색인어 추출과 역색인 구축, 검색 결과 추출 그리고 Semantic Indexing

모란소프트 주식회사

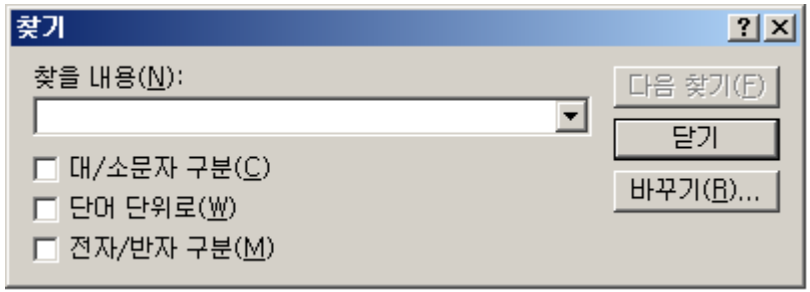
조 영 환

Agenda

- 색인의 개념과 색인어 추출 방식
  - 색인이란, 색인의 개념
  - 형태소 분석결과에서 색인어 추출하는 과정
- 검색 알고리즘
  - 추출된 색인어로부터 역색인의 구축 과정
  - 쿼리로부터 역색인을 참조하여 검색 결과를 추출하는 과정
- Semantic Indexing
  - Semantic Search에 대한 개요
  - 개념어 색인, 쿼리 색인, 은유적 색인, 감성 색인, 평가 표현 색인 등 소개

검색과 색인

- 검색 :
  - 대량의 문서 집합에서 "특정" 키워드가 포함된 문서를 찾아내는 것



  - 조건 : 모두 찾아 내어야 함, 더 적합것이 먼저 나와야 함

- 검색의 성능평가 요소
  - Recall : 모두 나와야 하는 것 중에 검색 결과에 포함된 것의 비율
  - Precision : 검색결과에 포함된 것 중에 제대로 나온 것의 비율

- 검색에 있어서 색인의 개념
  - 실시간 String Compare(== $ grep)로는 search 속도가 느리므로 속도를 빠르게 하기 위해서 무엇인가 장치 (역색인)를 구축하는 것

색인이란?

- 색인의 정의
  - 빠른 검색을 위하여 문서에서 검색의 될 만한 것(Lexicon)들의 존재여부와 위치(positing)를 미리 추출하여 빠른 탐색 자료구조를 구축하여 놓는 것
  - Collection내의 단어의 개수가 n이라고 하고 전체 Collection 에서 나타난 단어의 개수가 m이라고 한다면, Linear Search의 경우 O(N), Indexed Search의 경우 O(log M)이다. N >> M

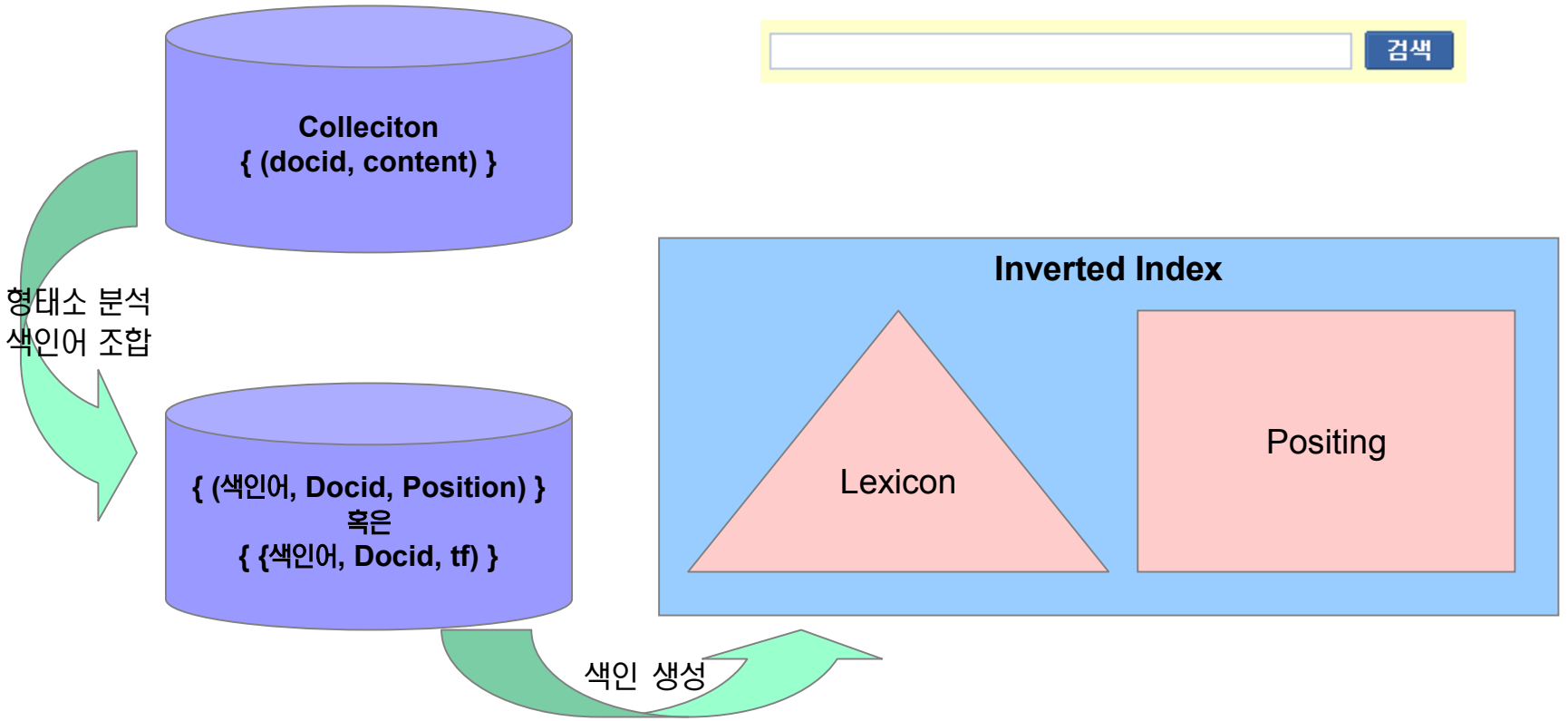
- 100만건 Collection과 관련된 숫자
  - 파일의 크기 : 3GB, 3K/doc
  - 색인어 총 추출 개수 : 3억개, { (Kwd, docid, tokenid) } 6.5 GB
  - 색인어-DocID UNIQ 개수 : 180M (179,759,835개) { (Kwd, docid) }
  - 색인어 UNIQ 개수 : 700만개, { (Kwd) } 151 MB
  - 형태소 분석 시간 : 80 min
  - 3억개 소팅시간 : 60 min

역색인 구축의 과정

- 역색인
  - 어떠한 키워드가 주어졌을때에, 어떠한 문서에서 나타났는지를 알려주는 자료구조

- 역색인 구축의 과정
  1. 원문에서 색인어를 추출한다. { (docid, tokenid, kwd) }
  2. 원문에서 각 문서당 색인어수를 센다. { (docid, kwd count) }
  3. 키워드 순서로 정렬한다. { (kwd, docid, tokenid) }
  4. 키워드당 역색인 벡터를 만든다. (Lexicon, Posting)
  5. 역색인 벡터를 압축한다.
  6. 키워드당 검색 순위를 미리 만든다.
  7. Lexicon을 Hashing, Btree, TRIE등의 자료구조로 색인한다.

색인의 과정


Colleciton
{ (docid, content) }
검색
형태소 분석
색인어 조합
Inverted Index
Lexicon
Positing
{ (색인어, Docid, Position) }
혹은
{ {색인어, Docid, tf) }
색인 생성

색인의 생성 - 렉시콘과 포스팅화일

문서 번호	문서 내용
1	Pease porridge hot, pease porridge cold
2	Pease porridge in the pot
3	Nine days old
4	Some like it hot, some like it cold
5	Some like it in the pot
6	Nine days old

(a) 근접 연산을 지원하지 않을 경우 (b) 근접 연산을 지원할 경우

색인어 추출 방식

- 색인어 추출이란?
  - 문서집합 (collection)에서 검색의 대상이 될 만한 것을 미리 추출하는 것
  - Prerequisite : 검색의 대상이 될 만한 것
    - 방식 1 : Pat tree 형식으로 Document을 그대로 코딩
    - 방식 2 : Bigram 방식으로 두글자씩을 색인어로 추출
    - 방식 3 : 형태소 분석에 의한 색인어 추출
      - 방식 3-1 : 최소단위 색인어만 형태소 색인
      - 방식 3-2 : 복합명사와 용언의 복합단어 색인
      - 방식 3-3 : 기존의 쿼리와 비교하여 쿼리 색인
      - 방식 3-4 : 영화명 등 띄어써진 단어들도 포함하여 명칭 색인
    - 방식 4 : 토큰 단위로 색인어 추출

- 색인어 추출 순서
  1. 컴퓨터 프로그램에 의한 자동추출 (자동색인어 추출기 = 형태소 분석기 + 색인어 조합)
  2. 전문가에 의한 색인어 후통제

Bi-gram 색인

- 방식
  - 2글자씩을 겹쳐서 색인어로 추출하는 것
  - 예) 정보검색시스템을 : “정보”, “보검”, “검색”, “색시”, “시스”, “스템”, “템을”
- 장점 :
  - Recall 100%. 빠지는 것 없이 모두 다 찾아 낼 수 있음
  - 색인어의 크기가 일정하여 렉시콘 구성이 쉬움
- 단점 :
  - 틀린 것도 많이 포함되며, 검색 품질이 심각하게 저하됨
  - 렉시콘 크기가 커지고, 포스팅의 크기 역시 커짐
- 적용처 :
  - 형태소 분석의 오류로 인한 성능저하가 우려되는 경우
  - 문서의 건수가 적고, Recall이 최우선시 되는 경우

형태소 분석에 의한 색인어 추출

- 형태소의 정의
  - 의미를 가지고 있는 최소 문자열
  - 예) 정보검색시스템을 : 정보, 검색, 시스템, 을, 정보검색, 검색시스템, 정보검색시스템
- 형태소 색인
  - 색인 대상 품사를 정하고 있어야 함
- 복합단어 색인
  - 단어 조합 기준을 가지고 있어야 함
- 쿼리 색인
  - 쿼리 DB가 준비되어 있어야 함
- 명칭 색인
  - 명칭 DB가 준비되어 있어야 함
  - "장 폴 벨몽드", "007 죽느냐 사느냐", "사랑이 지나가면"

