루어져있다.[4] DER는 base, check, failure배열에 reverse 배열이 추가된 구조

로 reverse 배열 크기만큼의 사전이 커지게 된다. EDER은 base와 check 배열과 검색테이블로 이루어지게 된다. failure와 reverse 배열은 검색 테이블을 만들 때만 필요하고, 검색시에는 필요 없는 부분이므로 base와 check 배열과 검색테이블로 재구성된다. EDER은 DER보다 사전의 크기가 반으로 줄지만 검색테이블이 추가되므로 비슷한 크기를 유지하는 것을 확인할 수 있다. 검색테이블의 크기는 키워드의 수와 관계가 있다. 키워드의 수가 늘어날수록 사전의 크기 증가하는 것을 확인할 수 있다.

그러므로 제안한 방법이 실험적 평가에서 컴퓨터 시뮬레이션 결과 사전의 크기는 키워드가 증가할수록 조금씩 증가하지만, 빠른 검색을 할 수 있으므로, 효과적임을 확인하였다. 또한 AC알고리즘에서는 "식물"이라는 단어를 검색할 경우 "식물" 키워드만을 검색하지만, DER 구조와 EDER 구조에서는 "식물"과 "동식물"처럼 복합 키워드도 검색이 가능하다.

V. 결 론

정보검색에서 사용하는 키워드는 의미정보가 내포된 체언을 주로 사용한다. 체언은 사물의 실체를 가리키는 말로서 문장에서 조사의 도움을 받아서 주어, 목적어, 보어의 구실을 하는 말이다. 모든 단어를 가지고 키워드 사전을 구현할 때는 사전의 크기가 커지고, 검색시간이 많이 걸리기 때문에 모든 품사의 단어를 키워드 사전으로 구성하는 것은 비효율적이다. 또한 수작업을 통해서 소규모로 구성하면 키워드 사전을 구성하는데 많은 시간이 걸리는 문제가 발생한다. 따라서 본 논문에서는 적은 인력으로 대용량의 키워드 사전을 구성하기 위해 이미 구축되어 있는 코퍼스를 이용한다. 사용한 코퍼스는 KAIST에서 구성한 100만 어절의 품사 부착 코퍼스로 구성하였다.

제안한 방법의 평가를 위해 코퍼스에서 추출한 체언들을 EDER 구조로 된 키워드 사전을 구성하고, 키워드 사전의 키워드를 이용하여 CD로 제공된 한국정보처리학회 13회 춘계학술발표 논문집과 한국정보과학회 27회 추계학술발표 논문집 중 500개의 파일을 포스팅한 결과, AC 구조에서는 "식물"이라는 단어를 검색할 경우 "식물"키워드 만을 검색하지만, DER 구조와 EDER 구조에서는 "식물"과 "동식물"처럼 LC키워드도 검색이 가능하였다. 또한 표1에서 보는 바와 같이 총 키워드의 수가 증가할수록 검색시간이 빠름을 보여주고 있고, 포스팅의 수에서도 AC보다 DER와 EDER 구조가 월등히 감소한 것을 확인할 수 있었다. 그러므로 제안한 방법은 컴퓨터 시뮬레이션 결과 효과적임을 확인하였다.

향후 연구과제로는 제안한 방법에서 나타난 사전구축시간이 많이 걸리는 문제점의 해결과 키워드별 문서분류 코드를 주어 자동문서 분류시스템을 구성하는 것을 목표로 하고 있다.

참고문헌

[1] Fuketa, M., Mizobuchi, S., Hayashi, Y. and Aoe, J. "A Fast Method of Determining Weighted Compound Keywords from Text Databases", Information Processing & Management, Vol.34, No.4, pp.431-442, 1998.

[2] Kimoto, H. "Automatic indexing and evaluation of keywords for Japanese newspapers", IEICE Trans. J74-D-I, No.8, pp.556-566, 1991.

[3] Makoto Okada, Kazuaki Ando, Kazuhro Morita, Jun-ichi Aoe, "An Efficient Determination of Keywords for Compound Words", Proceedings of 18th ICCPOL, Vol 1, pp317-320, March 1999.

[4] Kazuaki Ando, Toshiharu Kinoshita, Masami Shishibori, Jun-ichi Aoe, "An improvement of the Aho-Corasick machine", International Journal of Information Sciences, Vol 3, pp139-151, 1998.

[5] Kazuaki Ando, Masao Fuketa, Masami Shishibori, Jun-ichi Aoe, "Dictionary Structure for Morphological Analysis of Oriental Languages", Proceedings of 18th ICCPOL, Vol 1, pp533-538, March 1999.

[6] A. V. Aho, M. J. Corasick, "Efficient string matching: an aid to bibliographic search", Comm. ACM, Vol.18, No.6, pp.333-340, 1975.

[7] 정민수, "코퍼스로부터 구문분석을 위한 사전 구성", 군산대학교 컴퓨터과학과 석사학위논문, 1999.

자기소개

이진관(Jin-Kwan Lee)

1995 군산대학교 컴퓨터과학과 (학사)
1999 군산대학교 컴퓨터과학과 (석사)
2002~현재 군산대학교 컴퓨터과학과 박사과정
※관심분야 : 자연어처리, 정보검색, 음성인식, 유해 차단, 텔레메틱스

정규철(Kyu-cheol Jung)

1995 군산대학교 컴퓨터과학과 (학사)
1999 군산대학교 컴퓨터과학과 (석사)
2001~현재 군산대학교 컴퓨터과학과 박사과정
※관심분야 : 자연어처리, 정보검색, 음성인식, 유해 차단, 텔레메틱스

이태헌(Tae-hun Lee)

1993 군산대학교 컴퓨터과학과 (학사)
1998 군산대학교 컴퓨터과학과 (석사)
2001 일본 토쿠시마대학 지능정보과학과(박사)
※관심분야 : 지식 사전검색, 지적 문서관리 및 검색기술, 스트링 패턴 매칭, 정보검색 및 추출의 기초연구 및 응용, 자연언어처리의 기초해석(형태소, 구문해석 등)과 응용, 텔레메틱스

박기홍(Ki-hong Park)

1982 숭실대학교 전자계산학과 (학사)
1986 숭실대학교 전자계산학과 (석사)
1995 일본 토쿠시마대학 지능정보과학과(박사)
1997~1998 영국 Middlesex Univ 객원교수
1987~현재 군산대학교 컴퓨터과학과 교수
2004~현재 NURI사업 텔레메틱스 인력양성 사업단(군산대) 단장
※관심분야 : 자연어처리, 정보검색, 소프트웨어통합, 텔레메틱스