텍스트베이스에 관한 정보 검색을 위한 정보 표시 모델의 설계 및 구현

박 재 완

1. 서 론

사람은 일상생활속에서 여러 가지 유형(전공 서적, 신문, 잡지등)의 자료들을 통해서 지식을 얻고 또한 기억한다. 인간의 지능은 우수한 능력을 지니고 있으므로 필요한 부분들을 분류하여 기억할 수 있다. 그러나 인간의 기억력 또한 무한한 것은 아니므로 시간이 경과하면서 기억된 자료들을 점차 상실하게 된다. 하지만 컴퓨터와 같은 기계에 이러한 자료들을 저장해 놓으면 영원한 자료들의 저장 장소로 적합할 것이다. 이런 이유로 인해 정보 과학 분야에서의 언어 정보의 중요성을 인식하여 한국어 정보 처리에 필요한 모든 정보를 담은 어휘 데이타베이스(lexical database)를 개발하고 최신 컴퓨터 기술을 이용한 전자 사전 개발의 중요성이 대두된다[2][4][7].

본 연구의 취지는 이러한 자료 유형들을 보다 효율적으로 관리하고 유지 보수를 하기 위해서 자료들에 대한 화일 처리와, 화일 처리된 자료를 필요시에 다시 검색하여 그 자료를 필요로 하는 사용자에게 제시해 주는 정보 검색 시스템에 대한 연구이다.

현재 본 연구의 바탕인 전자 사전 개발실에서는 본 대학 한국어 사전 편찬실에서 신뢰성 있고 타당한 한국어 말뭉치를 표본 선정 기준으로 정하기 위해 설문 조사를 통해 〈표 1〉과 같은 300만 말뭉치를 이미 구축해 놓았다[4][5]

[7]. 그 말뭉치 속에는 텍스트외의 정보를 말뭉치에서 유지하기 위하여 여러 가지의 정보 표시(tagging)를 붙여 놓았다[6]. 이러한 정보 표시의 사용 목적은 첫째로는 텍스트외의 모든 정보를 유지시키기 위함이고, 둘째로는 말뭉치에 들어있는 자료들을 분류하기 위한 목적이고, 셋째로는 분류를 통한 자료 저장과 정보 검색에 그 목적을 둔다[11].

현재 주어진 300만 말뭉치에 대한 자료 유형의 종류로는 신문, 잡지, 소설 및 수필, 취미 및 교양, 수기 및 전기 그리고 교과서 등 여러 종류로 나뉘어져 있고 이러한 유형들에 정보표시를 줌으로 해서 다음에 임의의 유형의 자료를 검색할 때 검색의 편리함과 화일 구성의 효율성을 증대시킬 것이다. 이러한 정보 표시(이하 태그라 부름)의 종류로는 현재 표본의 종류를 분류하는 유형 태그와 표본의 출처를 나타내어 주는 태그, 제목에 대한 태그, 저자에 대한 태그, 표본의 생성 일자를 나타내어 주는 태그, 표본의 발췌 페이지에 대한 태그, 표본의 내용을 요약한 태그와 분류 태그 이외의 그 표본에 속해 있는 단어들의 유래를 나타내어 주는 태그(한자, 한글, 외래어 등)들이 현재 표본 분류 태그로 쓰이고 있다.

이러한 태그를 기입하는 것도 아직은 컴퓨터가 할 수 없으므로 인간의 수작업으로 이루워진다. 그러므로 이로 인한 많은 태그에 대한 오류가 발생하게 된다. 본 연구에서는 이러한 태그의 오류를 검출하고 그 검출된 오류를 정확한 태그로 바꾸어 주는 소프트웨어 개발과 또, 수정된 태그를 바탕으로 효율적인 자료 처리와 검색을 위한 화일 구조를 제안한다. 이 화일 구조는 기본적으로 계층 데이타베이스(hierarchical database) 모델에 바탕을 두고 있고 효율적인 항해(navigation)를 위해 역 링크(invert link)가 부가된다. 또한, 이미 전자 사전을 구축한 OED의 PAT 시스템[9]의 '질의어 방식(query method)'과는 달리 풀-다운(pull-down)과 질의적 검색을 이용한 '메뉴 유도 방식(menu-driven method)'을 정보 검색 시스템에서 사용한다.

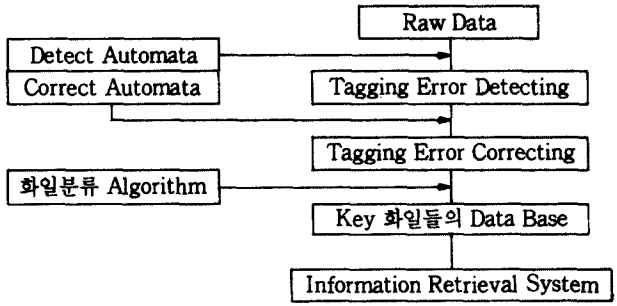
제 2장에서는 정보 표시 오류에 대한 오류 검출과 오류 수정에 관해 설명하고 제 3장에서는 수정된 자료에 대한 정보표시별로의 화일 처리에 관해 설명하고 제 4장에서는 인덱스된 화일 구조를 가지고서 메뉴 유도 방식을 통한 정

보 검색 시스템에 대해 설명하고 제 5장에서는 수행 결과 및 분석에 관하여 논하고 제 6장에서는 결론을 논하였다.

신문	33 %
잡지	20 %
소설및수필	18 %
취미및교양	10 %
수기및전기	9 %
교과서	5 %
희곡및시나리오	5 %
계	100 %

〈표 1〉 전문가 및 일반인 조사 결과에 의한 말뭉치 표본 선정 기준

본 연구의 전체적인 구조에 대한 설계는 〈그림 1〉과 같다.



〈그림 1〉 본 연구의 총괄적인 구조

2. 표본 데이타의 정보 표시 오류(Tagging Error) 처리

표본 데이타는 기계가 데이타를 읽어서 적절한 형식에 맞게 만든 것이 아니고 인간의 수작업을 통해서 얻어진 데이타이므로 이 표본에 많은 정보 표시 오류가 있다. 이러한 오류들을 찾아서 올바른 형태의 정보 표시로 바꾸어 주는 처리가 필요하다. 정보 표시 오류 처리 방법은 부록에 있는 〈표 2〉와 같은 태그 표(Tagging Table)를 기준으로 만들었다. 그러나 실제로 표본 자료는 이러한 태그표의 순서와는 무관하게 처리된 오류들이 상당히 많이 존재했다.

이러한 정보 표시 오류의 종류로는 여러가지가 다음에 설명되지만 그 중 순서에 대한 오류 문제에 있어서 어느 정도의 순서를 부여한 후 처리한다. 즉, 표본의 시작태그 다음에는 유형, 출처, 주제목 순으로 순서를 주고 그 이외의 년월 태그와 저자 태그는 항상 존재하게 구성한다. 단, 표본 유형이 신문인 경우에는 저자가 생략되어 있으므로 저자 태그의 존재를 무시하고 처리한다. 그 이외의 태그는 존재 여부를 임의적으로 구성한다. 순서가 필요한 태그가 삭제되었을 때는 임의적으로 그 태그를 생성시켜 처리한다. 위와 같은 규칙은 〈표 2〉와 많은 자료 처리에 있어서의 경험을 토대로 2. 1과 같은 정보표시 오류 처리 자동 장치(Automata)를 제안한다. 이 자동 장치에 대한 syntax diagram은 부록에 실었다.

2. 1 정보 표시의 오류 검출과 수정 자동 장치 (Tagging Error Detecting and Correcting Automata)

다음과 같은 정보표시의 오류 및 수정 자동 장치를 이용하여 정보 표시의 오류와 수정 작업을 수행한다.

sample ::= 〈ㅍㅎ〉 contents 〈/ㅍㅂ〉 dummy sample | ∈
contents ::= 〈ㅇㅎ〉 gettypeno 〈/ㅇㅎ〉 〈ㅊㅊ〉 chunks 〈/ㅊㅊ〉
〈ㅈㅁ〉 chunks 〈/ㅈㅁ〉 {〈ㅂㅁ〉 chunks 〈/ㅂㅁ〉}*
{rests1 | rests2 | rests3}
dummy[1] ::= !istag | ∈
gettypeno[2] ::= digit+
istag ::= 〈ㅍㅂ〉 | 〈/ㅍㅂ〉 | 〈ㅇㅎ〉 | 〈/ㅇㅎ〉 | 〈ㅊㅊ〉 |
〈/ㅊㅊ〉 | 〈ㅈㅁ〉 | 〈/ㅈㅁ〉 | 〈ㅂㅁ〉 | 〈/ㅂㅁ〉 |
〈ㅈㅈ〉 | 〈/ㅈㅈ〉 | 〈ㄴㅇ〉 | 〈/ㄴㅇ〉 | 〈ㅉ〉 | 〈/ㅉ〉 |
〈ㅇㅇ〉 | 〈/ㅇㅇ〉 | 〈ㅎㅎ〉 | 〈/ㅎㅎ〉 | 〈ㅇ〉 | 〈/ㅇ〉 |
〈ㅎㅇ〉 | 〈/ㅎㅇ〉 | 〈ㅎㅎㅇ〉 | 〈/ㅎㅎㅇ〉 | 〈〉 | 〈/〉
rests1 ::= 〈ㄴㅇ〉 dummynw 〈/ㄴㅇ〉 {〈ㅇㅇ〉 chunks 〈/ㅇㅇ〉} tail
dummynw# ::= digit+
digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
rests2 ::= {〈ㅈㅈ〉 chunks 〈/ㅈㅈ〉} 〈ㄴㅇ〉 dummynw 〈/ㄴㅇ〉

```
                〈ㅉ〉 dummy 〈/ㅉ〉 {〈ㅇㅇ〉 chunks 〈/ㅇㅇ〉} tail
rests3      ::= {〈ㅅㅅ〉 chunks 〈/ㅅㅅ〉} 〈ㄴㅇ〉 dummynw 〈/ㄴㅇ〉
                〈ㅉ〉 dummy 〈/ㅉ〉 tail
chunks      ::= 〈ㅎ〉 dummy 〈/ㅎ〉 chunks |
                〈ㅇ〉 dummy 〈/ㅇ〉 chunks |
                〈ㅎㅎ〉 dummy 〈/ㅎㅎ〉 chunks |
                〈ㅎㅇ〉 dummy 〈/ㅎㅇ〉 chunks |
                〈ㅎㅎㅇ〉 dummy 〈/ㅎㅎㅇ〉 chunks |
                dummy chunks |
tail        ::= chunks {〈ㅂㅁ〉 chunks 〈/ㅂㅁ〉 chunks}*
```

2.2 정보 표시의 오류 및 수정 자동 장치에 대한 구체적 설명

정보표시의 오류를 검출하고 또한 수정하는데 있어서, 태그의 무작위 순서가 허용되어 있는 상황에서는 처리가 불가능하다고 생각되어 부록편의 〈표 2〉와 많은 표본 자료들에 기초를 두고서 일정한 정보표시 규칙(tagging rule)을 찾아내어 그것을 자동 장치로 표시한 것이다.

(1) 사람에 따라 범할 수 있는 오류가 다양하기 때문에 그 모든 것을 제대로 수정하기는 힘들다. 하지만 오류를 검출하는데 있어서는 의미상(semantic)오류의 검출을 제외하고 100% 검출하게끔 하였다.

(2) 모든 표본에 있어서 〈ㅇㅎ〉 태그는 항상 〈ㅍㅂ〉태그 뒤에 사용되었다. 때문에, 이외에서 사용된 것은 모두 〈ㅎㅇ〉의 잘못된 표현이라고 간주하여 〈ㅎㅇ〉로 치환한다.