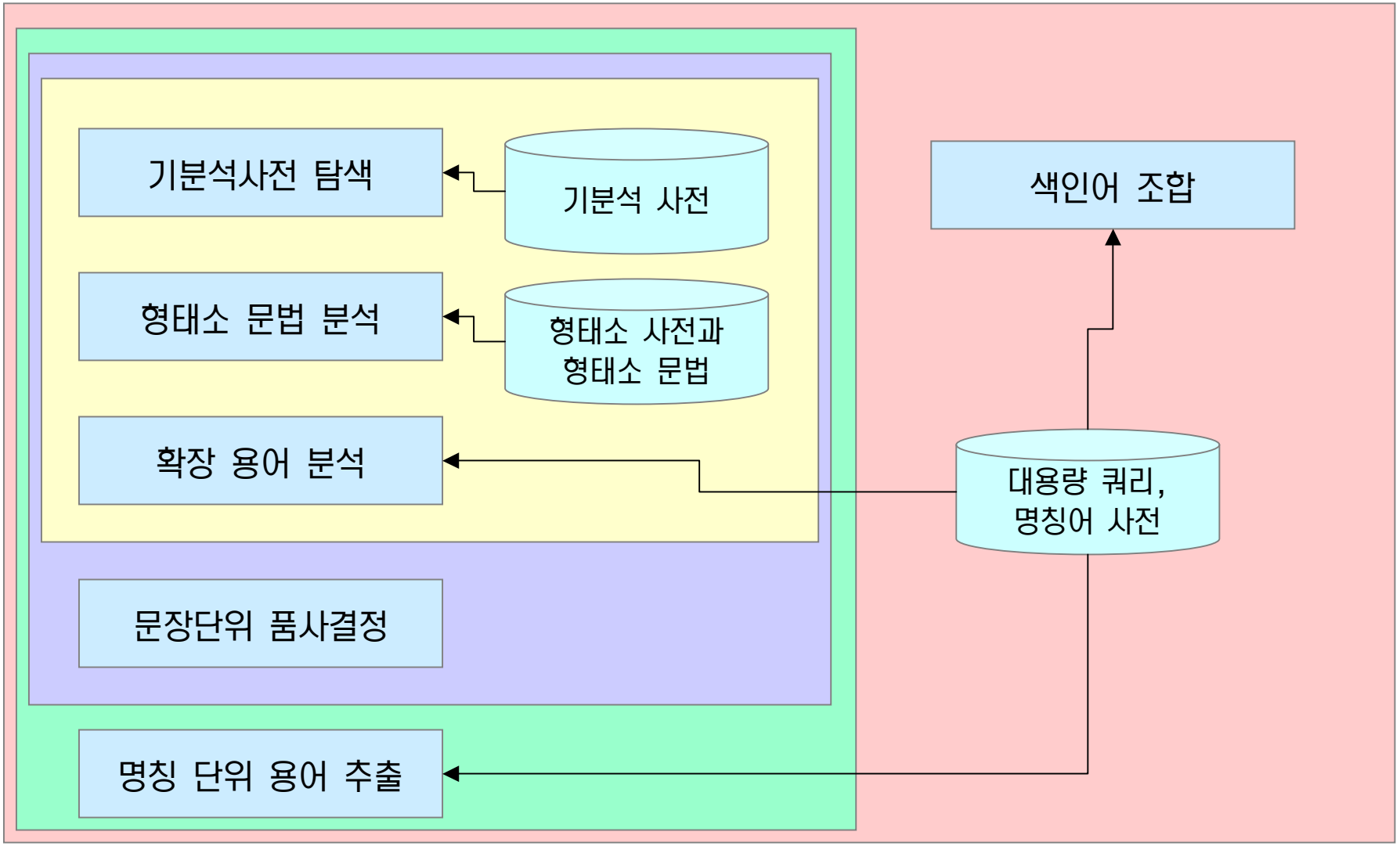
형태소의 품사 분류

대분류	소분류	세분류
체언	명사 nn	일반명사 nng
		고유명사 nnp
		의존명사 nnb
		명사추정 nnf
		복합명사 nnc
		접사형 nnx
	대명사 np	
	수사 nr	
	복합수사 nrc	
용언	동사 vv	
	형용사 va	
	보조용언 vx	긍정보조용언 vxp
		부정보조용언 vxn
	지정사 vc	긍정지정사 vcp
		부정지정사 vcn
수식언	관형사 mm	
	수관형사 mmn	
	부사 ma	일반부사 mag
		접속부사 maj

독립언	감탄사 ic	
관계언	격조사 jk	주격조사 jks
		보격조사 jkc
		관형격조사 jkg
		목적격조사 jko
		부사격조사 jkb
		호격조사 jkv
		인용격조사 jkq
	보조사 jx	
	접속조사 jc	
의존 형태	어미 e	선어말어미 ep
		종결어미 ef
		연결어미 ec
		명사형전성어미 etn
		관형형전성어미 etm
	접두사	체언접두사 xpn
	xp 접미사 xs	명사파생접미사 xsn
		동사파생접미사 xsv
		형용사파생접미사 xsa
		부사파생접미사 xsb
	어근 xr	

기호	마침표, 물음표, 느낌표	sf
	쉼표, 가운뎃점, 콜론, 빗금	sp
	따옴표, 괄호표, 줄표	ss
	줄임표	se
	붙임표 (물결, 숨김, 빠짐)	so
	외국어	sl
	한자	sh
	기타기호	sw
	숫자	sn

형태소 분석을 이용한 색인어 추출 방법



형태소 분석 결과와 색인어 조합

- 숫자 + 단위성 명사
  - 1997년에는 = 1997 + 년 + 에는 -> 1997, 1997년
- 명사 + 접미사
  - 확장성을 = 확장 + 성 + 을 -> 확장, 확장성
- 명사 + 용언파생접미사
  - 사랑하는 = 사랑 + 하 + 는 -> 사랑, 사랑하다
- 접두사 + 명사
  - 전대통령을 = 전 + 대통령 + 을 -> 대통령, 전대통령
- 용언어간 + "다"
  - 먹고서 = 먹 + 고 + 서 -> 먹다
- 용언어간 + 명사형 전성어미
  - 사라지기 = 사라지 + 기 -> 사라지다, 사라지기

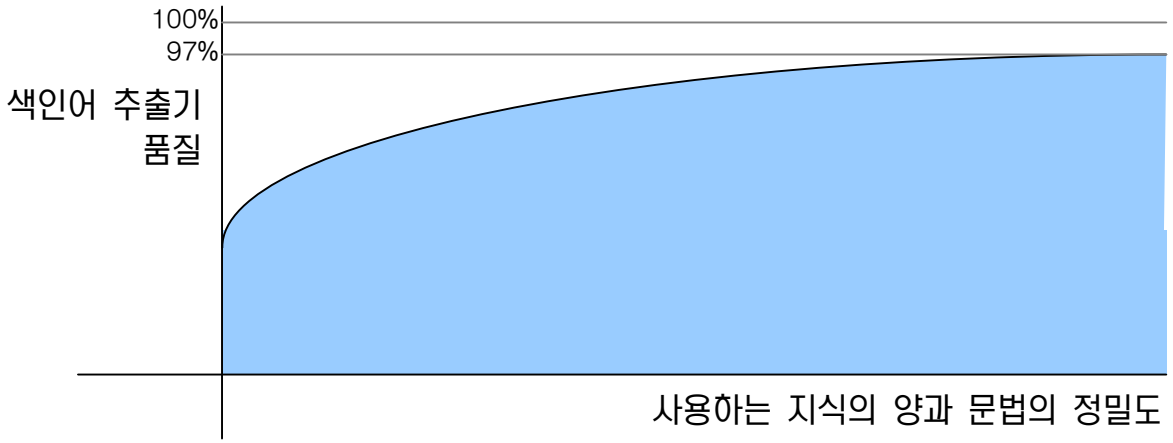
형태소 이상의 색인어 추출

- 명칭 DB와 쿼리 DB를 사용하는 것이 편리함

- 쿼리에는 명칭과 복합명사 등이 포함되어 있는 실제의 검색 키워드
- 입력 오류의 단어들이 많이 포함되어 있어서 쿼리로부터 적합한 단어를 선별하는 작업이 필요함
- QCTR - Query ClickThrough Rate와 Document Frequency를 이용하여 쿼리 선별작업

색인기와 검색 품질

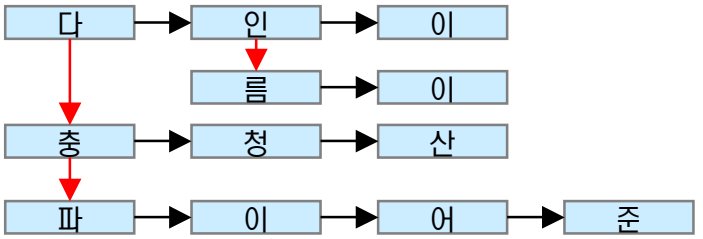
- 색인어 추출기가 좋아지면 검색 품질이 나아지는가?
  - 띄어쓰기 오류에 대한 견고성 : 더 많은 검색 결과를 추출
  - 오분석 결과의 감소 : 틀린 검색 결과 추출을 억제
  - 쿼리 반영 복합명사 추출 : 색인어 갯수의 감소로 시스템 최적화
  - 명칭 색인어 추출 : 이해하기 어려운 검색 결과의 최소화, 명칭어 최적 검색
  - 용언의 원형복원 : 자연어 질의가 가능
- 그러나 완벽한 색인어 추출기는 없다.



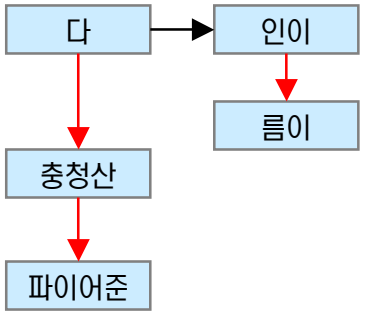
형태소 분석과 색인어 추출에 적합한 색인구조 : TRIE

```
$ mktrie2 utf8.txt utf16.trie2
        insert([다인이],[1]     <- [name])
        insert([다름이],[6]     <- [name])
        insert([파이어준],[11]  <- [name])
        insert([충청산],[16]    <- [poi])
        DATA LOAING DONE!!
        TRIE OPTIMIZE DONE!!
        Y-ARRAY Count DONE!!
        NODE NUMBERING DONE!!
        NODE ADDRESSING DONE!!
        Y-ARRAY GENERATION DONE!!


        ->[다|0|0]->[름이|6|1]
        ->{ ## }->[인이|1|0]
        ->[충청산|16|0]
        ->[파이어준|11|0]

        Character TIRE Nodes = 12       String Trie NODES = 5
        TOTAL NODES = 5, XNODES = 4, YNODES = 1, YADDRESS = 1
        FIRST NODE AT : 262144
        DOWNLOAD TRIE DONE!!
```

글자별 TRIE



String TRIE



Ynext = int

Xnext = 1bit

형태소 분석과 색인어 추출에 적합한 색인구조 : TRIE-2

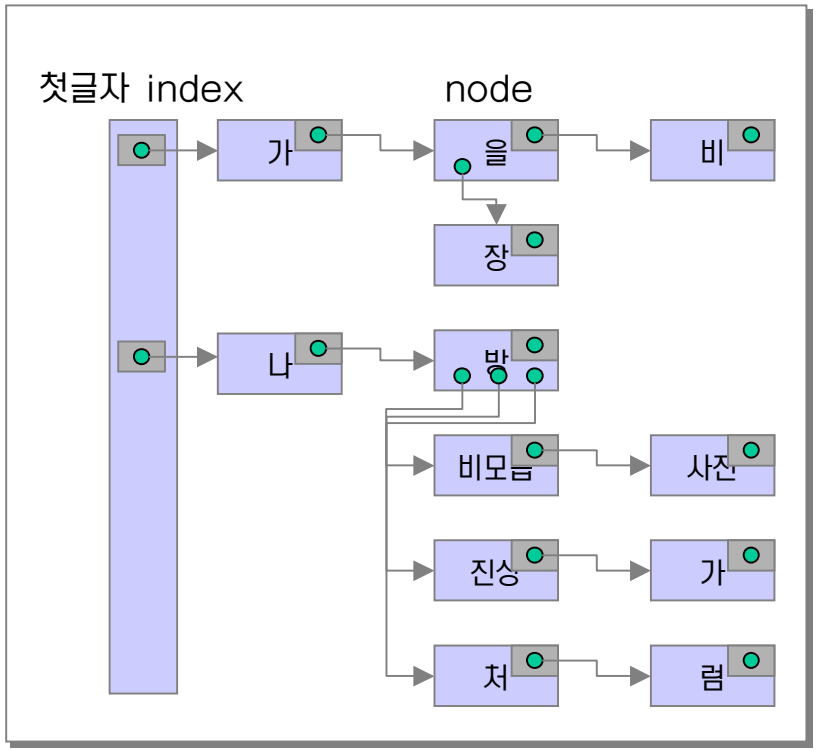
- 형태소 분석을 위해서 많은 자료가 투입되는데, 이것들의 효율적인 참조를 위한 탐색 알고리즘이 필요

```
struct  Trie_Index {
        unsigned char   *db;
        int         size;
        unsigned long TRIE_FIRST_CHAR_ADDR[256];
        };

struct  y_array_str {
        unsigned char   code;
        unsigned int    address;
};
```

TRIE를 컴파일하여 파일에 저장하고, 이 파일을 통째로 읽어들여서 빠른 Access가 가능함

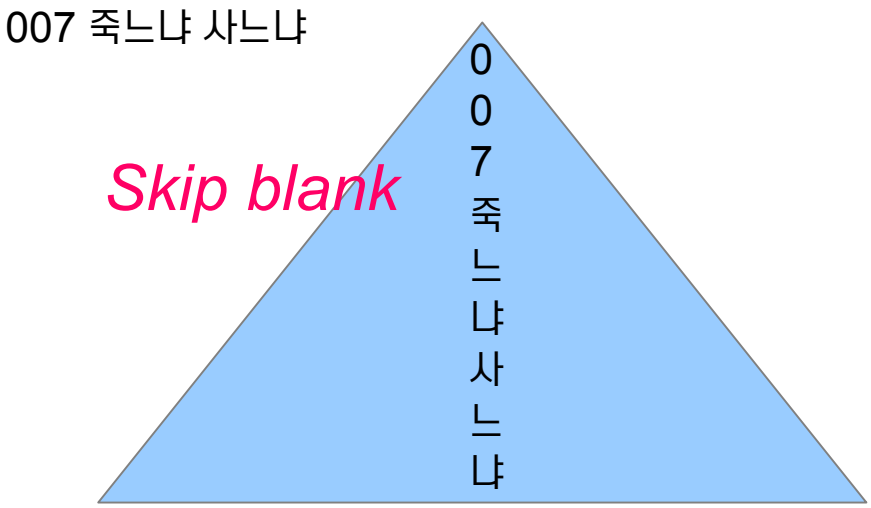
TRIE의 노드 분기에 binary search를 사용하여 공간을 절약하고 상대적으로 속도를 빠르게 함

