망망대해 인터넷에서 필요한 정보만 쏙쏙 찾아 주는 검색엔진 만들기

월드와이드웹은 거대한 정보의 집합체입니다. 이런 웹상에서의 정보 검색은 E메일 활용 다음으로 빈번하게 이루어 지고 있습니다. 현재 약 25억 개의 웹페이지가 검색 가능하지만 실제로는 이보다 훨씬 많은 웹페이지가 존재하면, 매일 수백만 개의 웹페이지가 새로 생겨나고 있다고 합니다. 이러한 정보의 바다에서 필요한 정보를 어떻게 건져 올릴 수 있을까요? 활용 특집을 통해 상용 정보 검색엔진의 구조를 분석해 보고, 직접 간단한 정보 검색엔진을 만들어 보겠습니다.

글 • 김현우 연세대학교 컴퓨터과학과 hwkim@yonsei.ac.kr

PART 1 구글을 사례로 본 검색엔진의 구조 해부

검색엔진은 무조건 많은 검색 결과를 보여주던 초창기의 단순한 형태에서 이제는 관련성 있는 어구들의 결과를 효과적으로 랭킹해 사용자가 만족할 만한 정보를 보여 주는 단계로 발전했습니다. 그 중 대표적인 것이 구글 검색엔진입니다. 1부에서는 이런 검색엔진의 구조를 해부하면서 검색이 이루어지는 원리를 이해해 봅니다.

인터넷 초창기에는 무조건 많은 검색 결과를 보여 주는 검색 서비스가 좋은 것으로 여겨졌습니다. 검색 서비스를 평가하는 척도로 색인된 웹페이지의 수를 이용했을 정도입니다. 하지만 지금은 인식이 많이 달라졌습니다. 검색엔진을 한 번이라도 이용해 봤다면 엉뚱한 결과가 너무 많이 나와 당황한 적이 있었을 것입니다. 보통 쓰레기 결과(junk result)라고 부르는데, 예를 들면 탤런트 '장동건' 에 대한 정보를 찾고 싶었는데 '마장동 건어물 가게' 가 나오는 식이지요. 예전에는 이렇게 동문서답이 주 특기인 사오정식 검색엔진이 많았으나 요즘에는 다행스럽게도 똑똑한 검색엔진이 많이 나와 있습니다. 그 중 대표적인 것이 '구글(Google)' 이라 할 수 있습니다.

인터넷 사용자들이 검색엔진의 기본 기능에 대해서는 대체로 친숙합니다. 보통 한 단어 정도를 입력하는 경우가 많은데, 이런 검색을 '단순 검색(Simple Search)' 이라고 합니다. 사용자들 대부분이 단순 질의를 사용한다는 것을 알기 때문에 단순 검색엔진들은 관련된 검색 결과의 정확성 대신 검색 대상 중 해당되는 모든 문서들을 보여 주는 데 만족합니다. 관련 문서들의 목록을 그대로 보여 주는 것이 관련성이 있는 어구들의 결과를 효과적으로 랭킹(ranking)해 보여 주는 것보다 훨씬 쉽기 때문입니다. 하지만 요즘 검색엔진은 웹 문서의 링크 구조를 활용해 랭킹 과정을 거치도록 합니다. 검색 결과의 첫 화면에 사용자가 만족할 만한 정보를 보여 주고자 하는 것입니다.

검색엔진의 구성 요소

검색엔진은 크게 다음과 같은 요소로 구성되어 있습니다.