〈ㅇㅎ〉보상돈〈/ㅇㅎ〉 ==〉〈ㅎㅇ〉보상돈〈/ㅎㅇ〉

(3) 가장 문제가 되는 것은 요약(〈ㅇㅇ〉) 태그인데 이것은 외래어(혹은 영어)의 뜻인 〈ㅇ〉의 잘못된 표기로 사용된 예가 많았고 〈ㅇㅇ〉의 위치가 일정하지 않았다. 그래서 〈ㅇㅇ〉은 항상 선택적으로 주어지게 했다. 또한 〈ㅂㅁ〉

1) !X 는 X를 제외한 모든 것을 받아들이는 것으로 정의한다.
2) gettypeno 와 dummynw의 처리시 숫자 이외의 것은 무시해야 한다.

~〈/ㅂㅁ〉 사이에 〈ㅇㅇ〉이 있는 경우나 있어야 할 위치가 아닌 곳에 〈ㅇㅇ〉이 있을 경우는 모두 〈ㅇ〉으로 치환시킨다. 이러한 처리는 많은 표본 자료들에서 얻은 경험을 바탕으로 처리한 결과이다.

〈ㅂㅁ〉사진의 가장 중요한것은 〈ㅇㅇ〉타이밍〈/ㅇㅇ〉〈ㅂㅁ〉
==〉〈ㅂㅁ〉사진의 가장 중요한 것은 〈ㅇ〉타이밍〈/ㅇ〉〈/ㅂㅁ〉

(4) 〈ㅉ〉 태그가 사용될 경우에는 항상 〈ㄴㅇ〉뒤에 사용된다. 그리고 〈ㅈㅈ〉는 사용될 경우 항상 〈ㄴㅇ〉앞에 온다. 그러므로 〈ㅈㅈ〉가 〈ㄴㅇ〉뒤에 올 경우에는 〈ㅉ〉로 치환시킨다.

〈ㄴㅇ〉7600〈ㄴㅇ〉〈ㅈㅈ〉253-275〈ㅈㅈ〉
==〉〈ㄴㅇ〉7600〈/ㄴㅇ〉〈ㅉ〉253-275〈/ㅉ〉

(5) 끝 태그에 slash('/')가 빠진 경우는 항상 삽입한다.

〈ㅇㅎ〉61〈ㅇㅎ〉 ==〉〈ㅇㅎ〉61〈/ㅇㅎ〉

(6) 태그 처리에 있어서 태그의 짝짓기(pair matching)를 기본으로 해 준다. 때문에 앞 태그나 뒷 태그중 하나가 빠지면 나머지 하나를 삽입한다. 들어가 있어야 할 자리에 앞 태그, 뒷 태그 모두가 빠진 경우는 둘 다 삽입한다. 유형의 내용이 빠질 때와 불완전한 화일일 경우, 화일의 유형 태그 내용에 99를 삽입한다. 즉, 99 유형은 유형 내용이 빠지거나 불완전한 화일들의 모임이 된다.

〈ㅍㅂ〉〈ㅇㅎ〉〈ㅈㅊ〉국정교과서 주식회사〈ㅈㅊ〉
==〉〈ㅍㅂ〉〈ㅇㅎ〉99〈/ㅇㅎ〉〈ㅊㅊ〉국정교과서주식회사〈/ㅊㅊ〉

(7) 표본 데이타에서는 태그의 어느 일부분이 삭제되어 쓰여진 오류가 있다. 이런 오류는 다음에서 설명하는 제 1 단계(PASS 1)에서 좌우 괄호를 〈, 〉 에서 (,)로 바꾸고 그 이후에 빠진 부분의 태그 표시를 해준다. 이 단계에서는 순서가 부여되는 오류는 순서에 맞게 수정하고, 그 이외의 오류는 태그 사용 목적인 모든 자료의 유실을 방지[12]하기 위해서 잘못 쓰인 것이 아닌

필요에 의한 표시인 것으로 간주하고 처리하기 위해서이다.

〈ㅍㅂ.〈ㅇㅎ〉64〈ㅇㅎ〉
==〉〈ㅍㅂ〉〈ㅇㅎ〉64〈/ㅇㅎ〉

이루워질 수 있는〈ㅎ. 방법〈/ㅎ〉
==〉이루워질 수 있는(ㅎ)〈ㅎ〉방법〈/ㅎ〉

2.3 오류 검출과 수정(Error Detection and Correction)단계

오류 검출과 수정을 위해 크게 두 단계로 나누어 처리한다.

2.3.1 제 1 단계 (PASS 1)

시작 태그 기호 '〈'와 끝 태그 기호 '〉'은 오로지 태그를 위해서 사용 되어야 함에도 불구하고 외국 지명이나 인명 혹은 인용문을 위해 사용되는 경우가 많았다. 이것을 방치했을 때 많은 문제점들이 발생했다. 그래서 제 1 단계에서는 표본 화일을 읽어 들이면서 태그 이외의 목적에 '〈'나'〉'을 사용했을 경우에는 모두 '('나')'로 바꿔 버린다. 이때 인용문 안에 태그가 있는 경우나 태그 없이 4자 이상의 글자를 포함하고 있는 경우만 처리해 준다. 3자 이하의 경우에 대해서는 제 2 단계 (PASS 2)에서 처리한다. 3자 이하는 태그일 가능성이 크기 때문이다.

〈문둥북춤〉 =〉 (문둥북춤)

2.3.2 제 2 단계 (PASS 2)

① 태그의 형식을 취한 것 중에서 태그의 내용을 자소별로 분석하여 태그의 진위를 밝혀 태그가 아니라고 판명난 것은 모두 (,)로 치환시킨다.

〈나라〉 =〉 (나라)

② ①에 의해 태그라고 판명났지만 실제로 태그가 아닌 것은 그 위치에 맞는 적절한 태그로 치환시켜 준다.

〈ㄴㅁ〉8500〈ㄴㅁ〉 =〉 〈ㄴㅇ〉8500〈/ㄴㅇ〉

③ 앞과 뒷 태그가 맞지 않을 경우 기본적으로 앞 태그에 맞추어 뒷 태그가 따라가게 했다. 때문에 앞 태그와 뒷 태그가 일치되지 않을 경우는 뒷 태그를 앞 태그에 맞추어 고친다.

〈ㅇ〉지엔피〈/ㅎ〉 =〉 〈ㅇ〉지엔피〈/ㅇ〉

④ 완전히 틀린 태그 오류는 오류 수정이 불가능하다. (semantic error)

〈ㅎ〉 티브이 〈/ㅎ〉 =〉 오류복구 불가능

3. 효율적인 정보 검색과 유지 보수를 위한 화일 구조

본 논문에서 제안하는 화일 구조는 기본적으로 계층 데이타베이스 모델에 바탕을 두고 있다. 각 표본 데이타들은 유형, 출처, 주제목, 부제목, 저자, 년월, 쪽, 요약, 표본 텍스트들로 구성되어 있다. 이런 것들은 하나의 표본 데이타를 구성하는 요소들로 계층 구조적인 성질을 지니고 있다. 즉, 하나의 표본이 부모 레코드이고 그 표본은 유형으로 분류 되어지므로 곧 유형은 표본의 대표적인 속성값이다. 나머지 출처, 주제목등은 표본에 대한 자식 레코드이므로 이런 식의 계층 구조 설계(hierarchical structure scheme)를 구성할 수 있다〈그림 3〉. 또, 항해(navigation)시 효율적인 측면을 고려하여 역 링크(invert link)를 구성해 놓았다. 이 역 링크는 하나의 자식 레코드에서 부모 레코드로 직접 항해할 수 있도록 구성한다.

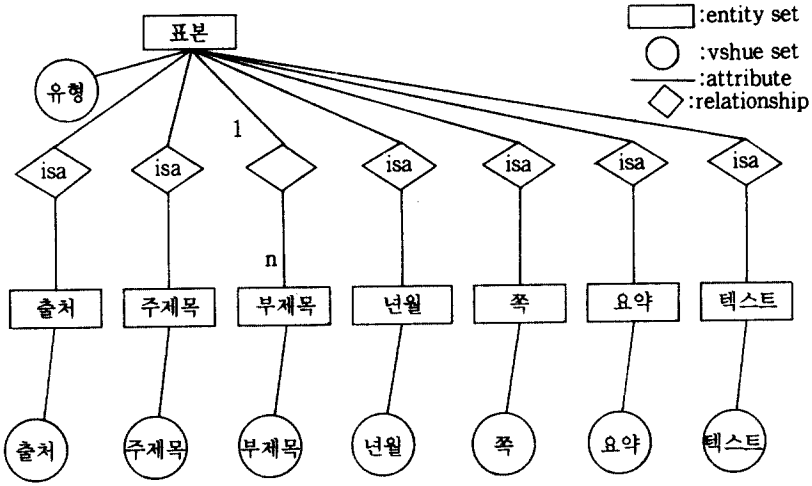
3.1 E-R 모델

하나의 표본은 유형으로 대표되고, 하나씩의 출처, 주제목, 저자, 년월, 쪽,

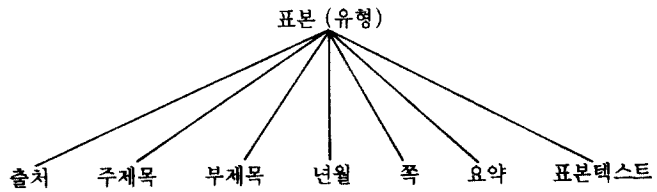
요약, 표본 텍스트들이 존재하며 하나 또는 그 이상의 부제목들이 존재한다. 따라서 〈그림 2〉와 같은 E-R 모델을 가정할 수 있다.

3. 1. 1 계층 구조 스키마 (Hierarchical Structure Scheme)

〈그림 2〉와 같은 E-R 모델은 특성상 계층 구조 모델과 유사한 성격을 갖고 있다. 따라서 본 논문은 〈그림 3〉과 같은 계층 구조 Scheme을 제안한다. 하나의 표본은 부모 레코드가 되며, 속성으로는 그 표본의 유형을 유지한다. 또, 그 표본을 구성하는 나머지 요소들은 자식 레코드들이 되며 서로 다른 화일들에 유지된다.



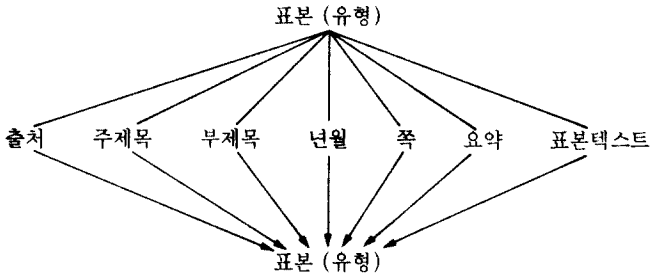
〈그림 2〉 본 논문에서의 화일구조 E-R 모델



〈그림 3〉 계층 구조 스키마

3. 1. 2 역 링크 (Invert Link)

계층 구조 모델은 부모 레코드에서 자식 레코드로 항해할 수 있는 링크가 존재한다. 따라서, 부모 레코드에서 자식 레코드의 항해는 손쉽게 할 수 있다. 그러나 어떤 자식 레코드에서 부모 레코드를 찾아갈 수 있는 방법은 존재하지 않는다. 이런 문제점은 검색이 수행될 때 큰 비효율성을 나타낸다. 즉, 항상 현재 탐색하는 레코드의 표본이 무엇인지 유지하고 있어야 한다. 이런 이유로 모든 자식 레코드들은 부모 레코드에 대한 역 링크를 유지하고 있다 〈그림 4〉.



〈그림 4〉 역 링크 (invert link) 구조

3. 2 물리적 화일 구조 (Physical File Structure)

태그들 중에도 표본 화일을 분류하고, 유지 보수를 하기 쉽게 구성 가능한 태그들로는 유형, 출처, 주제목, 부제목, 저자, 년월, 쪽, 요약 태그이고, 이러한 종류의 태그들을 키(key) 태그로 분류하고 그 외의 모든 태그는 표본 태그로 취급한다.