첫글자 index와 Node



공백이 있는 문자열에 대한 TRIE matching

- 원문 : 007 죽느냐 사느냐
- TRIE 키워드 : 007죽느냐사느냐



검색 알고리즘

- 검색 알고리즘
  - 색인어에 대한 역색인 구축과 질의어에서 색인어를 추출하여 (모든) 색인어를 포함한 문서를 추출하고 가중치를 계산하는 절차
  - 역색인의 구성과 매우 밀접한 관계가 있음
- 검색 알고리즘의 구성
  - 추출된 색인어로부터 역색인의 구축 과정
  - 쿼리로부터 역색인을 참조하여 검색 결과를 추출하는 과정
- 한국어 검색 알고리즘의 특징
  - 정보검색의 기본 이론에는 모든 단어는 독립적인 것을 가정하고 있지만, 한국어의 경우에 단어의 구성방식이 – 복합명사, 활용어 등 – 존재하므로 모두 독립적인 것으로 가정하기에는 무리가 있음
  - 언어 일반적인 관점에서 복합명사의 존재를 부인하기 어려움
  - 해외 검색 엔진의 한계 : 단어를 Segmentation하는 방식을 기본으로 가정
    - 예) 정보검색시스템을 : 정보, 검색, 시스템
    - 정보검색시스템을 == 정보를 검색하지 못하는 시스템

색인 생성 == 역색인의 구성 = 렉시콘 + 포스팅

- 렉시콘과 포스팅을 어떠한 방식으로 구축하는 가
  - 주어진 장비의 한계 : 주기억장치의 크기
  - 주어진 Collection의 크기 : 분산 환경?
  - 문서의 변경 여부 : online update/insertion
  - 데이터 값 일부의 변경여부 : 가격비교 Site, Market placement

문서 번호	문서 내용
1	Pease porridge hot, pease porridge cold
2	Pease porridge in the pot
3	Nine days old
4	Some like it hot, some like it cold
5	Some like it in the pot
6	Nine days old

렉시콘 파일		포스팅 파일
cold	2	1, 4
days	2	3, 6
hot	2	1, 4
in	2	2, 5
it	2	4, 5
like	2	4, 5
nine	2	3, 6
old	2	3, 6
pease	2	1, 2
porridge	2	1, 2
pot	2	2, 5
some	2	4, 5
the	2	2, 5

(a) 근접 연산을 지원하지 않을 경우

렉시콘 파일		포스팅 파일
cold	2	(1; 6) (4; 8)
days	2	(3; 2) (6; 2)
hot	2	(1; 3) (4; 4)
in	2	(2; 3) (5; 4)
it	2	(4; 3, 7) (5; 3)
like	2	(4; 2, 6) (5; 2)
nine	2	(3; 1) (6; 1)
old	2	(3; 3) (6; 3)
pease	2	(1; 1, 4) (2; 1)
porridge	2	(1; 2, 5) (2; 2)
pot	2	(2; 5) (5; 6)
some	2	(4; 1, 5) (5; 1)
the	2	(2; 4) (5; 5)

(b) 근접 연산을 지원할 경우

역화일 개요

- 역화일 색인의 필요성
  - 대량의 텍스트파일에 대한 탐색효율 증대

- 색인 대상에 대한 제한사항
  - 색인될 키워드의 집합인 제한된 어휘
  - 불용어목록은 색인에 포함되지 않는다
  - 하나의 단어나 색인 가능한 하나의 텍스트의 시작을 결정하는 규칙들의 집합
  - 색인될 문자열의 목록

- 역화일내에서의 두가지 검색 알고리즘
  - 색인값을 반환하는 키워드(속성) 검색 : “정보검색시스템”
  - 특정 속성값을 위한 가능한 색인을 검색 : 일시, 가격 등 숫자 범위

역색인의 구축

- 역색인 구축의 과정
  - 문서에서 색인어를 추출하여 문서의 번호와 빈도수를 출력한다. { (색인어, 문서번호, 빈도) }
  - 전체에 대해서 색인어로 정렬하여 문서번호 취합. { (색인어, (갯수, { (문서번호, 빈도) }) }
  - 렉시콘 { (색인어, 포스트) } + 포스팅 { (포스트, (갯수, { (문서번호, 빈도) }) }

Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

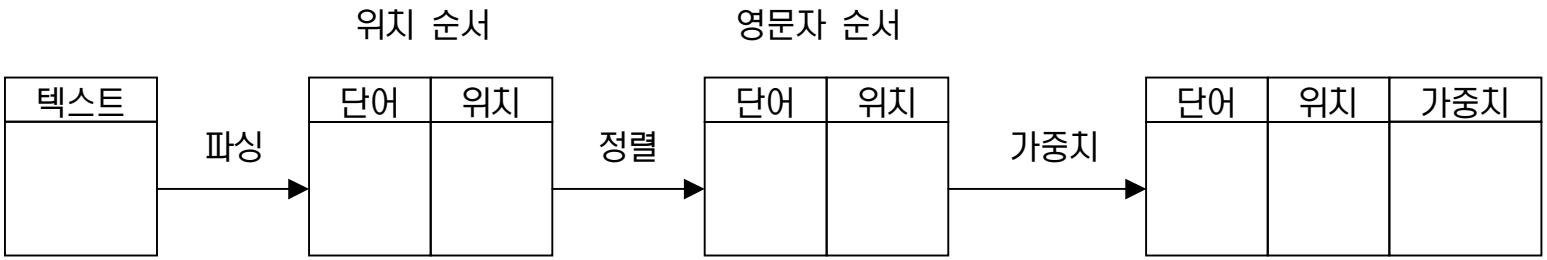
Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

렉시콘 화일에 이용되는 구조체

- 정렬된 배열
  - 단점 : 색인의 수정에 대한 높은 비용
  - 장점 : 구현의 용이성과 빠른 속도
- B+-트리
  - 접두 B+-트리
    - 트리색인의 우선키로서 단어의 접두사를 이용
    - 내부노드 : 키의 여러 숫자값을 갖는데, 각 키는 다음 레벨에 저장된 키들과 구별되는 가장 짧은 길이의 단어
    - 리프레벨 노드 : 관련된 자료에 따라 키-단어 자체를 저장
  - B+-트리
    - 단점 : 정렬된 배열을 이용하는 것보다 좀더 많은 기억장소를 이용
    - 장점 : 갱신의 용이, 더 빠른 검색시간, 보조기억장치의 이용시 더욱 효율이 좋다
- 트라이
  - 키워드를 표현하기 위해 키워드 집합에 대한 디지털 분해를 이용
- 해싱(Hashing)
  - 해싱함수 h(x)는 키 x를 주어진 범위(예: 0부터 m-1) 내의 정수로 사상시킨다.
- Signature

렉시콘 : 정렬된 배열을 이용한 역화일 구축

- 역화일 생성과정
  - 파싱 : 단어는 텍스트문서 내의 위치에 따라 단어목록으로 분할
  - 정렬 : 순서화된 항목목록으로 변경
  - 가중치 : 항목에 가중치를 주거나, 재배열 또는 파일을 압축시키는 일



렉시콘 : 정렬된 배열을 이용한 역화일 구축 (계속)

- 단어목록의 Inverting

항 목	레코드번호
pap	1
report	1
novel	1
technique	1
literat	1
result	1
technique	1
index	1
…	
report	2
charact	2
human	2
being	2
ab	2
…	

정렬 →

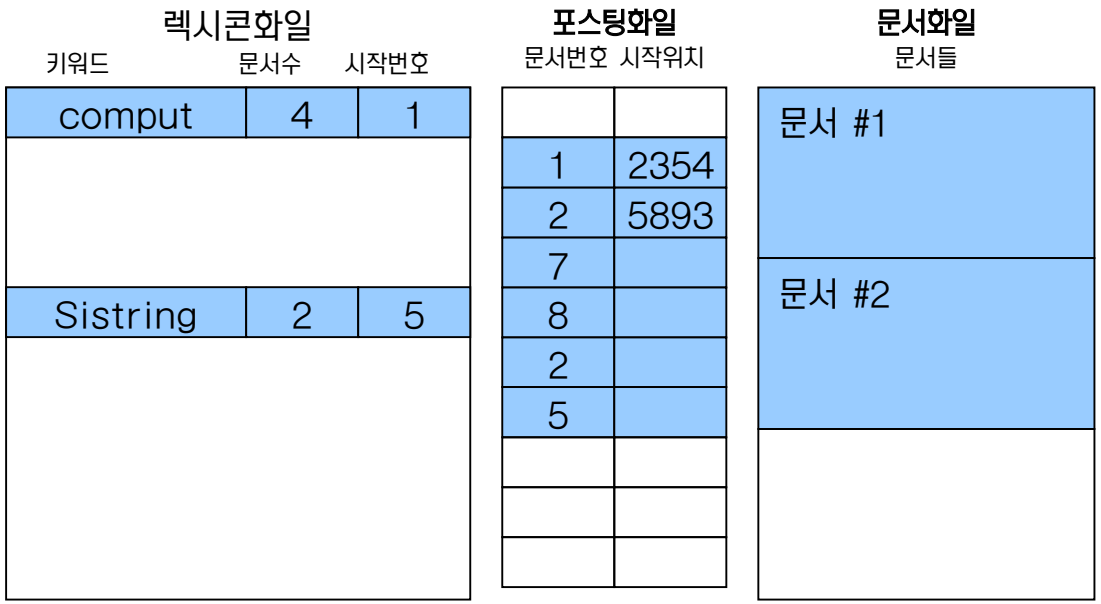
항 목	레코드번호
ab	2
being	2
charact	2
human	2
index	1
literat	1
novel	1
pap	1
report	1
report	2
result	1
technique	1
technique	1
…	
…	

중복 제거 →

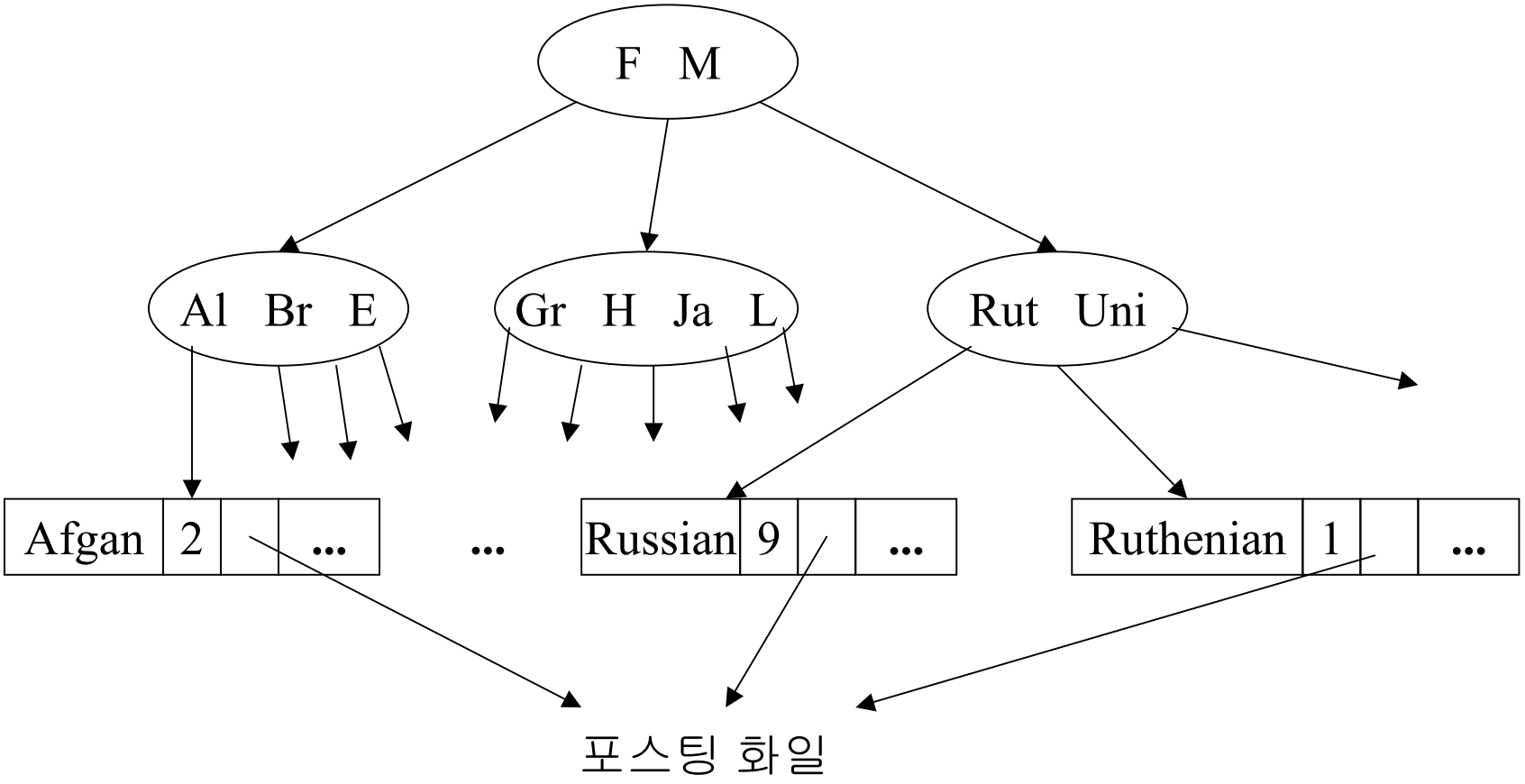
항 목	레코드번호	빈 도
ab	2	1
being	2	1
charact	2	1
human	2	1
index	1	1
literat	1	1
novel	1	1
pap	1	1
report	1	1
report	2	1
result	1	1
technique	1	2
…		
…		