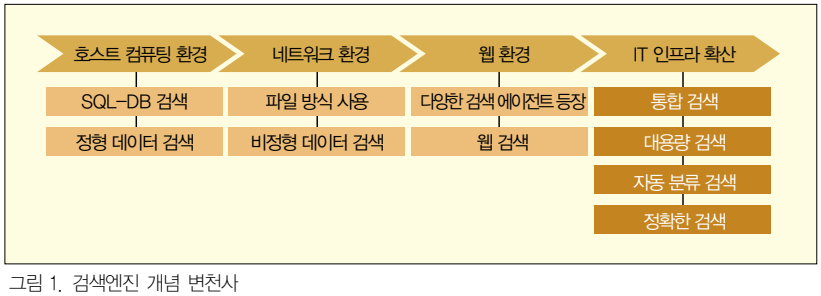

그림 1. 검색엔진 개념 변천사

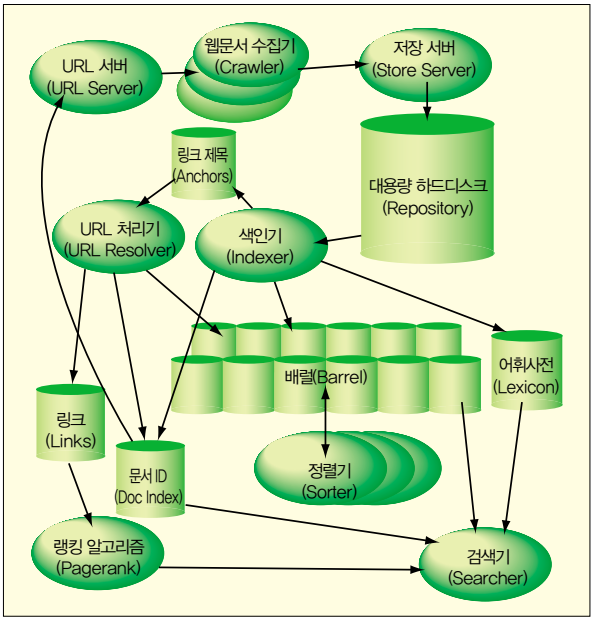

그림 2. 구글 검색엔진의 추상화된 하이레벨 구조

1 가져 올 웹페이지 주소를 관리하는 URL 서버

URL 서버는 단순히 큐라고 보아도 무방합니다. 앞으로 가져와야 할 웹페이지들의 주소를 관리하는 것이지요. 그렇다면 그 많은 주소들을 모두 입력해야 할까요? 그것은 절대로 아닙니다. 처음 시작점만 정해 주면 그 페이지에서 전방위 링크(forward link)들을 뽑아 내서 큐에 삽입하기 때문에 끊임없이 웹페이지를 가져 올 수 있습니다. 경우에 따라서는 다른 사이트를 가리키는 링크와 같은 사이트의 다른 페이지를 가리키는 링크를 구분해 관리하기도 합니다. 같은 사이트 링크만을 따라 가도록 하면 사이트 내의 검색엔진이 되는 것이고, 두 가지 모두 따라 가도록 하면 대규모 웹 검색 엔진이 되는 것입니다.

2 웹페이지를 가져 오는 웹문서 수집기

그러면 웹페이지는 어떻게 가져 올까요? URL 서

버가 다운로드해야 할 웹페이지의 주소를 웹문서 수집기(Web Crawler)에게 알려 주면 웹문서 수집기는 주어진 주소로 소켓 접속을 시도합니다. HTTP도 텔넷이나 FTP 접속과 마찬가지로 소켓을 통해 접속하는 것이지요. 보통 80번 포트를 사용합니다. 다른 포트를 사용할 경우에는 URL에 포트 번호가 주어지므로 정보를 제대로 읽어 내기만 하면 어떠한 웹페이지도 읽어낼 수 있습니다. 다음은 80번 포트를 사용하지 않고 다른 포트(9999번)를 사용할 경우 URL의 예제입니다.

http://www.pserang.com:9999/dir/test.html

우리가 웹브라우저를 통해 어떤 URL을 입력하여 접속했을 때는 그 페이지가 가리키고 있는 그림들을 모두 다운로드 해서 화면에 보여 주지만, 텍스트 기반의 검색엔진에서는 HTML 파일 하나만을 가져 옵니다. 좀 더 정성들여 잘 처리하고자 한다면 프레임 정보를 읽어서 따로 따로 가져 온 다음 통합하여 마치 하나의 페이지인 것처럼 보이게 합니다. 메인 페이지에는 링크들만 있고 주 정보는 프레임 페이지에 있는 경우가 많습니다. 여기서 주의할 것은 프레임 안에 또 다른 프레임(subframe)이 존재할 수 있다는 점입니다. 모든 정보들을 가져 왔으면 이들을 통합한 후 파싱(parsing)하여 링크 정보와 본문을 추출해 낼 수 있습니다. 링크 주소는 URL 분석기로 넘어가 상대 경로는 절대 주소로 변환되고, 추후 처리시 혼돈이 생기지 않도록 정규화(normalization) 과정을 거친 후 URL 서버의 큐에 삽입됩니다. 본문(body text)은 형태소 분석기(morphological analyzer)에 의해 언어의 가장 기본적인 단위로 분할되며, 이를 이용한 다양한 자연언어 처리 과정을 통해 요약문 생성, 명사 추출, 수사 표현(numerical expression) 정규화, 인명 · 지명 등의 고유 명사 인식 등이 이루어 집니다.