다음에 설명할 각 화일에 대한 바이트 수는 화일에 대한 검색을 임의적 접근(random access)이 가능하도록 하기 위해서 화일 레코드 크기를 고정시켜야 한다. 그러나 모든 화일의 각 키 태그에 대한 내용의 크기가 가변적이기 때문에 많은 표본 자료를 분석한 결과 가장 적절한 레코드 크기를 각 화일별로 배정했다(heuristic method). 각 화일의 내용이 결정된 레코드 크기보다 더

를 경우에 대해서는 3. 3절에서 설명한다. 다음 각 태그별로의 화일 구조 구성에 관해 논한다.

(1) 유형 화일 (태그 표시 : 〈ㅇㅎ〉 ~ 〈/ㅇㅎ〉)

유형 태그는 300만 말뭉치를 구분하기 위한 표본들의 유형을 나타낸다. 이 유형 화일의 인덱스를 이용해서 동일한 유형 화일들을 한데 모으기가 용이하다.

예) 신문, 잡지, 소설 및 수필등

또한 유형 태그는 각 표본 유형에 적절한 정수값(integer number)으로 각 유형을 종류별로 세분화 시킨다. 그러므로 표본은 유형별로 모두 분류되어질 수 있다. 예를들어 신문의 '정치'와 같이 분류된다.

● 화일 구조 (4 바이트)

· 유형에 대한 화일 처리는 현재 존재하는 유형(20가지 유형)외에도 확장을 고려하여 1-99까지의 유형들이 존재한다고 가정하고서 처리를 한다. 이 중에서 99 유형은 표본의 유형 부분의 삭제나 또는 태그의 오류들을 묶은 화일이다.

· 유형 화일은 4 바이트로 같은 유형의 제일 마지막번째의 주(main) 화일에 대한 인덱스만을 가지고 있다. 이 유형 화일과 주 화일과의 관계는 주 화일 설명시 언급한다.

(2) 출처 화일 (태그 표시 : 〈ㅊㅊ〉 ~ 〈/ㅊㅊ〉)

출처 태그는 각 표본 유형의 출처(펴낸곳)를 나타내는 태그이다. 이 출처는 태그를 통해서 같은 유형의 표본이라도 출처가 다르면 표본 내용이 달라지므로 이 출처 태그 또한 키 태그가 된다.

예) 신문 '정치' 유형에 대한 출처 ==〉 '동아일보', '서울신문'등

● 화일 구조 (14 바이트)

· 출처 태그는 대부분 책 이름이나, 신문 이름을 나타내는 태그이므로 최대 5글자로 잡고 10 바이트로 처리한다. 출처 화일 또한 정보검색시 주요한 키 화일이 되므로 인덱스 처리를 하기위해 4 바이트의 인덱스 필드를 가진다. 이 4 바이트는 자기 자신이 속한 주 화일을 지시한다.

(3) 주제목 화일 (태그 표시 : 〈ㅈㅁ〉 ~ 〈/ㅈㅁ〉)

주제목 태그를 키 태그로 선택한 이유는 사용자 측면에서 찾고자 하는 내용을 검색시 화면에 보여줌으로 인해 사용자가 원하는 내용의 화일을 쉽게 선택하게 하기 위해서 주제목 화일을 만든 것이다.

● 화일 구조 (34 바이트)

· 주제목의 내용은 그 표본의 가장 핵심이 되는 내용을 나타내므로 길이가 그다지 길지 않으므로 15자 정도를 최대로 정했다. 최대 글자수를 15자로 잡았으므로 30 바이트 크기이다. 나머지 4 바이트에 대한 설명은 출처 화일 때와 동일하다.

(4) 부제목 화일 (태그 표시 : 〈ㅂㅁ〉 ~ 〈/ㅂㅁ〉)

부제목 태그를 이용한 화일의 생성 목적은 주제목과 동일하다.

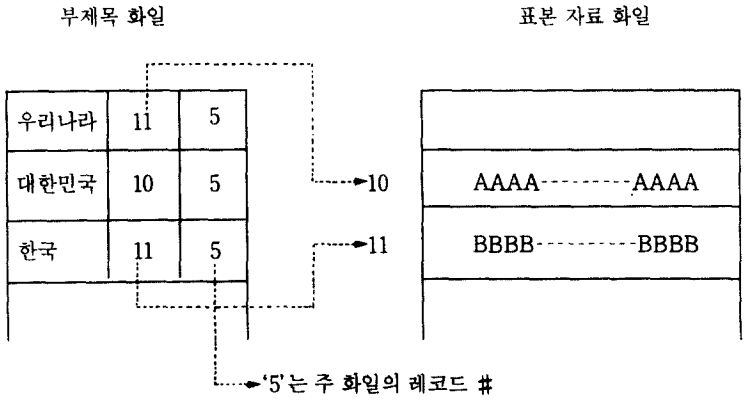
● 화일 구조 (38 바이트)

· 단 부제목은 같은 표본 내에서도 서로 내용을 달리하며 여러번 쓰일 수 있으므로 같은 표본 유형을 지시하는 주 화일 인덱스 포인터 (4 바이트)와 자기 자신이 속해 있는 표본 화일의 레코드 번지를 지시하는 인덱스 포인터(4 바이트)를 가지고 있다. 그러나 주 화일의 부제목에 대한 포인터는 8 바이트가 아닌 4 바이트만 가지고 있으면 된다. 이 4 바이트는 현재 주 화일이 지시하는 부제목 화일의 시작 레코드 번지를 나타낸다. 정보검색시 주 화일에 있는 부제목 인덱스를 찾아가서 같은 주 화일에 대한 인덱스를 가지고 있는 모든 부제목들을 검색하는데 용이하고 필요시에는 부제목에 해당되는 표본 내용만 검색하기가 편리하다 (〈그림 5〉)

〈ㅍㅂ〉

〈ㅂㅁ〉우리나라〈/ㅂㅁ〉……〈ㅂㅁ〉대한민국〈/ㅂㅁ〉
AAAA------AAAA
〈ㅂㅁ〉한국〈/ㅂㅁ〉
BBBB------BBBB

〈/ㅍㅂ〉



즉, 모두 같은 표본임을 나타냄

〈그림 5〉 부제목 화일 구조

(5) 저자 화일 (태그 표시 : 〈ㅈㅈ〉 ~ 〈/ㅈㅈ〉)

저자 태그 또한 사용자가 같은 표본 유형에서도 어느 특정 인물이 쓴 내용을 필요로 할 때 정보검색시 필요한 키 태그이다. 저자에 대한 내용은 신문 유형에서는 삭제되어 있고 그 이외의 나머지 유형에 대해서는 필히 들어있다.

● 화일 구조 (14 바이트)

· 출처의 화일 구조 생성 목적과 동일하다.

(6) 년월 (태그 표시 : 〈ㄴㅇ〉 ~ 〈/ㄴㅇ〉)

'년월'은 앞의 화일들과 같이 따로 화일을 구성한 것이 아니고 4 바이트 정

수 유형(integer type)으로 버퍼를 통해 주 화일의 4 바이트로 받아들인다. 현재 300만 어휘뭉치 내에서의 년월 태그에 대해서는 확실성이 부여되어 있지 않으므로 이 '년월'에 대한 내용은 검색후 필요시 주 화일에 쓰여 있는 내용을 화면에 보여주기 위한 목적으로 태그 필드에 포함시킨다. 그러므로 인덱스 포인터에 대한 바이트 수는 필요하지 않다. 또 필요시에는 화일로서의 확장이 용이하다.

(7) 쪽(page) 화일 (태그 표시 : 〈ㅉ〉 ~ 〈/ ㅉ〉)

'쪽'에 대한 화일 구성은 출처, 저자와 동일하다. 다른 면은 '쪽'도 '년월'과 마찬가지로 이 '쪽' 화일을 통한 검색은 이루어지지 않는다. 왜냐하면 어느 책 몇 페이지에서 발췌했다는 내용을 사용자가 기억하기 어려우므로 이 화일도 역시 검색이 다 이루어진 이후에 다른 화일의 내용과 함께 화면상으로 보여줌으로 인해 사용자에게 발췌 페이지를 알려주는 역할을 한다. '년월'처럼 주 화일 안에 바로 기입해 두기 어려운 것은 '년월'은 4바이트로 고정되어 있지만 '쪽'은 레코드 길이가 가변적이므로 따로 화일을 구성해야 한다.

● 화일 구조 (14 바이트)

· 현재 '쪽'에 대한 4 바이트의 인덱스 포인터는 주 화일의 인덱스 포인터를 나타내는 것이 아니고 임의의 번지가 들어 있다. 검색시 주 화일을 통해서만 쪽 화일을 검색하므로 어떤 문제도 발생되지 않는다. 나머지 10 바이트에는 페이지 내용이 들어 있다.

(8) 요약 화일 (태그 표시 : 〈ㅇㅇ〉 ~ 〈/ㅇㅇ〉)

요약 화일의 필요성도 역시 '년월', '쪽'화일과 마찬가지로 검색 후에 사용자에게 이 화일에 있는 요약 내용을 보여주기 위해 필요하다. 왜냐하면 이 요약 화일을 통한 정보 검색은 유형, 출처, 저자, 주제목, 부제목과는 달리 검색 수행시 주 기억장치의 공간을 상당히 많이 필요로 하므로 시간과 메모리 낭비를 초래한다[10].

● 화일 구조 (128 바이트)

· 요약 화일의 내용은 표본의 내용 크기와 같으므로 디스크 헤드가 한번 움직일 때 읽어 들이는 레코드 크기를 기준으로 해서 128 바이트로 결정했다. 그 중 124 바이트는 요약 태그에 대한 내용이 들어가고 나머지 4바이트에 대한 내용은 출처, 저자의 화일 구성 방법과 동일하다.

(9) 표본 화일 (그 이외의 태그들)

표본 화일의 내용은 그 앞의 모든 키 태그들과 그 내용을 제외시킨 그 이외의 표본 화일의 내용을 담아두기 위한 화일이다.

이 표본 화일의 내용은 사용자가 원하는 표본의 내용이므로 키 태그를 통한 화일로써 선택한 내용을 사용자에게 화면을 통해서 보여주기 위해 모아둔 화일이다.

● 화일 구조 (128 바이트)

· 표본 화일의 생성 목적은 요약 화일과 동일하다.

(10) 주(main) 화일

검색시 꼭 필요한 화일로서 모든 레코드 내용은 표본 화일에 대한 인덱스들로만 구성되어져 있다. 즉, 한 표본이 처리될 때마다 하나의 주 화일이 생성된다.

· 다음에 검색할 때 이 주 화일의 인덱스 내용을 따라가며 검색하므로 검색하는데 있어 효율성을 제시하는 화일이다.