렉시콘 : 정렬된 배열을 이용하여 구현한 역화일

- 가장 간단한 형태의 역색인 구성
- 개선의 여지
  - 렉시콘에서 키워드의 길이
  - 렉시콘에서 키워드 탐색 - 이진탐색
  - 포스팅에 문서번호 + 위치 혹은 빈도수
  - 포스팅의 압축
  - 문서화일에 더 많은 정보를 추가



렉시콘 : 역화일에 이용되는 구조체: 접두 B+-트리



렉시콘 : Signature file

- 시그니처 파일 (signature file)
  - 개략적 필터(inexact filter) 방식에 기반:
  - 개략적 필터 방식이란, 일단 유사하지 않은 많은 문서를 빠르게 제거하여 비교 대상이 되는 문서의 수를 크게 줄이고, 유사할 가능성이 있는 문서에 대해서 자세하게 검사하는 방식을 사용함
  - 전체 파일의 내용을 순차적으로 검색하지 않고, 파일의 내용을 부호(signature)화하여 소량의 공간에서 질의 검색하는 방법
  - 중첩 코딩(superimposed coding): 시그니처 파일 생성방법

- 중첩 코딩 방법 (Signature 생성 방법)
  - 문서를 일정한 키워드를 가지는 논리적 블록으로 나누고, 각 키워드에 대해서는 해싱 함수를 사용하여, F개 비트로 구성된 “keyword signature”를 생성하며,
  - 해당 블록에 속하는 모든 “keyword signature”를 OR하여 “block signature”를 구성한다.
  - 이때, 각 “keyword signature”는 일정 개수(= m)의 비트만이 ‘1’로 Set된다.
  - 중첩 코딩의 예제 (F = 16, m = 3)

키워드 (단어)	시그니처 (F=16 비트)
데이터베이스	0010 0100 0000 0001
시스템	0000 1000 1000 0010
질의어	0100 1000 0100 0000
블록 시그니처	0110 1100 1100 0011

렉시콘 : Signature를 이용한 검색

- 시그너처 파일을 이용한 검색 절차
  - 첫째, 질의문에 포함된 키워드를 사용하여 질의 시그너처를 생성한다.
  - 둘째, 생성한 시그너처와 저장된 블록 시그너처를 비교(AND 연산)하여 후보 문서를 결정한다.
  - 셋째, 후보 문서를 액세스하여, 실제로 질의문에 포함된 키워드를 가지고 있는지 검사한다.
  - 필터링 단계(Filtering Step): 상기 첫째 및 둘째 과정으로서, 시그너처를 사용하여 유사 가능성이 높은 후보 문서(candidate documents)를 구한다.
  - 후처리 단계(Post Processing Step): 상기 셋째 과정으로서, 해당 후보 문서가 실제로 유사한지(키워드를 모두/일부 포함하는지) 검사한다.

- **질의문: "시스템"이 포함된 텍스트를 검색하라.**

  1) **질의 시그니처 생성** 0000 1000 1000 0010

  2) **질의 시그니처** 0000 1000 1000 0010
  AND) **블록 시그니처** 0110 1100 1100 0011
  0000 1000 1000 0010
  (= **질의 시그니처**)

  3) **상기 2)와 같은 조건을 만족하는 문서들에 대해 "시스템"이 실제로 포함되었는지 검사**

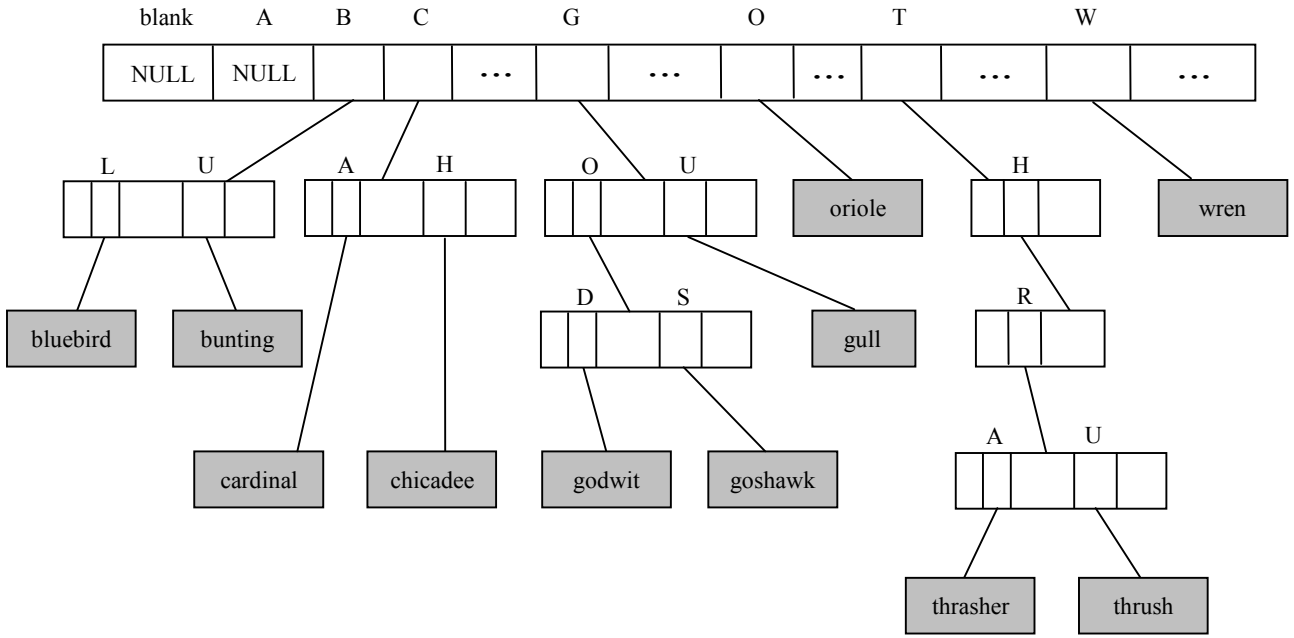
렉시콘 : Signature의 적용

- 시그니처 파일이 적합한 경우
  - PC급에서 사용되는 중 규모의 DB
  - 검색 질의 빈도가 낮은 DB (B-트리 인덱스보다 비용 저렴)
  - 병렬 처리가 가능한 DB (빠른 비교를 위해서…)

- 시그니처 파일의 장단점
  - 전문(full text) 검색보다 2배 정도 빠름.
  - 10 ~ 15%의 추가 공간이 필요 (역 파일의 인덱스가 50~300%의 추가 공간이 필요한 데 비해서 비교적 적은 양임)
  - 추가적인 삽입만을 허용하므로 삽입 연산이 간단
  - 대규모 DB에서는 속도가 저하

렉시콘 : TRIE

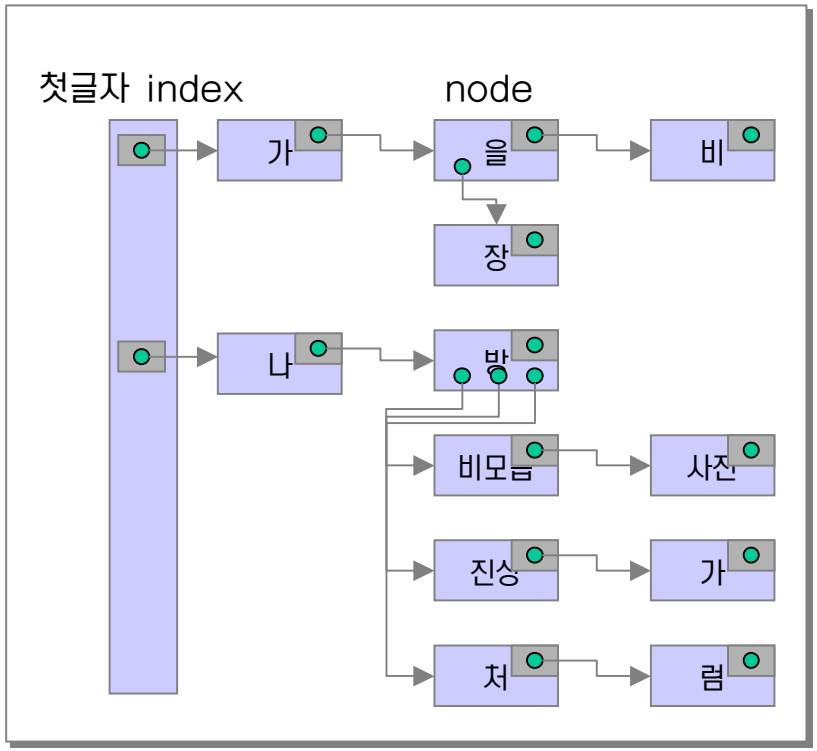
- Definition
  - an index structure that is particular useful when the keys vary in length
  - two types of nodes
    - branch node : contains pointer only
    - element node : key data



렉시콘 : TRIE : 실제 구현 사례

- 첫글자 mapping Table + String Nodes + X link + Y link Set
- 탐색 알고리즘
  - 첫글자 table에서 시작 노드 주소 입수
  - 노드를 따라가면서 노드의 스트링과와 탐색 스트링이 일치하는 만큼 이동
  - 노드의 스트링과 탐색 스트링이 일치하지 않으면 binary search로 시작 노드 찾기

첫글자 index와 Node



포스팅 : 포스팅화일의 개요

- 렉시콘과 포스팅 화일

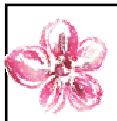
Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
1	1
2	1
2	1

Usually in memory

Gap-encoded, on disk



포스팅 :포스팅화일의 구성

- 포스팅 파일에 기입하는 정보
  - 문서번호 : 문서의 고유번호
  - 문서에서의 위치 : byte 위치, 토큰 번호, 문장 번호, phrase 번호
  - 문서에서 나타난 횟수 : 위치를 대신하는 경우

- 포스팅 파일의 압축
  - 포스팅 정보는 최악의 경우에 본문의 전체일 수 있다. 예) (A A A) (A A A) (A A A) (A A A) (A A A)
  - 보통 문서번호가 정렬되어 포스팅 파일을 구성
  - 예) 사진 = { 1, 4, 5, 10, 21, 33, 55 }
  - 포스팅의 압축 : computer라는 단어에 대해서 docIDs 283154, 283159, 283202를 대신해서 107, 5, 43 를 저장하여도 되지 않을까.

		encoding	postings list										
	the	docIDs	...		283042		283043		283044		283045	...	
		gaps				1		1		1		...	
	computer	docIDs	...		283047		283154		283159		283202	...	
		gaps				107		5		43		...	
	arachnocentric	docIDs	252000		500100								
		gaps	252000	248100									

포스팅 : 포스팅의 압축

- 포스팅의 크기는 문서전체에서 나타난 단어의 개수와 거의 비슷함
  - 예) 검색 대상 :
    - 800,000 documents, 200 tokens per document, 6 characters per token and 100,000,000 postings
    - Collection Size = 800,000 * 200 * 6 byte = 960MB
    - Uncompressed Postings file = 100,000,000 * log2 800,000 / 8 bit = 250 MB

- 포스팅 압축은
  - 문서번호를 표현하기 위해서 더 작은 bit을 사용하도록 하는것
  - 예) 800,000 문서의 경우에 20bit 보다 작은 양을 사용하도록 하는 것
  - Doc ID는 오름차순으로 저장
  - 예) Brutus: 1, 3, 7, 70, 250 …

    1, 2, 4, 63, 180 … (d-gap)

- 포스팅 압축 방법
  - Variable byte codes
  - Gamma codes

포스팅 : Fixed Length Compression

- Byte-aligned compression
  - (position, Value)
  - 32비트 숫자에 대해서 바이트의 위치를 2비트로 표시할 수 있다.

Value	Uncompressed Bit String
1	00000000 00000000 00000000 00000001
2	00000000 00000000 00000000 00000010
4	00000000 00000000 00000000 00000100
63	00000000 00000000 00000000 00111111
180	00000000 00000000 00000000 10110100

160 bit

Value	Uncompressed Bit String
1	00 000001
2	00 000010
4	00 000100
63	00 111111
180	01 000000 10110100

48 bit

Variable Length Compression : Gamma encoding

- Gamma codes for gap encoding
  - length is $\lfloor \log_2 G \rfloor$ in unary and uses $\lfloor \log_2 G \rfloor + 1$ bits to specify the length of the binary encoding of the offset
  - offset = $G - 2^{\lfloor \log_2 G \rfloor}$ in binary encoded in $\lfloor \log_2 G \rfloor$ bits.