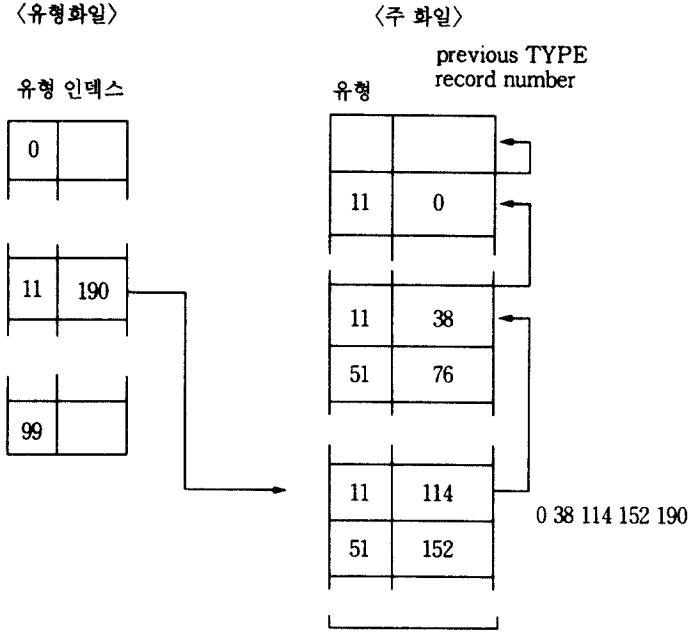
● 화일구조 (38 바이트)

· 주 화일의 38 바이트는 유형 부분(6 바이트)과 년월 부분(4 바이트)을 제외하고는 모두 인덱스 값들로 이루워져 있다. 이 인덱스들은 각각 해당되는 화일을 지시하므로 주 화일을 통해서 다른 화일을 쉽게 검색할 수 있게 구성한 것이다.

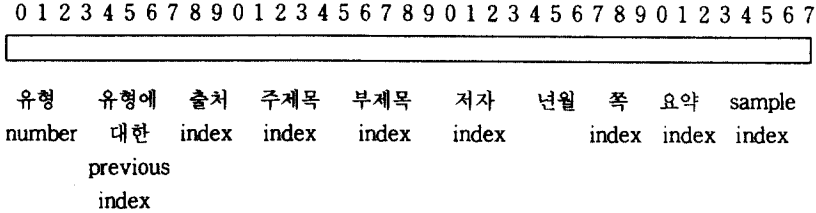
· 유형 필드는 앞에서 설명한 유형 값을 받아들이기 위한 2 바이트와, 동일한 유형들을 연결 하기 위한 인덱스 포인터(previous type record number)로 4 바이트가 필요하다. 주 화일과 유형 화일과의 관계를 그림으로 나타

내면 〈그림 6〉과 같다.

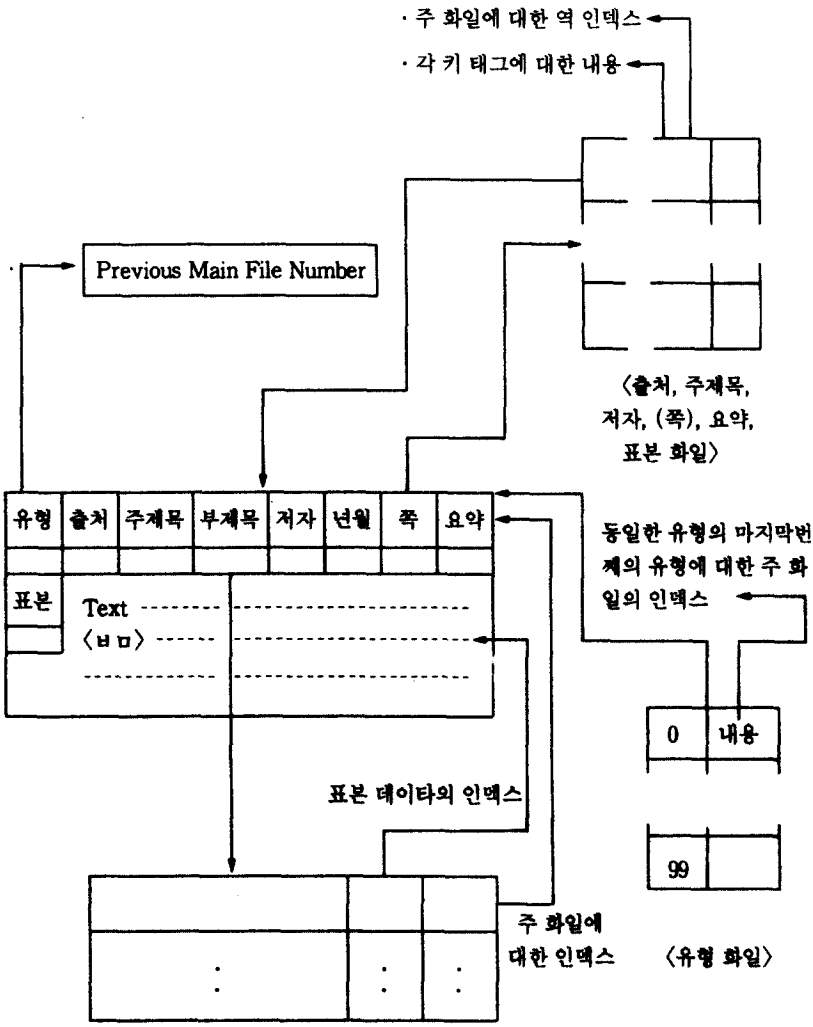
· 키 태그를 추가할 때는 그 태그에 대한 화일 구성과 주 화일에 그 태그에 대한 인덱스 필드만 늘려 주면 되므로 확장이 용이하다. 주 화일의 구성을 그림으로 설명하면 〈그림 7〉과 같다. 〈그림 8〉은 현재 있는 키 태그에 대한 전체 화일들에 대한 구성을 나타낸다.



〈그림 6〉 유형 화일과 주 화일의 관계



〈그림 7〉 주 화일의 인덱스 구조



〈그림 8〉 전체적인 화일 구조

3.3 임의적 접근(Random Access)이 가능하고 길이가 가변적인 레코드

화일 구조를 구성할 때 화일 검색을 임의적 접근(random access)이 가능하도록 하기 위해서는 화일 레코드 크기를 고정시켜야 한다. 이때 레코드 크기를 결정하는 것이 매우 중요하다. 레코드 크기가 너무 길면 화일 레코드의 낭비가 심해지고 너무 적게 레코드 크기를 잡으면 데이타의 손실이 유발되므로 다음과 같은 방식을 제시한다.

● 새로운 화일 구성 방법

· 레코드 크기를 고정시키면서 각 레코드 끝에 포인터 필드를 두어 하나의 레코드에 수용할 수 없는 데이타는 그 포인터 필드에 연속(continue)태그 '0'을 주어 계속 되는 데이타임을 나타내고, 모든 데이타를 저장 시킨 후는 완료 표시인 END('@') 표시를 한 후 포인터 필드에 자신이 가르키는 화일에 대한 포인터를 기재한다. 또한 부제목 화일은 인덱스를 두개를 가지고 있으므로 END표시 이외에 SEND('~')표시를 두어 연결되어진 화일을 나타내도록 하였다. '@'와 '~'표시를 택한 이유는 한글 문자들과 서로 교차 되는 비율이 적은 문자를 사용하려한 목적에서 선택했다.

· 즉 탐색할 때 그 레코드의 END 부분에 도달하면 그 이후의 레코드 부분을 검색하지 않고 END 전까지의 자료를 검색하여 처리한다. 그러므로 비 정형적인 데이타 저장에 대한 메모리 측면도 어느 정도 효율성을 나타낼 수 있고 또한 검색 시간에 있어서도 저장되지 않은 필드를 검색하지 않으므로 효율성이 발휘된다.

결론적으로 주 화일을 임의적으로 접근하기 위해서는 화일의 크기를 새로운 화일 구성 방법에 의해 구성했다. 이로인해 화일 탐색을 빠르게 할 수 있고, 메모리 관리 측면에서 볼 때 선형 접근(linear access)시의 화일 크기를 미리 고정시키는 것에 비해 상당한 효율성을 발휘한다. 또한 각 화일의 레코드에는 역 링크 (invert link)가 구성되어 있기 때문에 어느 화일 (주제목, 부제목등)에서 탐색을 시작하여도 다른 모든 화일의 접근이 가능하다.

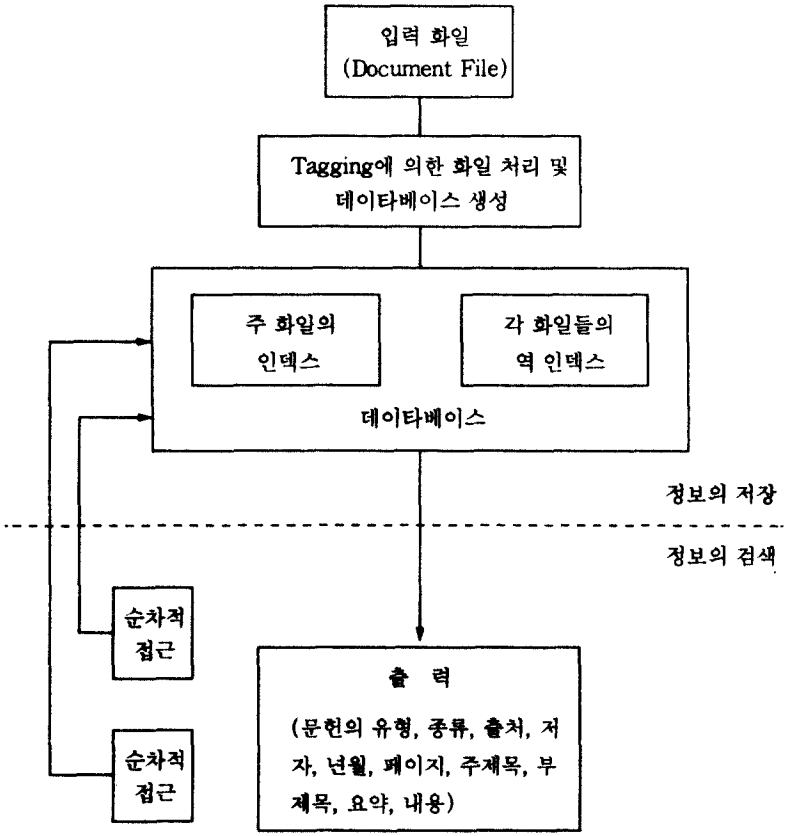
4. 정보 검색 시스템(Information Retrieval System)

4.1 정보 검색 시스템의 개요

검색(retrieval)이라는 용어와 탐색(search)이라는 용어는 모두 찾아 낸다는 의미를 가지고 사용되고 있다. 이 두 가지 용어의 차이점은 탐색이란 막연히 찾아 본다는 의미이지만 검색이란 찾은 자료를 원하는 사용자에게 보여주는 과정까지를 일컫는다[13]. 정보 검색 시스템은 시스템의 사용자가 필요로 하는 정보를 수집하여 내용을 분석한 뒤, 찾기 쉬운 형태로 조직하여 두었다가 정보에 대한 요구가 발생했을 때 정확한 정보를 신속하게 제공하는 것을 말한다[1].

검색 시스템에서 자료를 검색할때 검색의 키가 되는 것은 첫째 주제, 표제 등에서 중심이 되는 단어를 추출하여 그 단어를 키워드로 하여 검색하는 방법과 둘째 고정전거목록(fixed authority lists)이나 분류표를 사용하여 검색하는 방법 그리고 세째로는 한글 키워드인 초성 테이블을 이용한 검색[3]과, 네째로는 먼저 자연어(document file)를 읽어 들이고 이를 분석하여 키워드들을 추출한 뒤 이에 대한 Occurrence를 사전에 등록한다. 그런 다음, 각 키워드에 대한 검색이 가능하도록 역 데이타 화일(inverted data file)을 구성하는 방법[14]등이 있다.

본 논문에서는 이미 분류 저장 되어진 데이타베이스 내에 있는 각 키 태그에 의한 화일 구조를 검색의 키로 해서 다음과 같은 정보 검색 시스템을 제시한다. 화일 구조에서 임의 접근이 가능 하도록 이미 설명한 주 화일은 모든 키 화일에 대한 인덱스들을 가지고 있고 각각의 키 화일들은 주 화일에 대한 역 인덱스(invert index)를 가지고 있으므로 정보 검색시 화일 탐색을 할때 임의적으로 처리하므로 효율성이 부여된다. 그림으로 설명하면 〈그림 9〉와 같다.