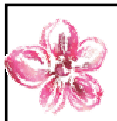
Value	Uncompressed Bit String
1	00000000 00000000 00000000 00000001
2	00000000 00000000 00000000 00000010
4	00000000 00000000 00000000 00000100
63	00000000 00000000 00000000 00111111
180	00000000 00000000 00000000 10110100

Value	Uncompressed Bit String
1	00 000001
2	00 000010
4	00 000100
63	00 111111
180	01 000000 10110100

48 bit

Value	Compressed Bit String
1	0
2	10 0
4	110 00
63	111110 11111
180	11111110 0110100

35 bit



포스팅 : 현실적인 압축 방법 variable byte code

- 이론상 좋다. (ex. gamma code) But…
  - OS는 8, 16, 32 bit 단위로 접근
    - 세부 비트로 접근하는 건 실제로는 과부하다.
    - 쿼리 프로세싱시 퍼포먼스 하강.
    - 실전에서는 간단한 byte/word-aligned 방법이 유리

- 현재 대부분의 하드웨어에서 바이트 단위가 가장 작은 접근 단위이다.
  - Suggest use of variable byte code

바이트 단위 연산

Continuation Bit	Value

- Continuation Bit : High 1 bit
  - X Value의 마지막 Byte : CB= 0
  - X Value를 인코딩 하기위해 중간에 존재하는 Byte : CB=1
- Example
  - $0 < X < 2^7$, use 1 byte
    - 0bbbbbbb
  - $2^7 \le X < 2^{(7*2)}$, use 2 bytes
    - 1bbbbbbb 0bbbbbbb
  - $2^{(7*2)} \le X < 2^{(7*3)}$, use 3 bytes, and so on …
    - 1bbbbbbb 1bbbbbbb 0bbbbbbb
- Used by many commertial/research system
  - Fast Decoding
  - Best 25%, Worst 125% (32-bit Integer)
  - Inefficient for very small gap

Value	Compressed Bit String
1	00000001
2	00000010
4	00000100
63	00111111
180	10110100 00000001

Use 48 bit

포스팅 : Word-aligned compression

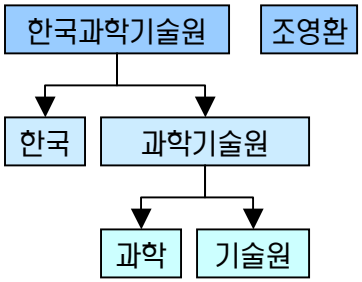
- Variable byte encoding은 gap이 상당히 작을 경우 성능이 좋지 않다.
- Word-aligned compression
  - 하나의 워드에 들어있는 Integer값은 같은 Bits 정보로 구성이 된다.
  - 적당한 파티셔닝 계획은 posting list에서 가장 큰 Doc ID Gap을 기준으로 한다.
- Word-aligned compression 방법론
  - Simple-9 : 4 selector bits + 28 data bits
  - Relative-10 : 2 selector bits + 30 data bits
  - Carryover-12 : Case of 7bit and 4bit in previous table

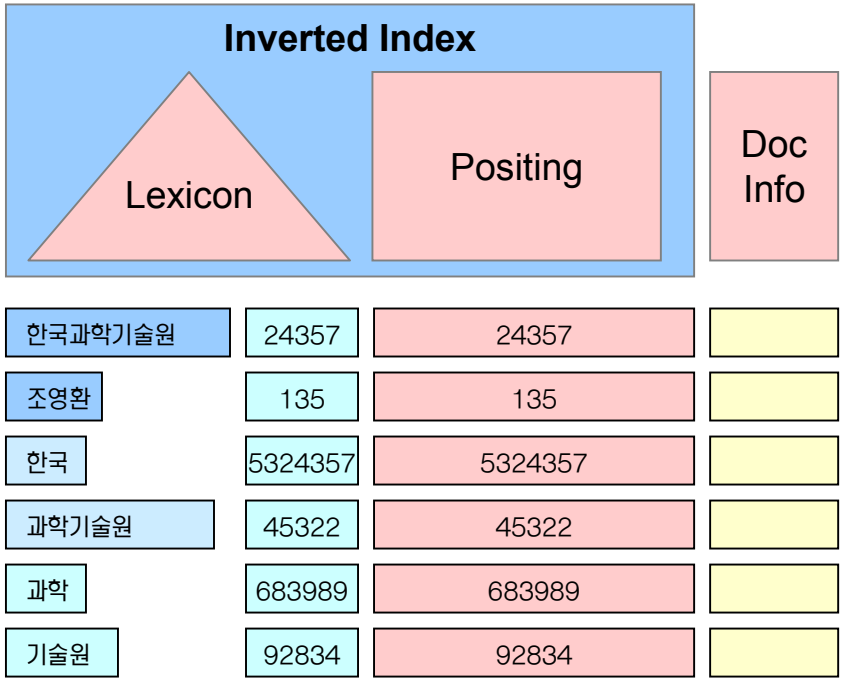
검색 알고리즘

- 쿼리 분석
- 렉시콘 탐색
- 포스팅 정보를 가지고 랭킹 연산
- 검색결과 문서번호 선택
- 각 문서에서 summary 생성

검색의 과정

한국과학기술원 조영환 [검색]





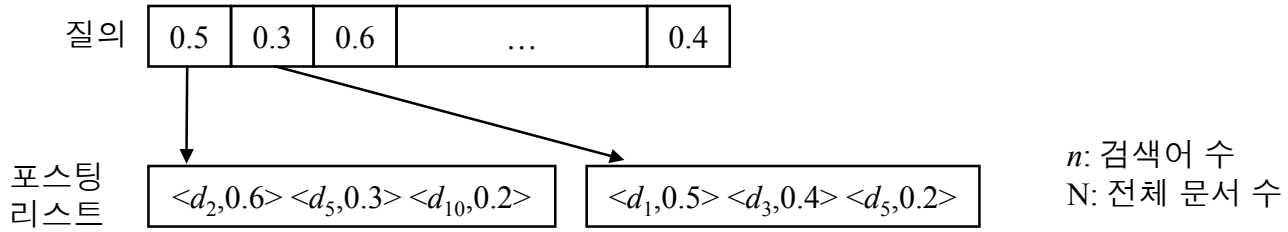
문서번호	점수	위치
문서번호	점수	위치
문서번호	점수	위치

1) 검색결과 문서 계산 = about 135
   단말 노드들에서 최소값 채용
2) 적은 post를 기준으로 AND 연산
3) 일정 개수 이상의 검색결과가 발견되면 중단

조영환	한국과학기술원	
조영환	과학기술원	한국
조영환	기술원	과학	한국

검색 결과 추출

- 검색결과 추출
  - 검색 질의어에 포함된 단어들에 대해서 Posting을 가지고 와서, 교집합을 구하는 과정



n: 검색어 수
N: 전체 문서 수

t_1처리후

1	0.0
2	0.3
3	0.0
4	0.0
5	0.15
~	~
10	0.10
~	~
N	0.0

t_2 처리후

1	0.15
2	0.3
3	0.12
4	0.0
5	0.21
~	~
10	0.10
~	~
N	0.0

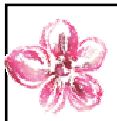
·····

t_n처리후

1	0.36
2	0.62
3	0.29
4	0.0
5	0.21
~	~
10	0.54
~	~
N	0.0

정렬

43	0.89
21	0.76
2	0.62
100	0.59
~	~
98	0.01
4	0.0
~	~
N	0.0



검색 결과 추출 : 기본적 누산기 방법의 개선

- 기본적 누산기의 메모리 낭비를 줄임
  - 누산기 생성 단계를 생략
  - 유사도 계산 과정 중에 누산기를 생성하고 초기화

1단계: 누산기 생성 및 유사도 계산

질의에 포함된 각각의 검색어 (t, w_{qt})에 대해 다음을 수행

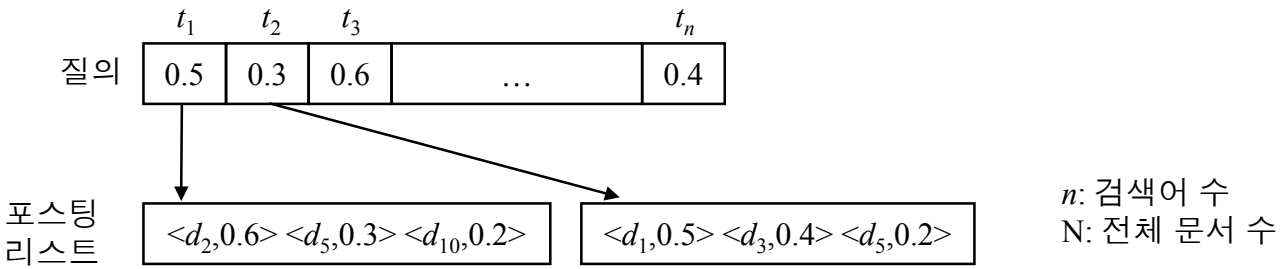
1. 역화일로부터 검색어 t에 대한 포스팅 리스트를 읽음
2. 포스팅 리스트에 포함된 각각의 포스팅 (d, w_{dt})에 대하여,
   a. 문서 d에 대한 누산기가 존재하지 않을 경우, 누산기 A_d를 생성하고 그 값을 0으로 초기화: $A_d \leftarrow 0$
   b. 검색어 t와 문서 d 사이의 부분 유사도 $w_{qt} \times w_{dt}$를 계산하여 이를 문서 d의 누산기에 합산

$$A_d \leftarrow A_d + w_{qt} \times w_{dt}$$

2단계: 검색 결과 생성

문서들을 누산기 값에 따라 정렬한 후,
사용자가 원하는 수만큼의 상위 문서들을 검색 결과로서 반환

검색 결과 추출 : 개선된 누산기 방법



n: 검색어 수
N: 전체 문서 수

t_1처리후

2	0.3
5	0.15
10	0.10

t_2처리후

1	0.15
2	0.3
3	0.12
5	0.21
10	0.10

.....

t_n처리후

1	0.36
2	0.62
3	0.29
~	~
5	0.21
~	~
10	0.54
~	~

정렬

43	0.89
21	0.76
2	0.62
100	0.59
~	~
98	0.01

검색 결과 생성

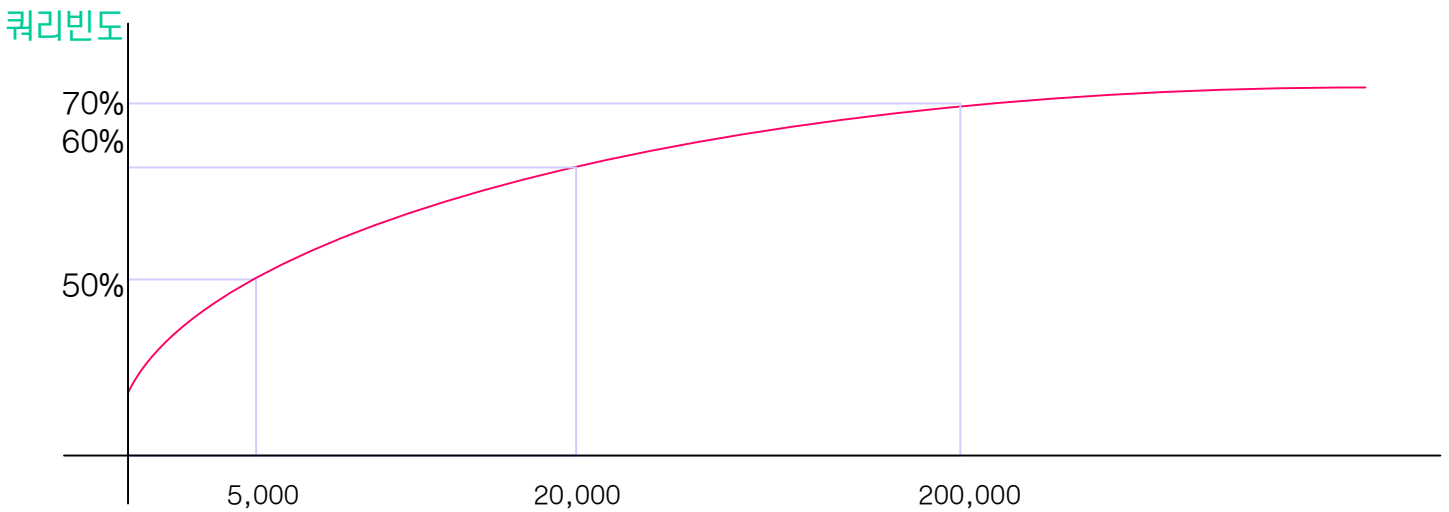
- 검색 결과 생성 단계
  - 누산기 값에 따라 문서들을 정렬한 후, 사용자가 원하는 수만큼의 상위 문서들을 검색 결과로서 반환