〈그림 9〉 본 논문의 정보 검색 시스템

4. 2 항해 기법 (Navigation Method)

항해 기법은 크게 두가지 방법으로 나뉜다.

4. 2. 1 순차적 접근시 항해 기법

순차적 접근은 유형, 출처, 저자, 주제목, 부제목 순으로 접근된다. 이때, 항

해는 항상 표본(부모) 레코드에서 시작한다. 또, 항해할 표본 레코드들은 표본 주소 공간에 유지된다.

먼저, 어떤 유형이 선택되면 유형 화일에서 항해가 시작되어, 같은 유형의 표본 레코드들의 인덱스들을 표본 주소 공간에 모아둔다. 그 다음에 표본 레코드에서 자식 레코드로의 항해를 한다. 자식 레코드의 항해를 한 후에 그 레코드의 값들을 화면에 나타낸다. 다시, 사용자로부터 선택된 자식 레코드에 해당되는 표본 레코드들이 표본 주소 공간에 유지된다.

순차적 접근은 Display-Select Cycle을 형성한다. Display는 표본 주소 공간에 있는 표본의 자식들 중 순차적으로 출처, 저자, 주제목, 부제목을 나타내게 된다. Select는 display한 것 중 특정 값이 선택되며, 선택된 레코드의 부모에 해당되는 표본 레코드가 표본 주소 공간에 유지된다.

4. 2. 2 질의적 접근시 항해 기법

질의적 접근은 특정 자식 Entity Set에 대해서만 수행되며, 선택된 자식 레코드에 해당하는 표본이 지시될 수 있어야 한다. 이를 위해 다음과 같은 항해 기법을 사용한다.

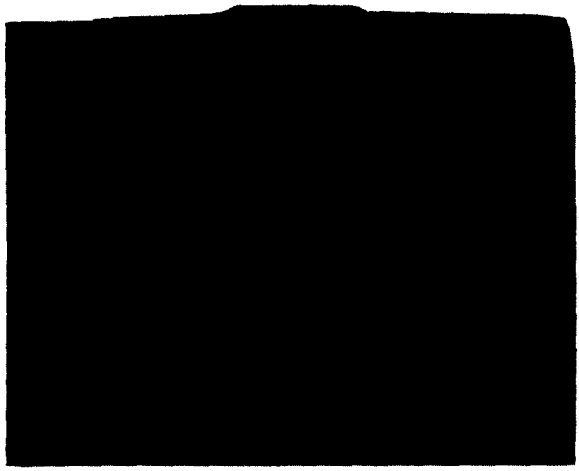
먼저, 입력된 질의어(query)를 특정 자식의 Entity Set에서 대응(matching)을 시도한다. 만약 대응된 레코드가 있다면 그 레코드가 유지하고 있는 역 링크을 이용해 부모 레코드로 항해를 한다. 다시, 부모 레코드에서는 자신의 모든 자식들에 대해 항해를 해 나가면서 표본의 모든 내용을 display 한다.

4. 3 키 화일을 이용한 정보 검색 시스템

현재 주어진 300만 말뭉치들의 표본의 유형에는 부록의 〈표 2〉와 같이 크게 6 가지로 분류 되어져 있다. 이러한 표본의 유형에 대한 각각의 분야들은 정보 표시중 유형 태그로 분류 되어 있고, 화일 구조의 주 화일에 각각 저장되어 있다.

본 논문의 정보 검색의 궁극적인 목적은 모든 키 화일들을 탐색해서 사용자에게 필요한 키 화일들의 내용과 그 키 화일에 대한 표본(sample) 화일에 대한 내용을 정보 표시를 이용하여 보여주는 것을 그 목적으로 한다.

본 논문에서의 정보 검색 시스템은 4. 2절의 항해 기법을 사용한 두가지 방법으로 분류된다. 즉, 순차적으로 검색하는 방법과 질의어를 통해서 검색하는 방법이 있다. 그림으로 본 논문의 정보 검색 시스템을 설명하면 〈그림 10〉과 같다.



〈그림 10〉 본 논문의 정보검색 시스템

4. 3. 1 순차적 접근 방법

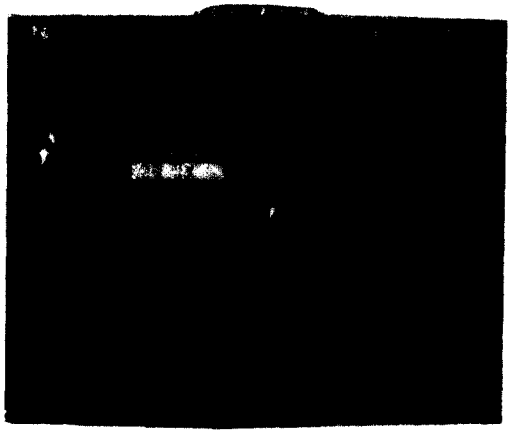
순차적 접근 방법은 사용자가 찾고자 하는 내용을 확실히 알지 못할 때 사용하는 방법이다. 순차적 접근 방법의 필드로는 유형, 출처, 저자, 주제목, 부제목, 표본 필드들이 있다. 이 접근 방법은 일정한 순서대로 사용자의 의도와는 관계없이 일련의 항목들을 선택해 나가게끔 하는 것이다. 각각의 필드에서 생성되는 선택 항목의 수는 순차적으로 어떠한 항목들을 선택하느냐에 따라 달라진다. 각각의 필드별로 설명하면 다음과 같다.

(1) 유형 필드

유형 필드를 선택하면 현재 데이타베이스에 수록된 모든 표본의 유형들이 display된다. 즉 신문, 잡지, 소설 및 수필, 취미 및 교양, 수기 및 전기, 교과서가 화면상에 나타난다. 사용자는 이들 중 하나 이상을 선택할 수 있다. 선택되어진 유형이 여러개일 때는 그들 각각을 OR 개념으로 묶어 처리하고 다른 필드들과의 관계는 AND 개념으로 묶어 처리한다. 사용자가 원하는 유형을 선택하게 되면 그 유형에 대한 종류들이 화면에 나타난다. 이런 방식으로 유형과 종류를 선택하게 되면 주 화일에 있는 2 바이트의 유형에 대한 값을 얻게 된다. 이 값을 유형 화일에서 찾아서 그 유형 화일에 있는 주 화일 인덱스를 얻게 된다. 이 인덱스 값은 선택된 유형에 대한 마지막 번째의 주 화일의 인덱스이고 또한 주 화일에는 같은 유형의 바로 이전의 주 화일에 대한 역 인덱스를 가지고 있으므로(previous type record number) 역 인덱스 값이 '0'이 나올때까지 유형 화일을 검색한다. 선택되어진 유형에 대한 주 화일에 있는 출처 인덱스를 찾아서 출처 화일에 있는 출처들을 화면에 출력하게 된다.

유형에 대해서 화면에 출력되는 메뉴의 크기는 가변적으로 형성된다. 즉, 그 유형의 내용 중 가장 긴 내용을 기준으로 메뉴 윈도우의 크기가 조종된다. 본 논문의 정보검색 시스템에서 지원되는 모든 메뉴 윈도우의 크기는 가변적으로 조절된다. 그림으로 설명하면 〈그림 11〉, 〈그림 12〉와 같다.

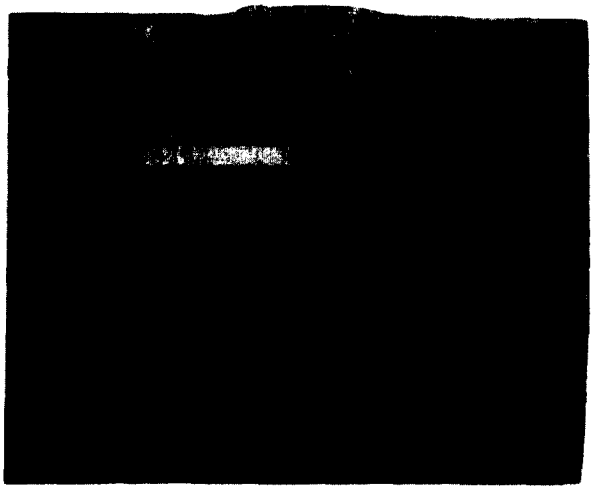
〈그림 11〉 순차적 접근에서의 유형 필드



〈그림 12〉 순차적 접근에서 유형 필드에 대한 종류들

(2) 출처 필드

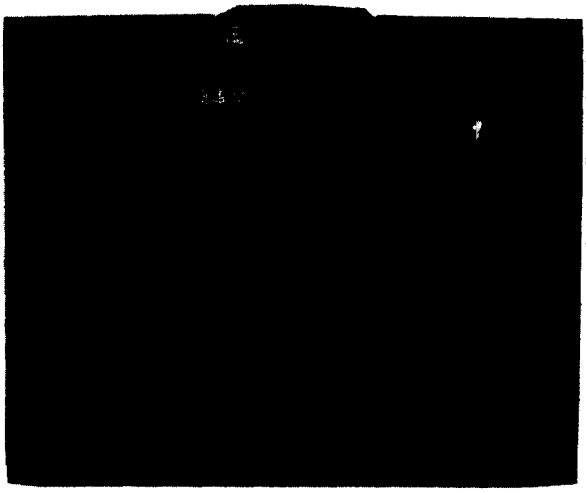
유형 필드에서 선택한 항목들에 대한 출처들이 화면에 출력된다. 원하는 출처를 선택하면 그 출처에 대한 처리가 시작된다. 처리 과정은 유형 필드에서 처리되는 과정과 같다. 원하는 내용이 없거나, 불 확실할 때는 항목을 선택하지 않고 넘어갈 수 있다. 그럴 경우는 유형 필드에서 선택된 모든 항목에 대한 주 화일들의 인덱스들을 가지고서 다음 필드로 넘어간다. 그림으로 설명하면 〈그림 13〉과 같다.



〈그림 13〉 순차적 접근에서 출처필드

(3) 저자 필드

출처 필드에서 사용자가 선택한 항목들에 대한 저자들이 화면에 출력된다. 그 이외의 사항은 '출처 필드'에서 처리하는 과정과 동일하다. 그림으로 설명하면 〈그림 14〉와 같다.



〈그림 14〉 순차적 접근에서의 저자필드

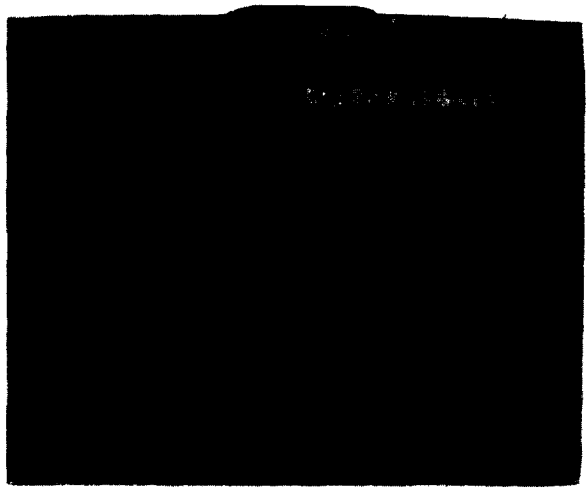
(4) 주제목 필드

이전의 필드들에서 선택된 항목들에 대한 모든 주제목 내용이 화면에 출력된다. 이 주제목 필드에서 항목들을 선택하게 되면 '부제목' 필드로 넘어가는 것이 아니고 바로 표본 필드로 넘어 가게 된다. 그 이외의 과정은 이전의 필드에서의 과정과 동일하다. 그림으로 설명하면 〈그림 15〉와 같다.

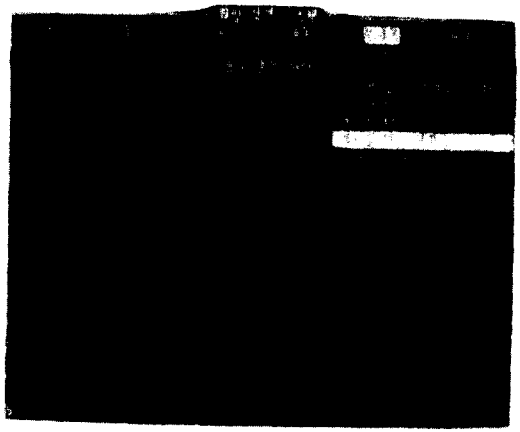
(5) 부제목 필드

주제목 필드에서 선택할 사항이 없거나 모를경우에 부제목 필드가 화면에 나타난다. 사용자가 필요한 내용을 선택 하게 되면 '표본' 필드 처리 과정으로 들어가게 된다. 만약 선택할 내용이 이 필드의 내용에 없다면 사용자는 아무것도 선택하지 않고 이 필드를 빠져 나갈 수 있다. 이 경우에는 찾고자 하는 내용이 없는 것으로 간주하고 다시 〈그림 10〉과 같이 정보 검색 시스템의 초

기 상태로 되돌아 가게 된다. 그 이외의 처리 과정은 이전의 필드들과 동일하다. 그림으로 설명하면 〈그림 16〉과 같다.



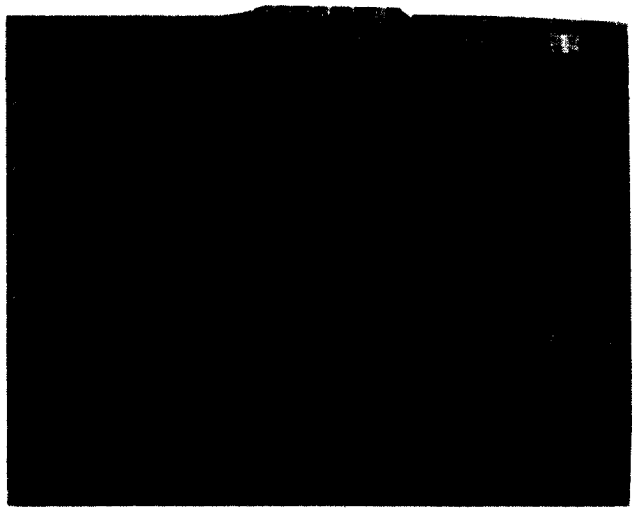
〈그림 15〉 순차적 접근에서의 주제목 필드



〈그림 16〉 순차적 접근에서의 부제목 필드

(6) 표본 필드

표본 필드의 기능은 현재까지의 선택의 완료를 나타내어 준다. 주제목 또는 부제목에서 사용자가 원하는 내용을 선택하면 시스템에서 자동적으로 표본 필드를 선택하고, 지금까지 선택되어진 키 화일에 대한 표본 화일의 모든 내용을 화면에 보여준다. 즉, 주 화일에서 갖고 있는 모든 인덱스에 대한 각각의 화일들의 레코드 값을 화면에 보여준다. 또한, 화일 처리 이전과 같이 모든 데이타를 보여주는데, 이전의 비정형적인 형태가 아닌 정형된 형태의 데이타로 보여주는 것이다. 즉, 표본 필드에서는 이전의 필드에서 선택되어진 항목들을 표본 필드 처리 과정에서 기억하고 있다가 표본 필드의 형태에 맞게 다시 재조정하게 된다. 모든 표본의 내용들을 화면에 출력 시킨뒤 다시 이 시스템의 초기 상태 〈그림 10〉으로 되돌아가게 된다. 그림으로 설명하면 〈그림 17〉과 같다.



〈그림 17〉 순차적 접근에서의 표본 필드

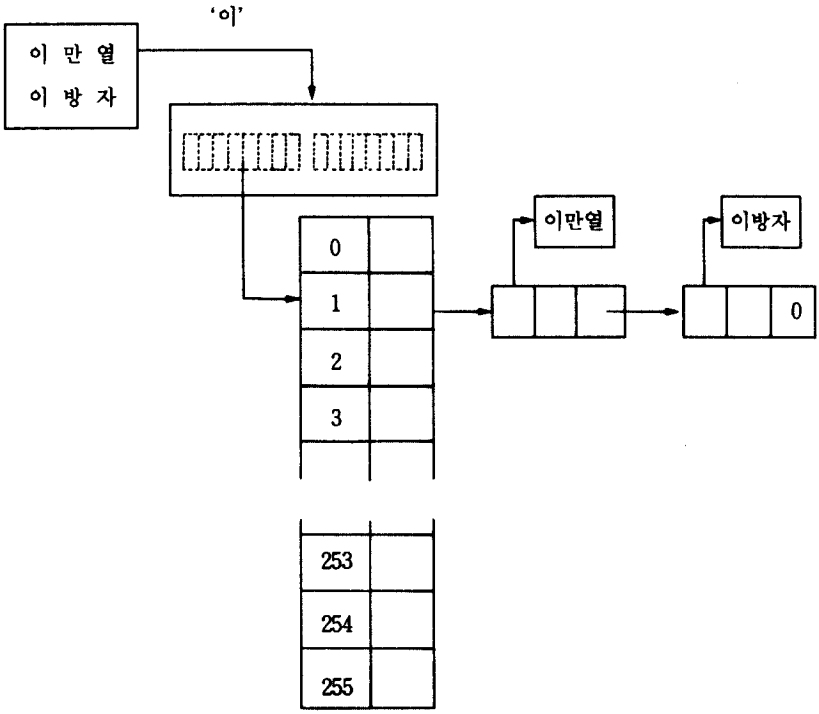
4. 3. 2 질의(Query)적 접근 방법

순차적 접근 방법은 때에 따라서 지루한 느낌을 주는 면이 있다. 순차적 접근은 사용자가 원하는 내용을 잘 알지 못하고 시스템에서 주어지는 순서를 통한 접근 방식이므로 직접 원하는 내용에 대한 검색만을 할 수 없다. 질의적 접근 방법은 사용자가 찾고자 하는 내용에 대해서 확실성이 있을 때 직접 그 내용을 해당되는 필드에서 key-in함으로써 빠른 검색 시간을 지원할 수 있는 방법이다. 질의적 접근 방법의 필드로는 출처, 저자, 주제목, 부제목, 표본 필드가 있다. 출처, 저자, 주제목, 부제목 필드 중 사용자가 원하는 필드를 선택한 후 시스템에서 제공하는 공란에 직접 key-in을 한다. 시스템에서 key-in한 내용을 그 필드를 지원하는 화일을 직접 검색하므로써 빠른 접근이 가능하다. 질의적 접근에서의 key-in한 내용과 각각의 필드를 위한 화일들과의 대응(matching)은 두 가지로 방법으로 나뉜다.

(1) 해싱(Hashing)을 이용한 방법

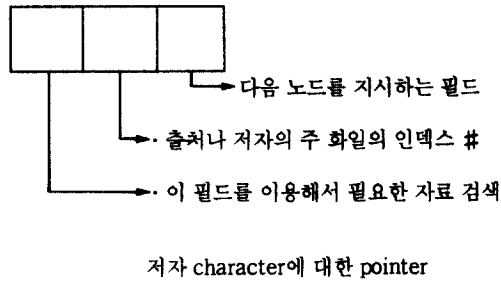
출처와 저자 필드에서는 해싱 기법을 이용해서 처리한다. 출처와 저자의 내용의 길이는 그다지 길지 않고, 또한 key-in한 내용의 크기와 화일에 있는 내용의 크기가 같으므로 해쉬 테이블을 통한 정보 검색이 가능하다.

한글이 2 바이트임을 착안하여 출처와 저자 필드인 경우 하나의 단어(출처 필드에서는 출처 이름, 저자 필드에서는 저자명)에서 첫번째 글자중 첫번째 바이트를 0~255 (총 8비트 이므로)까지 매핑(mapping)시키는 해쉬 함수(hash function)를 구성한다. 여러 단어들이 해쉬 테이블에서 같은 번지에 놓여서 오버플로우(overflow)가 발생했을 때는 오버플로우 처리(overflow handling)로써 체인(chaining)기법을 사용한다[15] 〈그림 18〉.



〈그림 18〉 해쉬 테이블 구성

각 노드는 〈그림 19〉와 같이 구성된다.

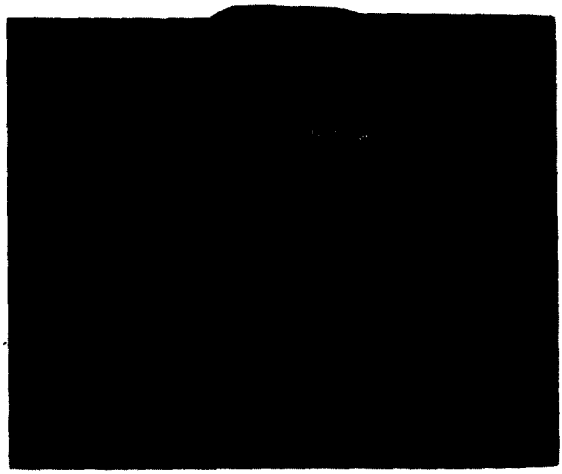


〈그림 19〉 해쉬 테이블의 각 노드 구성

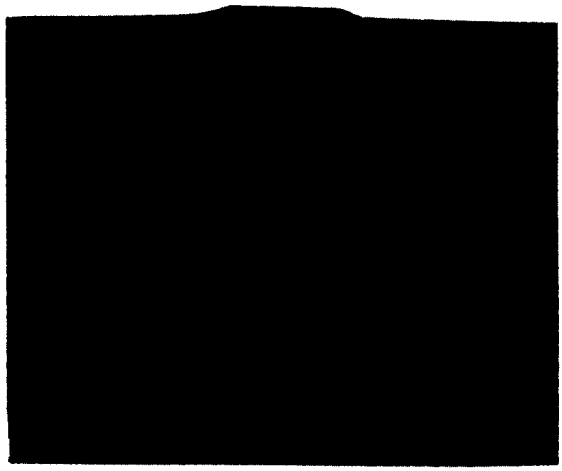
또한 key-in한 것과 대응을 시키기 위해서는 해쉬 테이블에 저장할 때나, key-in한 내용 모두를 태그와 공백을 무시한 상태로 저장하여 둔다. 그러므로 태그와 공백에 의한 대응의 문제점은 해결된다.

(2) 선형 대응 방법(Linear Matching Method)

주제목과 부제목에 대한 대응 방법은 key-in한 것과 주제목, 부제목 화일에 있는 내용을 태그와 공백을 무시한 상태로 Straightforward Algorithm을 이용한 선형 방법으로 대응시켜 찾는다[16]. 해쉬 테이블로 구성하기에 고란한 이유는 주제목, 부제목의 레코드 값이 일정하지 않고 또한 그 값이 크기 때문에 해쉬 테이블 구성시 많은 메모리 낭비를 초래한다. 이런 문제점 때문에 직접 주제목, 부제목 화일을 탐색하며 대응되는 모든 화일에 대한 표본을 순차적으로 접근할 때 표본 필드에서 처리 하듯이 화면에 출력한다. 그 이외의 사항은 순차적 접근 방법과 동일하다. 예는 주제목에서 질의적 으로 검색할 때의 과정을 〈그림 20〉 과 〈그림 21〉에서 보인다. 그 이외의 필드에서의 display되는 상태는 주제목과 동일하므로 생략하도록 한다.