- 문서들의 효율적인 정렬
  - 전체 문서를 정렬하는 대신 지정된 수만큼의 상위 문서를 추출하여 정렬
  - *r*-선택 (*r*-selection) 알고리즘 사용
    - *r* 이 N보다 매우 작은 숫자일 경우 *N* 개의 값들로 부터 *r*개의 상위 값들을 추출
    - 힙 자료 구조 이용 (최소 힙)
  - 직접 작성한 qsort 함수를 사용
    - Unix, Linux 등에서 지원하는 qsort 함수의 속도가 느림

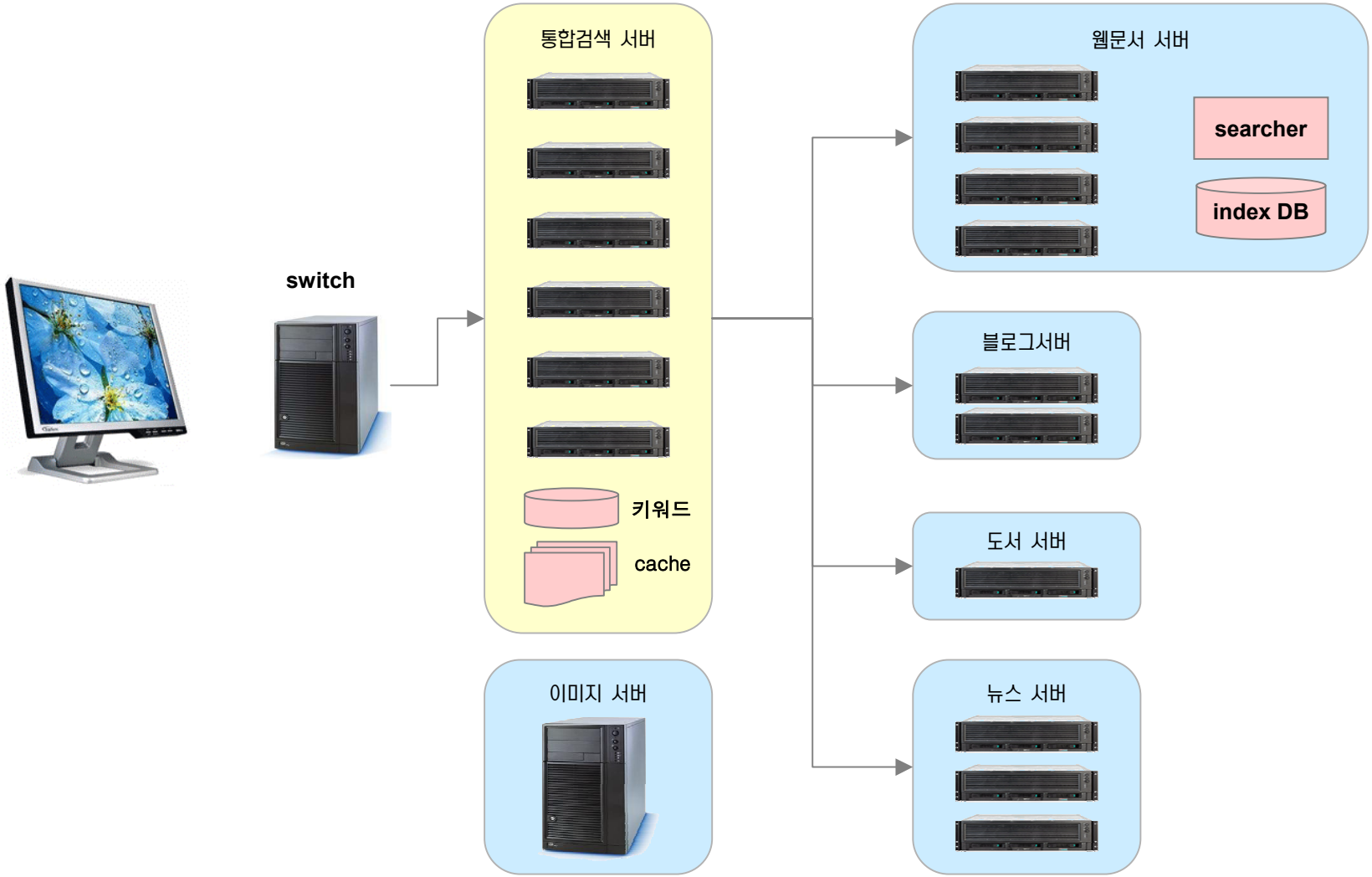
빠른 검색을 위한 장치들

- 역색인 캐시
  - 대부분의 검색 쿼리가 1개의 단어이기 때문에 단일 단어에 대한 역색인에 검색순위를 미리 계산해서 결과를 만들어 놓는 방법

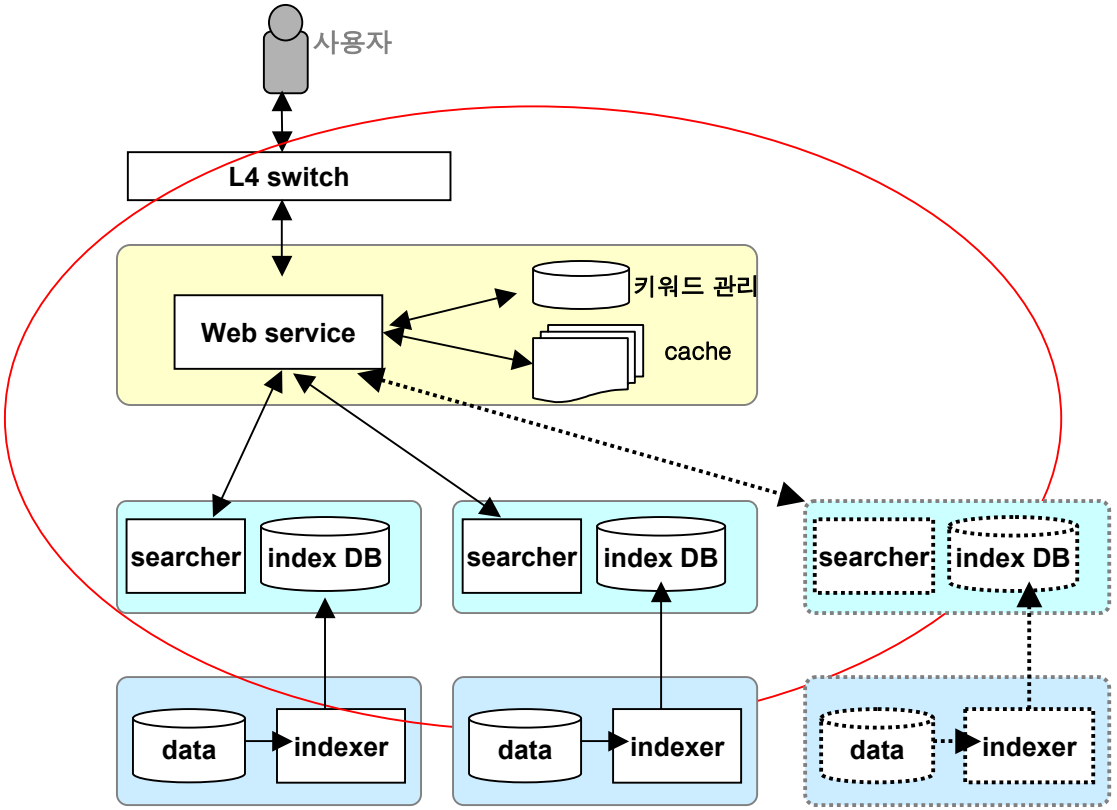
- 쿼리 캐시
  - 쿼리 입력시 자주 발생하는 검색어에 대해서 미리 검색 결과를 작성하여 놓고 그대로 검색결과를 내보내는 방법



통합검색의 구성 예)


통합검색 서버
웹문서 서버
searcher
index DB
switch
키워드
cache
블로그서버
도서 서버
이미지 서버
뉴스 서버

통합검색 - 네트웍 서비스 방식 및 병렬 처리

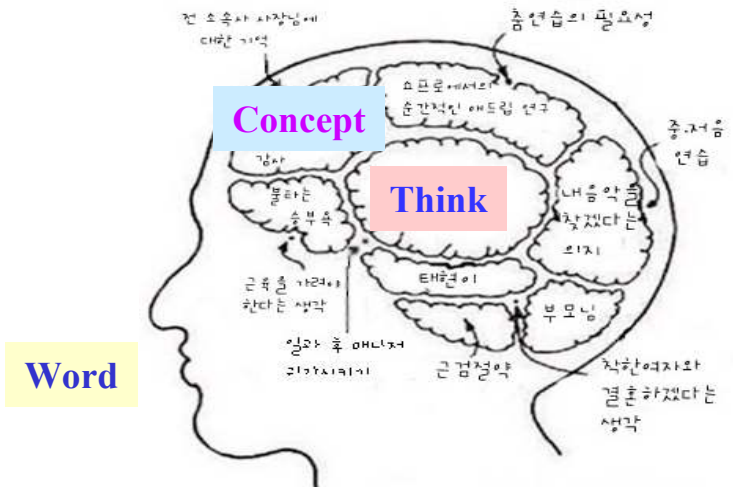


1. 네트웍 구조 서비스 :
   - 프론트에선 색인이용한 검색서비스를 하지 않음.
   - 색인파일을 프런트로 이동 후 서비스 하는 방식과 상이
   - 서비스 및 시스템 확장성 향상, 색인 이동 시간 단축

2. 병렬 처리 :
   - 백단의 검색기에 서비스 요청시 병렬 처리
   - 병렬처리시 어느 한 컬렉션에 타임아웃이 발생하면 워치독을 이용하여 각 컬렉션 작업 취소
   - 케싱되지 않은 키워드에서 검색 속도 향상

Semantics in Search

- Traditional Search vs. Semantic Search
  - Traditional Search
    - 단어의 의미를 구별할 수 없다는 가정을 기본으로 함
    - 단어 그자체를 Lexicon으로 구성
  - Semantic Search
    - 단어를 글자(character)의 나열로 보는 관점에서 벗어나서, 단어의 의미를 반영하려는 시도
    - 문서를 단순한 단어의 나열로 보는 관점에서 벗어나, 문서의 내용을 반영하려는 시도
    - 개념을 Lexicon으로 구성

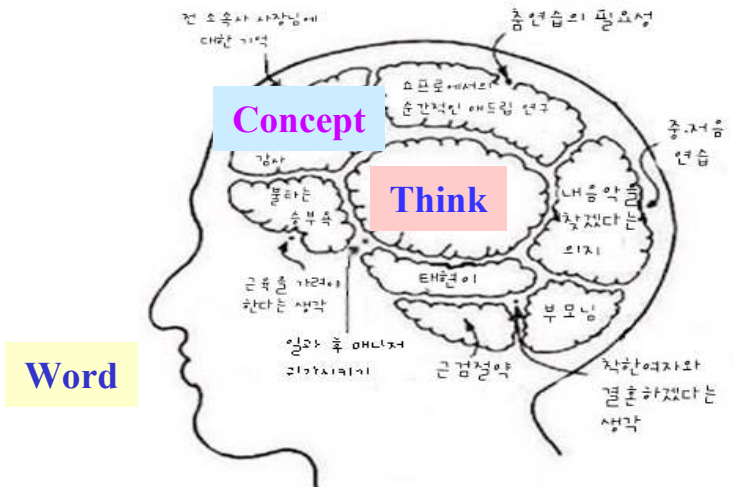


Semantic Indexing

- 개념어 색인
- 쿼리 색인
- 은유적 색인
- 감성 색인
- 평가 표현 색인

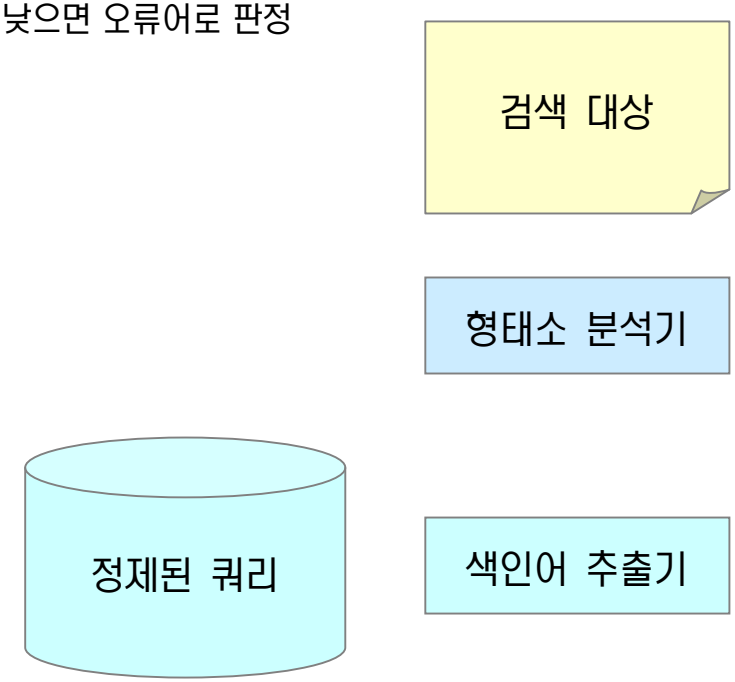
개념어 색인

- 동음이의어, 이형동의어에 대한 대응
- Word -> Concept 테이블이 필요
- 동의어 사전이라고도 불림



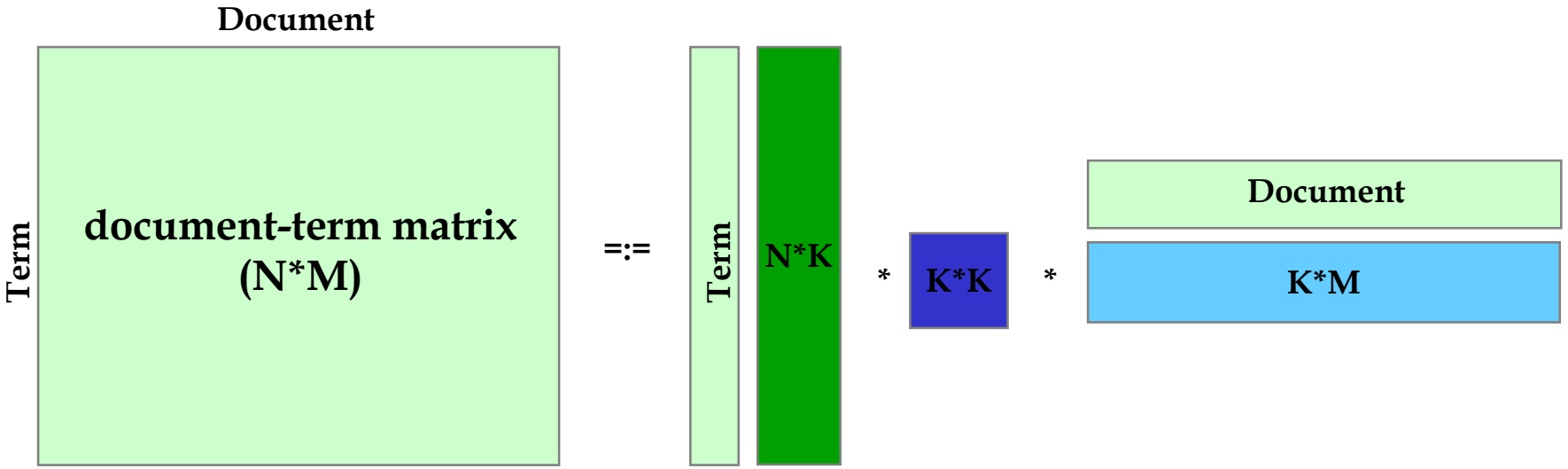
쿼리 색인

- 색인어 추출시 오류를 최소화하기 위하여 검색질의어를 색인어 추출시에 적용하여 과거에 질의 되었던 모든 단어들이 추출되도록 하는 것
- 최신단어와 형태소 분석 오류에 대한 해결책의 일부로 적용이 가능
- 사용자의 쿼리에는 입력오류가 다수 포함되어 있기 때문에 로부터 정제하는 작업이 필요
  - 쿼리 입력후 컨텐츠를 Click하는 비율이 상당히 낮으면 오류어로 판정



은유 색인

- Latent Semantic Indexing
  - 단어에 대해서 300차원 정도의 Vector로 의미를 표현
  - 단어와 문서관계 matix가 있으면 단어에 대한 은유적 의미 벡터를 추출하는 것이 가능함
  - 거대 matix 연산을 하여야 하므로 분산 처리 환경을 구축하고 SVD 적용
  - PLSA 등 여러가지 방법이 시도되고 있음



Tag 색인

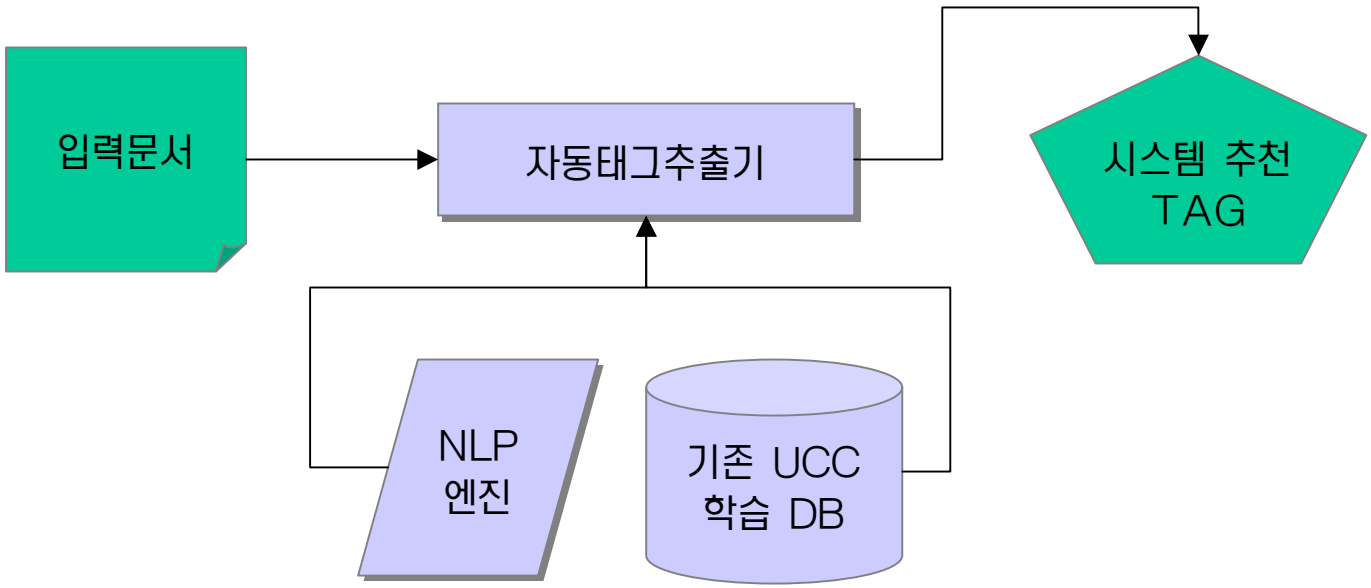
- UCC 검색과 네이게이션의 새로운 장치
  - UCC : user creates contents
  - 블로그, 카페 게시글, BBS, 사진/동영상 등의 멀티미디어 컨텐츠에 부착
- 사용자 Tag :
  - UCC 작성자가 자신의 UCC에 직접 붙여주는 Tag
  - 다른 사용자들이 검색할때에 검색어로 입력할만한 단어를 등록자가 미리 입력하는 것으로 내용을 요약해서 표현할 수 있는 단어들의 나열
  - 주제, 소재, 등장인물, 배경 장소 등
- Sys Tag
  - 고급 NLP 기술로 자동으로 추출하는 대표 키워드
  - 사용자 Tag를 유도하기 위한 추천
  - UCC 간의 연관글 링크를 통한 글 읽기 Thread 생성

How to make AutoTag

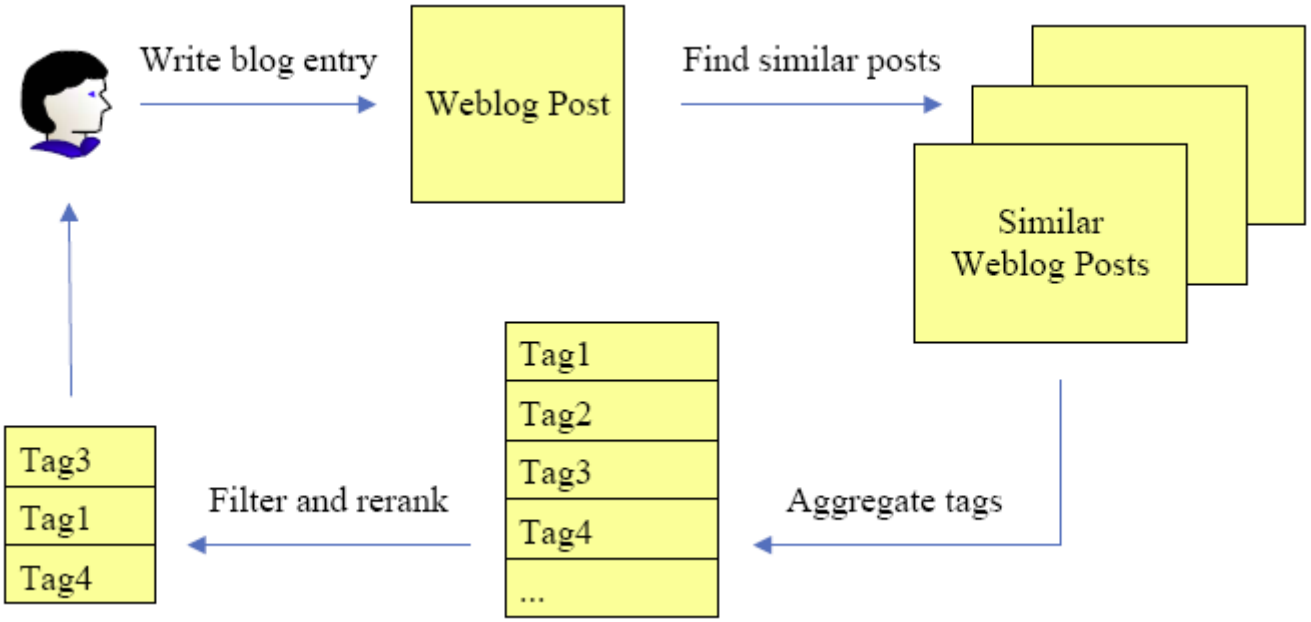
- NLP Method
  - 무엇 : 컴퓨터 프로그램이 문서의 내용을 분석해서 Tag를 자동 선택
  - 장점 : 사용자의 간섭없이 Tag 추출, Tag 재조정등이 가능
  - 단점 : 고급 NLP 엔진을 사용하여도 성능에 문제가 있음, 고객의 심증적인 동의에 의문
- Collaborative Method
  - 무엇 : 유사한 Post의 Tag들을 모아서 Reranking해서 사용자가 선택
  - 장점 : NLP 방식의 내용분석이 필요없고, 검색엔진 만으로도 구성이 가능
  - 단점 : 사용자가 컨펌해주어야 하는 절차, 기존에 많은 양의 Tagged Post가 존재하여야
- Hybrid Method : NLP with Collaboration
  - 무엇 : 제목과 사진에 대해서는 Collaborative method를 Contents에 대해서는 NLP Method를 사용하여 Tag 추출

NLP Method

- 문서에서 검색검색용 색인어를 추출하는 자동 색인기의 업그레이드 버전
- 문서의 색인어(index word)를 추출하고, 이를 문서 내에서의 관점과 문서집합(corpus)에서의 관점으로 순위를 부여하는 방식
- 정보검색용 색인어 추출은 기본단어를 꼭 포함하여 가능한 한 많은 단어를 추출하지만, Tag용 자동색인기는 문서에서 중요한 것이어야 하고, 통용되고, 틀리지 않은 것이어야 하므로 사용하는 지식베이스가 검색엔진용 색인어 추출기보다는 고급지식을 사용하여야 한다.



Collaborative Method



Hybrid Method : NLP with Collaboration

Tag Cloud와 Tag Map

- Tag Cloud란?
  - Tag Companion
  - Tag Guide
  - Tag Navigation

- Tag Cloud간의 Nevigation을 제공

Tag 추출에의 제약점

- 문서를 대표하는 것을 추출하여야 한다.
  - 문서를 대표한다는 것은 무엇을 기준으로 삼을 수 있는가?

- 사람이 사용하는 것을 추출하여야 한다.
  - 문법적으로 오류가 있는 것을 추출하면 안된다.

- 아직 Tag 추출에 대한 명확한 정의가 성립되어지지는 못한 상황
  - 확정적인 분야 : 중요한 단어라기 보다는 특정 Topic의 단어만 추출 (예: 지명만 추출, 인명만 추출)
  - 미확적적인 분야 : 검색 서비스의 새로운 모델로서의 가능성

Tag의 수준 분류

- 단어 Tag
  - 예) 삼성전자, 가을, 사진, 서울, 아파트
- 명칭 Tag
  - 예) 007 네버세이 네버어겐, 장 폴 벨몽드, 서울시 강남구 개포동
- 분야명 Tag
  - 예) [연예인명], [상장사], [현직 국회의원], [최신곡], [여행지]
- Phrase Tag
  - 예) 김옥분 속옷 노출, 가수 싸이의 병역특례, 한화회장의 보복폭행, 가장 잘 팔리는 책
  - 예) 축구를 잘하는 방법, 2007년에 예상되는 히트상품
- Fact Tag
  - 예) 비겼다(한국, 사우디, 축구경기), 비가온다(서울 경기지역), 주가(삼성전자, 643000)

Tag의 수준 분류

- 단어 Tag
  - 고유명사(인명, 지명, 기관명, 상품명 등)에 대해서는 구체적인 사례를 발견하는 것이 가능하지만, 일반명사(예: 사랑, 가을, 낙엽 등)에 대해서는 구체화가 요구된다. 단어단위를 뛰어 넘는 명칭이나 상황표현에 대해서는 Tag의 범위를 벗어나므로 이들을 포함할 수가 없다. 국어사전의 명사의 갯수는 보통 10만개, 사람 이름 50만개, 상품명/기업명/지명 등을 합하면 350만개 수준

- 명칭 Tag
  - 고유명사와 명칭 (named entity: 노래제목, 책 제목, 사건명 등)에 대해서 구체적인 사례를 발견하는 것이 가능하다. 그러나 일반명사에 대해서는 구체화가 불가능하고 상황표현 (예: 김옥분 속옷 노출, 가수 싸이의 병역 특례, 한화회장의 보복폭행, 가장 잘 팔리는 책, 포카리스웨트를 만든 회사 등)은 표현의 범위를 벗어나므로, 검색과 발견 기능의 촉진 역할을 하는 매개체(실시간 검색어 등)의 기법을 사용할 수 없다.