〈그림 20〉 질의적 접근에서의 주제목 필드



〈그림 21〉 주제목 필드에서의 표본 내용

5. 수행 결과 및 결과 분석

5. 1 시스템

지금까지 300만 말뭉치속에 있는 여러 가지의 정보표시에 대한 정보표시 오류 검출 및 수정, 화일 처리, 정보검색 시스템에 관하여 기술하였다. 이를 구현하기 위한 프로그래밍 언어로는 Turbo C (Ver 2. 0)를 사용하였으며 조합형 한글 카드가 들어 있는 IBM PC-AT 호환 기종인 PRO-3000을 사용하여 구현하였다. 이때 입력 문서나 구현시에는 2-byte 조합형(상용 조합형)코드의 한글 코드를 사용하였다.

5. 2 수행 결과 및 분석

입력 문서로 5개의 표본 화일을 임의로 선정하여 정보표시 오류 검출 및 수정과 화일 처리 과정을 부록에 실었다.

5. 2. 1 정보 표시 오류 검출과 수정에 관한 결과 분석

현재 구축되어 있는 300만 말뭉치중 10만 어절(25개의 표본 화일)의 정보 표시 오류 검출 및 수정에 대한 처리 과정을 분석해 본 결과 총 515 개의 정보 표시 오류가 검출되었다. 이 중 451 개의 오류를 수정함으로써 약 87% 의 수정 성공율을 보였다. 나머지 64 개의 수정이 안된 정보 표시 오류의 형태를 살펴 보면 다음과 같다.

(1) 앞 또는 뒷 태그가 삭제된 오류 (25%)

태그의 어느 한 쪽이 삭제되었을 경우, 앞 태그를 기준으로 해서 처리하므로 앞 태그 삭제시에는 수정이 불가능하고, 또한 뒷 태그 삭제시에도 복합 명사의 띄어쓰기 원칙에 의한 뒷 태그 삽입에 대한 오류 수정 문제가 발생했다.

(예) 〈ㅂㅁ〉호암갤러리 〈/ㅇ〉〈ㅂㅁ〉
==〉〈ㅂㅁ〉호암갤러리〈ㅇ〉〈/ㅇ〉〈/ㅂㅁ〉

〈ㅎ〉나우 정밀과 〈ㅎ〉전자〈/ㅎ〉부품
==〉〈ㅎ〉나우 정밀과〈/ㅎ〉〈ㅎ〉전자〈/ㅎ〉부품

(2) 서로 다른 형태로 pair matching을 이룬 경우의 오류 (15%).

앞 태그를 기준으로 해서 pair matching을 시키므로 앞 태그를 정확히 기입하지 않았을 경우에 오류 수정이 불가능했다.

(예) 〈ㅎ〉티브이〈/ㅇ〉 ==〉 〈ㅎ〉티브이〈/ㅎ〉

(3) 태그 표시중 앞 또는 뒤가 잘못 쓰였거나 삭제된 오류(21%)

태그 표시('〈', '〉')중 어느 한 쪽이 다른 기호로 쓰여지고 또한 pair matching이 이루워지지 않았을 경우 데이타로 간주함으로써 생긴 오류이다.

(예) 세워질 수 있는 〈ㅎ〉의미적 하의 관계
==〉세워질 수 있는 (ㅎ)의미적 하의 관계

(4) 유형 99를 생성시키는 오류(39%)

표본과 표본을 분류시킬 수 있는 키 태그들(특히 유형 태그)이 생략되었을 경우, 이런 표본들을 모아두기 위해 순서에 입각해서 새로운 유형 99를 삽입함으로써 분류할 수 없는 화일들을 모아서 새로운 표본 화일을 생성한다. 이럴 경우, 편집자 입장에서의 오류가 발생한다.

(예) 〈ㅂㅁ〉(1)〈ㅎㅎ〉주시경〈/ㅎㅎ〉
==〉〈ㅍㅂ〉〈ㅇㅎ〉99〈/ㅇㅎ〉〈ㅊㅊ〉〈/ㅊㅊ〉〈ㅈㅁ〉〈,ㅈㅁ〉〈ㅂㅁ〉
(1)〈ㅎㅎ〉주시경〈/ㅎㅎ〉

(5) semantic상의 오류와 태그 표시의 앞과 뒤를 다른 표시(예를들자면, '('나')')로 잘못 쓴 경우는 오류 검출이 되지 않으므로 수정이 불가능하다.

(예) 〈ㅇ〉년〈/ㅇ〉, 어떠한 (ㅎ)명사(/ㅎ) 분포에 대한

위와 같이 오류 수정이 불가능한 이유는 편집 과정에서 정확한 tagging table에 의한 규칙을 지키지 않았다는 것과 이로 인해 많은 표본 자료들을 통한 heuristic한 방법에서 처리함으로 인해 발생된 것이다. 결론적으로 본 논문에서 제시한 tagging model이 lexical database나 전자 사전 구축시에도 큰 수정 없이 확장성을 갖기 위해서는 위에서 설명한 정확한 tagging rule에 의한 편집 작업이 필요하다. 정확한 규칙하에서의 tagging table이 존재한다면 본 논문의 tagging model은 정보표시의 오류 처리와 화일 처리를 general하게 수행시킬 수 있다.

5. 2. 2 화일 처리와 정보 검색 시스템에 대한 결과 분석

본 논문에서 제안한 화일 구조는 3. 3절에서 언급 된 것처럼 random access가 가능하고 레코드 길이가 가변적일 수 있다. 이 구조에서의 화일 처리 속도는 레코드들의 길이에 좌우된다. 레코드 길이를 작게 잡으면 화일 접근 횟수가 많아지므로 처리 속도가 늦어지고, 반면에 레코드 길이를 크게 잡으면 화일 접근 횟수가 줄어들므로 처리 속도는 빨라지나 메모리 낭비가 심해진다.

본 논문에서는 디스크 헤드가 한번 움직일 때 읽어들이는 데이타 양(128 bytes)에 기준을 두어 화일들의 레코드 크기를 결정하였다. 비록 레코드 크기가 heuristic하게 결정되었으나, 대개의 경우 1~2번 정도의 접근이면 원하는 데이타를 검색할 수 있었다.

본 논문의 정보검색 시스템의 질의적 접근은 화일에 대한 평균적 접근 횟수인 N/2 (N : 레코드 갯수)인데 비해, 해쉬을 이용한 경우에는 평균적으로 2~3번 정도의 접근이면 검색이 가능했다.

6. 맺음말

본 논문은 정보표시를 이용하여 오류를 검출하고 수정과 동시에, 화일 처리 및 표본 데이타에 대한 정보 검색 시스템에 관해 기술하였다.

본 연구의 필요성은 첫째, 현재 한국어 사전 편찬실에서 계속적으로 corpus화하여 구축하고 있는 표본 자료를 필요에 따라 사용자에게 보다 빠르고 편리한 검색 결과를 제공해 주는 것이고 둘째, 표본 자료를 분류 및 저장의 측면에서 효율적으로 관리해 주는 것이고 셋째로는, 정보 표시의 목적인 텍스트외의 정보를 말뭉치에서 정확하게 유지시키는데 있다.

그러나, 연구를 진행함에 있어 정보 표시 사용에 대한 문제점이 나타났다. 정보 표시를 사용하는데 있어 부록의 〈표 2〉와 같이 미리 약속된 규칙하에서 사용해야 하는데도 불구하고 그렇지 못한 경우들이 많았다. 정확한 규칙 안에서 표본 편집 작업을 한다면 정보표시 오류 검출 및 수정에 대한 정확성을 더욱 높힐 수 있을 것이다.

본 연구에 이어 앞으로 더 많은 연구의 필요성을 느낀 다음 몇가지 사항들을 나열한다.

1. Tagging Model에 근거한 각종 텍스트 정보들을 처리할 Utility

(1) 태그 편집기(Tagging Editor)

앞서 설명한 정보 표시 오류의 검출과 수정의 정도를 더욱 높이기 위해서

표본 선정 후 그 표본을 입력할 때 태그 편집기 내에서 작업을 한다면 키 태그에 대한 오류의 수정 문제점이 다소 감소될 것이다. 즉, 편집기에 미리 키 태그들이 입력되어 있고 표본 편집자는 알맞은 곳에 그 내용을 입력시킨다. 또한, 각 표본 유형 마다 태그 사용 규칙을 세우고 편집기를 구성할때 그 유형별로 선택할 수 있는 메뉴를 만든다. 그러므로 입력할 유형별로 키 태그들의 범위가 달라지고 화일 처리를 할 때도 각 유형에 따라 과정이 달라지게 된다. 이런 방법을 이용하면 본 논문에서의 어느정도 순서에 입각한 오류 수정의 단점이 개선 되어질 것이다. 그리고, 키 태그 내용과 표본의 내용를 구분하기 위해 다음과 같은 정보 표시를 삽입한다. 키 태그의 시작을 알리는 〈ㅅㅈ〉~〈/ㅅㅈ〉과 표본 내용의 시작을 알리는 〈ㅈㄹ〉~〈/ㅈㄹ〉표시를 삽입하면 화일 처리를 하기가 한결 편리하고 확장시키기가 용이할 것이다. 예를 들면 다음과 같다.

〈ㅍㅂ〉
〈ㅅㅈ〉〈ㅇㅎ〉64〈/ㅇㅎ〉〈ㅊㅊ〉국정교과서〈/ㅊㅊ〉〈ㅈㅁ〉나의 〈ㅎ〉생활
〈/ㅎ〉〈/ㅈㅁ〉〈ㅈㅈ〉문교부〈/ㅈㅈ〉〈ㄴㅇ〉9001〈/ㄴㅇ〉〈ㅉ〉197-300〈/ㅉ〉
{〈ㅂㅁ〉〈/ㅂㅁ〉}{〈ㅇㅇ〉〈/ㅇㅇ〉}〈/ㅅㅈ〉
〈ㅈㄹ〉{ ----표본자료들 ---- }〈/ㅈㄹ〉
〈/ㅍㅂ〉

(2) 대화식의 태그 수정기 (Interactive Tagging Corrector)

2 장에서 이미 설명한 태그 오류 검출 및 수정 단계는 오류 수정을 위해 어느정도 순서를 부여해 놓았다. 이 단계의 문제점은 태그 오류에 대한 수정이 표본 화일 선정과 편집을 하는 입장에서 볼 때는 자신의 의도와는 다른점이 발견될 경우도 있다. 이런 문제점은 순서가 임의적인 자료에 대해서는 수정이 불가능하기 때문에 발생되는 경우이다. 해결 방안은 (1)에서 구상한 태그 편집기 상에서 현재 오류 검출은 거의 100%의 확률을 가지고 있으므로 오류가 검출될 때 그 부분을 편집자에게 알려주어 알맞은 형태의 태그로 바꾸어 주는 기능을 하는 대화식을 통한 태그 수정기를 구상해 본다. 현재 이런 방식을 취하는 예로는 Turbo C를 들 수 있다. 단점은 오류가 상당수 발생할 경우 편집

자가 일일이 다 고쳐 주어야 한다는 것이다. 이럴 경우는 default로 2장에서 이미 구현한 태그 오류 검출 및 수정기에서 처리를 하고 오류 발생이 적을 때는 직접 편집자가 오류를 수정하는 선택의 방법을 이용하면 더욱 효율적인 오류 수정을 할 수 있을 것이다.

(3) 표본 자료에 keyword index를 붙여주는 utility에 관한 연구

표본 자료를 읽어 들이면서 단어들의 빈도수(frequency)에 의한 keyword들을 결정하고 이들을 쉽게 access할 수 있는 index를 만드는 연구가 필요하다.

2. 한글 문서에 대한 Query Processing에 관한 연구

본 검색 시스템은 고정된 형태의 query만을 제공하기 때문에 사용자와 대화 형식으로 자료를 검색하지 못한다는 단점이 있다. 따라서, 본 논문에서 사용된 고정된 query외에 SQL이나, QBE(Query By Example)와 같은 query language를 사용자에게 제공하는 것이 필요하다. 이 것은 부가적으로 Query Compiler를 필요로 할 것이며, Optimization 기법에 대한 연구 또한 필요할 것이다.

위의 두가지 연구가 추가 되어진다면 앞으로 전자사전 개발에 있어서 입력된 말뭉치에 대한 문서 관리 측면 과 사용자가 필요로 하는 자료에 대한 검색 측면의 확실성을 부여하게 될 것이다. 또한, 전자 사전 개발에 있어서의 본 논문의 또 하나의 문제점은 구현시 사용한 시스템의 주 메모리(main memory) 용량의 한계성과 한글 코드 사용에 대한 문제점이 발생되었다. 본 논문에서 사용한 IBM PC-AT에서 주 메모리 문제점이 검색 시스템 구현시 발생했고 한글 코드를 2 바이트 조합형 코드를 사용함에 있어 대형 시스템에서 지원하는 한글 코드와의 통신에 관한 문제가 발생되었다. 해결 방안은 대형 시스템에서 지원하는 한글코드를 사용해서 전자 사전 개발에 필요한 과정의 구현이 이루어 져야 할 것이다.

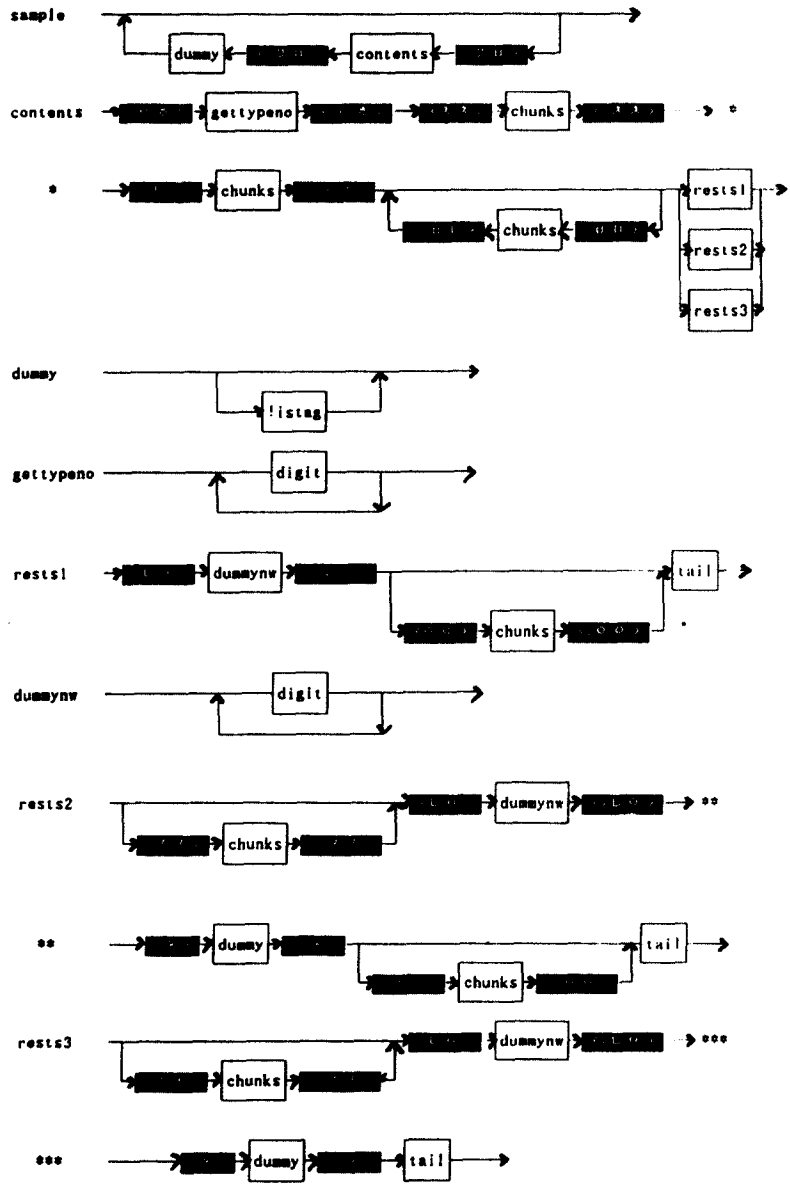
부록 1 〈표 2〉 Tagging Table

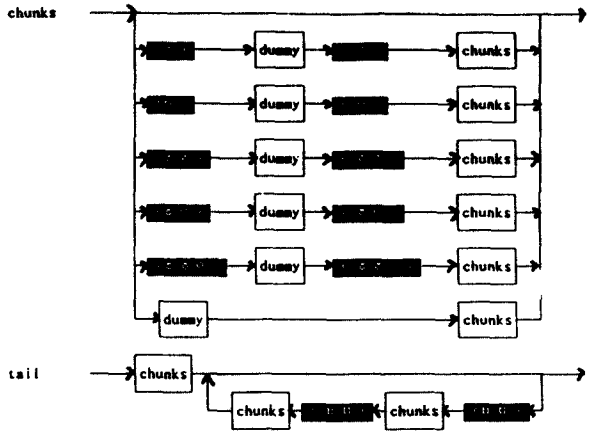
Tag의 종류 / 표본의 유형		유형	출처	제목		저자	연월	쪽	요약	한자		외래어		한글·한자 외래어
				주제목	부제목					only	한글	only	한글	
		〈ㅇㅎ〉	〈ㅊㅊ〉	〈ㅈㅁ〉	〈ㅂㅁ〉	〈ㅈㅈ〉	〈ㄴㅇ〉	〈ㅉ〉	〈ㅇㅇ〉	〈ㅎ〉	〈ㅎㅎ〉	〈ㅇ〉	〈ㅎㅇ〉	〈ㅎㅎㅇ〉
1) 신문	정치	11	(예) 동아일보 매일경제 신문	^		N/A	yr,mon	N/A	^	^		^		^
	경제	12												
	사회	13												
	문화	14												
	오락 및 스포츠	15												
2) 잡지	여성 및 가정	21	(예) 월간조선	^		^	yr,mon	(예) 181-189	^	^		^		^
	정치 및 사회	22												
	문학	23												
	취미 및 오락	24												
3) 소설 및 수필	수필	31	책명 (부제목 포함)	^ [title]		^	yr,mon	^	N/A	^		^		^
	단행본	32												
	역사소설	33		[title]										
4) 취미 및 교양	수공예 및 취미	41	책명 (부제목 포함)	[title]		^	yr,mon	^	N/A	^		^		^
	생활정보	42		[title]										
5) 수기 및 전기	위인전, 전기 및 수기	51	책명 (부제목 포함)	[title]		^	yr,mon	^	N/A	^		^		^
	실화집	52		[title]										
6) 교과서	초	61	책명 (부제목 포함)	^		^	yr,mon	^	N/A	^		^		^
	중	62												
	고	63												
	대	64												

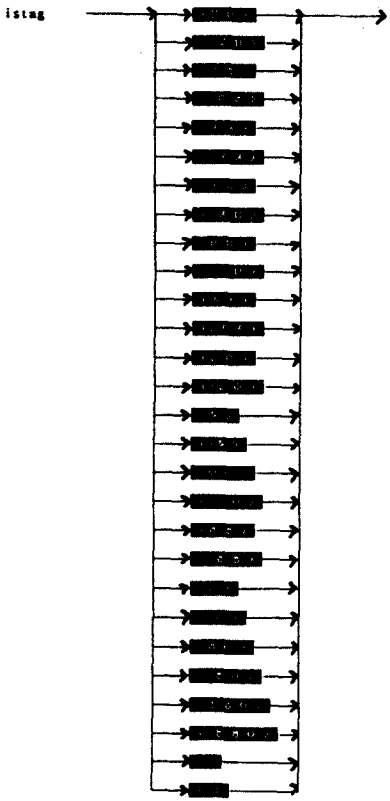
* reference
1. 시작과 끝 tag를 구별
(예) 〈ㅍㅂ〉-〈/ㅍㅂ〉

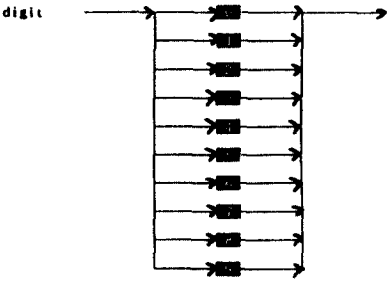
2. N/A → Not Applied
3. '^' → '있음' 표시

부록 2 Tagging Automata의 Syntax Diagram




chunks
dummy
chunks
dummy
chunks
dummy
chunks
dummy
chunks
dummy
chunks
dummy
chunks
tail
chunks
chunks
chunks


istag


digit

참 고 문 헌

(1) 정영미. 1988. "정보검색론". 정음사.
(2) 최윤철, 송만석. 1990. "한국어 전자사전 개발의 현황과 과제". 한국정보과학회 국어정보처리 춘계워크샵 학술 발표논문집.
(3) 홍종화. 1986. "초성 테이블을 이용한 한글 문헌 정보 검색 시스템의 설계 및 구현". 한양대 산업대학원 석사학위 논문.
(4) 이상섭, 남기심외. 1988. 사전 편찬학 연구, 제 1집, 제 2집. 탑 출판사.
(5) 이상섭. 1989. "뭉치언어학 : 사전 편찬의 필수적 개념". 한글 및 한국어 정보처리 학술 발표 논문집.
(6) 정찬섭. 1989. "한국어 어휘 뭉치의 표본 선정 기준". 한글 및 한국어 정보처리 학술 발표 논문집.
(7) 조재수. 1984. 국어 사전 편찬론. 과학사.
(8) Darrell R. Raymond and Frank WM. Tompa. 1988. "Hypertext and the OXFORD ENGLISH DICTIONARY". Comm. of ACM. Vol. 31.
(9) Gaston H. Gonnet. 1987. "Examples of PAT applied to the Oxford English Dictionary". UW centre, Canada, July.
(10) Gaston H. Gonnet. 1987. "Efficient searching of Text and Pictures Extended Abstract". UW centre, Canada.
(11) Header Fawcett. 1988. "Using Tagged Text to Support Online Views". UW centre, Canada.
(12) Header fawcett. 1988. "Adopting SGML : The Implications for writers". UW centre, Canada.
(13) Salton, G. and M. j. McGill. 1983. "Introduction to Modern Information Retrieval". McGraw-Hill, New York.
(14) 김영택. 1990. "한국어 정보 처리의 현안 문제와 향후 과제". 제 2회 한국정보과학회 춘계워크샵 , 국어 정보 처리.
(15) 이석호. 1985. "화일 처리론". 정익사.
(16) G. DE V. SMIT. 1982. "A Comparison of Three String Matching Algorithm", SOFTWARE PRACTICE AND EXPERIENCE. VOL. 12, 57-66.

☐ ABSTRACT

A Design and Implementation of Tagging Model to Retrieve Information Regarding Textbase

Park, Jae wan

Human being can classify and memorize the necessary knowledge through many types of data information since his intelligence is very excellent in the seveal aspects. Since man's memory, however, is not unlimited, data remembered come to be lost as time goes by. By this reason, in the field of information science, the lexical database which is containing all of the information for a Korean information processing is being developed by recognizing the importance of language information and the necessity of Korean Electronic Dictionary Development using the recent computer technology has shown its head.

Lexicographical Center in Yonsei University constructed three million corpus selecting sample data for the dictionary construction. The corpus was tagged to reserve all information except the text within itself because it can't classify and memorize the sample data naturally like human beings. Constructing corpus manually can provoke many kinds of errors. Among these errors, tagging error is a very serious problem.

This paper is a study on the design and implemention of tagging automata for the error detection and correction as a link of Korean Electronic Dictionary Development, a study on the file processing for classifying and storing the sample data efficiently, and a study on the Tagging Model to retrieve efficiently information regarding textbase using the file-processed database.