- 분야명 Tag
  - 단어들의 집합을 분야로 볼것인지, 분야에 속한 단어들을 찾아볼 것인지에 대해서는 고민의 여지가 있지만, 두 가지 모두 장단점은 가지고 있다. (예: 남자가수명 : 조용필, 싸이, SG 워너비, 태진아, 이승철, 바비킴) Tag로써 분야를 설정하는 것은 정보의 분류적 속성을 이용한 발견의 과정에서는 상위분야로의 이동, 즉 일반화의 범주로 이동시키는 것인데, 이것은 Tag로서는 부적합한 것이 아닌가 싶다. 다만, 정보의 내용이 부실한 경우에 혹은 상위 범주로의 이동 경로로의 안내역할로 삽입되어야 하는 것 같다. 그러므로 정보 자체에 상위 개념의 Tag를 부착하기 보다는 Navigation Path를 만드는 과정에서 즉, Tag Cloud에서의 역할 일 것 같다.

Tag의 수준 분류

- Phrase Tag
  - 명사구(예: 가수 싸이의 병역특례 문제, 한화회장의 구속)와 관형된명사구 (예: 축구를 잘하는 방법, 2007년에 예상되는 히트상품)에 대해서 구체적인 사례를 발견하는 것이 가능하다. 그러나 사실에 대한 표현 (예: 삼성전자의 순이익이 감소하였다. 주가가 사상 최고가를 경신하였다. 북한과 미국의 정상회담이 불확실하다) 등에 대한 표현의 범위를 벗어나므로, 에이전트로서의 사실 수집 기능이 약하다. 즉, 사람을 대신하는 검색/발견 행위에의 적용이 불가능 하다

- Fact Tag
  - 예) 비겼다(한국, 사우디, 축구경기), 비가온다(서울 경기지역), 주가(삼성전자, 643000)
  - 사실에 대한 표현을 Tag로 하는 것으로 사람이 정보자체에 직접 접근하여 내용을 읽지 않아도, 해당 사실에 대한 다양한 산술적인 계산을 자동화할 수 있다.
  - 문서내의 문장에 대해서 작은 단위의 문장 혹은 사실 혹은 Fact 단위로 Rewriting하여 인간과 더불어 기계도 Count할 수 있도록 변환한 것을 사실단위 Tag라고 한다. 사람과 기계가 동일하게 이해할 수 있는 공통의 단위이므로 마치 숫자를 더하거나 평균내거나 정렬하거나 하는 것과 유사하게 문서 혹은 정보에 대해서도 이러한 Operation을 가능하게 하려는 것이다. 이때에 문제가 되는 것은 구절단위의 문제와 더불어 생략(ellipsis)과 조응(anaphora)문제와 더불어 동의어(synonym), 동형이의어(Homonym)의 문제를 밟고 넘어가야 한다. 그러나 부정확한 성능이 문제가 됨에도 불구하고, 그 결과값을 이해하는 사람의 지능이 이 문제를 극복하여 줄 것으로 예상된다. 그 이유는 사람으로서는 불가능한 혹은 값이 너무 비싼 정보 reading, counting, comparison의 기능을 기계가 대신하여 주기 때문이다. 기계가 이해한 내용을 다른 기계적인 시스템의 입력으로 사용하기에는 무리가 있겠지만, 사람이라는 고지능의 시스템에 오류치의 범위와 함께 입력하여 고지능의 사람이 해독하는 것은 가능할 것으로 보여지기 때문이다.

Topic을 기반으로 하는 AutoTag

- 접근방식
  - 색인어 Ranking by Content
  - 문서에서 중요하게 사용된 단어들의 공통적인 Topic을 추출하고
  - 이러한 Topic에 관련된 문서내의 단어를 AutoTag로 추출
  - Phrase 단위의 색인어를 추출, 전체 문서 집합에서의 출현 빈도 고려

- Topic 사전
  - 예: 350만개의 명칭 사전, 15000개의 Topic 분류

EN1001. **알제리** 지리·관광정보^아프리카^알제리
가르다이아, 그랑데르그오리앙탈사막, 그랑데르그옥시당탈사막, 모스타가넴, 미티자평야, 베자이아, 블리다, 비스크라, 스킥다,
아하가르산지, 안나바, 알제, 알제리, 에디엘레, 엘골레아, 오랑, 인살라, 젤파, 지젤, 콩스탕틴, 타실리, 타하트산, 투아트, 틀렘센,
하시르멜, 하시메사우드,
EN113. **한국 문학** 문학^한국문학^기타
가루지기타령, 가산고, 가전체설화, 가정유고, 가정집, 가주집, 각암집, 각포집, 간송집, 간암문집, 간암집, 간옹문집, 간이집, 간재사고,
. . .
경와만록, 경와집, 경운가, 경허집, 경헌유고, 경화가, 계림잡전, 계묘일기, 계방고사, 계방록, 계서야담, 계아가, 계원필경, 계창집,
계하유고, 고계집, 고금소총, 고담일고, 고도암유고, 고봉집, 고산구곡가, 고산문집, 고산유고, 고산집, 고상사별곡, . . .
EN138. **보험** 사회과학^경제학^통계학^보험
가격협정보험, 가계보험, 가축보험, 간이생명보험, 간접손해, 강박보험, 강제보험, 강제책임보험, 갹출연금제도, 거치배당금,
. . .
정액보험, 조합보험, 종신보험, 중복보험, 책임보험, 초과보험, 충돌약관, 컨소시엄보험, 톤틴연금, 평준보험료,
표준보수제, 항공보험, 항해보험, 해상보험, 혼합보험, 화물보험, 화재보험, 확정보험,

평가패턴의 색인

- 목적
  - 문서내에 특정 대상물에 대한 평가의 의도가 노출되는 단어 혹은 구절 패턴을 추출하는 것
  - { (대상, 평가패턴) }

- 평가표현의 분류
  - 화자의 감정을 표현하는 단어
    - 예) 슬프다, 화가난다, 기쁘다, 싫다, 짜증스럽다, 열라 기분 나쁘다, 미칠 것 같다.
  - 제품이나 서비스에 대한 느낌이나 상태의 평가를 나타낼 수 있는 단어
    - 예) 좋다, 우수하다, 어렵다, 멋지다, 쓸만 하다, 후지다,
  - 분야별 상태 표현
    - 예) 주가가 하락하였다, 수출이 잘되고 있다, 성능의 개선이 있다, 퇴출
  - 이모티콘
    - 예) -_-; , ㅠ.ㅠ, ^^

- 총 10개 분야에 대해서 총 1만개의 평가 표현을 수집

평가패턴 추출

- 고급 NLP 분석결과를 이용한 패턴 추출
  - 문서 단위로 문서 내의 모든 내용을 분석하여 감성 패턴, 평가 패턴 등을 발견하는 방식
  - 형태소 분석, 구절 Chunking, 문장 구조 분석, 문장 관계 분석 등의 과정을 통하여 입력 Text 문서에 대한 내부 분석 자료를 만들고 이를 기본으로 각종 패턴을 분석하는 것

- TRIE를 이용한 패턴 추출
  - 형태소 분석과 구절 Chunking 정도의 분석을 하고, 문장 단위로 TRIE 탐색하면서 패턴을 추출하는 방식

감성 표현의 색인

- 목적
  - 문서내의 감성적 표현과 감성벡터를 추출
  - 21400개 표현, 20개의 Vector값

- 감성 사전
  - Source
    - 사람이 수집
    - {감성단어, 범주, 감성단어 형태의 고정 여부, 감성벡터}

  - 수집결과
    - 일반적 감성표현 : 19000개
    - 외설적 표현 : 468개
    - 유치스러운 표현 : 1280개
    - 상스러운 표현 : 408개
    - 이모티콘 : 180개

희망적이다(-1),두렵다(1),만족하다(-1), 좋다(-1),싫다(1),밝다(-1)
수상하다
암둔하다
어두컴컴하다
어둑어둑하다
어둠 깊어가다
어둠으로 차다
추운 터널

기쁘다(2),만족하다(2),좋다(2),싫다(-2),밝다(2)
가슴 찢어질 듯 기분 좋다　그냥 웃다
그냥 웃어주다　그냥 즐기다
금쪽같다　내심 만족해 하다
좋은 기분 느끼다　진심을 알다
참 잘됐구나　참 잘했어요
참 좋다　참 좋은 기분 느끼다
참 좋은 사람　친목
친목 도모하다　친밀
친숙하다

감성 표현의 대상

- 인간의 기본 감성을 나타낼 수 있는 단어
  - 예) 기쁘다, 행복하다, 즐겁다, 재미있다, 불쌍하다…
- 문서의 전체 분위기를 좌우할 수 있는 단어
  - 예) 합격, 사고, 간암말기…
- 성관련, 음란성 단어
  - 예) 섹스, 키스씬, 노출, 빤스…
- 욕설, 비속어
  - 예) 개스끼, 씨발…
- 비표준어, 인터넷상의 용어
  - 예) 리플, 삥땅, 삽질, 홧팅, 추카…
- 이모티콘
  - 예) :-), ㅠ-ㅠ, (:-(…

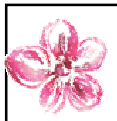
고민
고민 그만하다
고민 해결하다
고민에 빠지다

기운 잃다
기운 차다
기운 차리다
기운 없다

끔찍한 여름휴가
여름휴가

얼굴에 그늘 없다
얼굴에 그늘 지다
골치 아픈 문제
골치 아픈 문제 마무리하다
골치 아픈 문제 없애다
골치 아픈 문제 해결하다

긍정표현을 기술하는 것이 원칙.
그러나…
곱게 보이지 않다
극찬 아끼지 않다
근심걱정 끊이지 않다



감성 패턴 추출의 예

네비게이션 맵은 말할필요 없는 '맵피'라서 네비 성능은 죽여 준다.
근데 GPS 수신은 잘 모르겠네요. 처음에 인식하는 속도가 좀 느린것도 같고...
일단 GPS 잡고나면 잘 수신되는 편인것 같습니다.
지하철같은데서(지상) DMB보고 있으면 사람들이 다 기웃거리면서
신기하다는 표정으로 시선집중...
DMB 화질은 정말 좋습니다. 3.5인치에서 이정도 화면이면 훌륭하죠.
옛날에 휴대용 TV를 생각해보면...ㅋㅋ 격세지감입니다.
사무실에 있는 사람들도 DMB를 보고 모두 '우와~'합니다.
거치대는 정말 디자인이고 조작성이고 아주 괭이지만
PDA는 실물이 훨씬 멋집니다. 가벼워서 휴대성도 좋고...
이정도 가격에 이만한 성능을 가지기는 쉽지 않은거 같습니다.
가격 더 떨어질때까지 기다리려다가 정신건강을 위해 그냥 샀는데
잘한거 같습니다. 만족합니다. ㅋㅋ

좋다	= 136
기쁘다	= 100
만족하다	= 74
밝다	= 69
흥분하다	= 35
두렵다	= 5
희망적이다	= 3
자부를느끼다	= 1
불쌍하다	= 1
후회하다	= -3
슬프다	= -34
싫다	= -103

[0|23] <네비 성능> [죽여 준다]
[0|23] <네비 성능> [필요]
[2|44] <GPS 수신> [모르다]
[6|278] <DMB 화질> [시선 집중]
[6|282] <DMB 화질> [집중]
[6|138] <DMB 화질> [신기하다]
[6|155] <DMB 화질> [좋다]

[14|237] <실물> [멋지다]
[15|251] <휴대성> [좋다]
[16|278] <성능> [쉽다]
[18|303] <정신건강> [떨어지다]
[18|325] <정신건강> [잘하다]
[20|719] <> [만족]

의미의 반전

- 문장에서 평가 표현과 감성표현을 반전시키는 표현이 발생
  - 예: ~지 않다, 라고 생각할 수 없다,
  - 보조용언을 묶은 후에 추출된 평가/감성 표현에 대해 positive/negative를 판단