3 웹페이지의 모든 정보를 담고 있는 저장 서버

웹페이지에 대한 모든 정보를 담고 있는 대용량 저장소를 가지고 있는 서버를 저장 서버(Store Server)라고 합니다. DB를 사용해 웹페이지를 관리할 수 있도록 하지만 이 자료가 검색용으로 쓰이는 것은 아닙니다. 단지 보관을 하여 추후에 다시 색인을 할 수 있도록 하기 위한 것이지요.

검색을 위해서도 DB를 쓰기는 하지만 오라클 등 우리가 흔히 알고

정확률(presicion)과 재현률(recall)

정보 검색 시스템의 성능을 측정하는 척도로 정확률과 재현율이 사용됩니다. 정확율은 검색된 문서들이 사용자의 질의와 얼마나 잘 부합되는가를 뜻하며, 재현율은 검색된 문서들 중에서 적합한 문서가 어느 정도인가를 나타냅니다. 〈그림 3〉과 수식을 보면 이해가 한결 쉬울 것입니다.

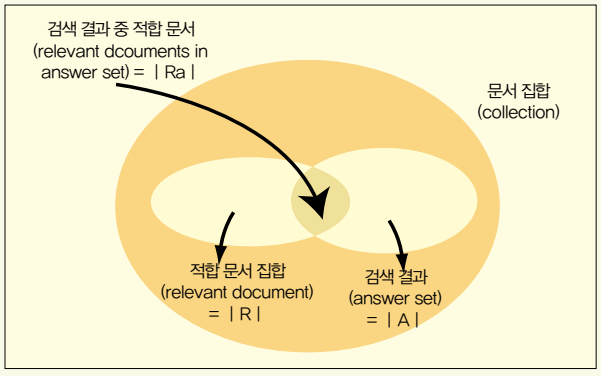

그림 3. 재현률과 정확률에 관한 그림

재현률과 정확률에 관한 수식은 다음과 같습니다.

- 재현율(Recall) = $\frac{|R_a|}{|R|}$
- 정확률(Precision) = $\frac{|R_a|}{|A|}$

다음과 같이 정확률을 높이기 위한 방법이 몇 가지 있습니다.

- boolean 연산자의 and를 적절히 사용합니다.
- 인접 연산자를 사용합니다.
- 검색어를 선정할 때 일반적인 용어를 지양하고 특정어를 사용합니다.

보통 재현률과 정확률은 반비례 관계입니다. 즉, 정확율을 높이면 재현율이 떨어지고, 재현율을 높이면 정확율이 떨어지고 마는 것이지요. 검색엔진에서 '사랑'이라는 키워드로 검색하면 많은 문헌이 검색되지만, 이들 검색된 문헌 중에서 다시 적합한 문헌을 찾아야 하는 과정이 요구됩니다. 이는 재현율을 높이되 정확율을 낮추는 검색 기법이라 할 수 있습니다. 모 검색엔진의 TV광고에서처럼 가수 '핑클'이라는 검색어으로 '서핑클럽'이라는 불필요한 문헌까지 검색이 되는 등의 검색 잡음이 발생할 수 있습니다.

반대로 '사랑의 슬픔'이라고 하면 '사랑'과 '슬픔'이라는 단어가 모두 포함된 문헌만 검색되므로 이 문헌들이 자신이 원하는 문헌일 확률이 높아집니다. 이는 정확률을 높이는 검색기법이라 할 수 있습니다. 그러나 필요한 문헌이 원래 '사랑의 아픔'인데 만약 '사랑의 슬픔'으로 착각하고 있을 때는 '사랑의 슬픔'으로 검색하면 영원히 그 문헌을 찾을 수가 없습니다.

그러므로 이러한 경우에 대비하여 정보 검색시에는 재현률과 정확률을 면밀하게 고려해야 합니다. 만일 boolean 연산자가 지원되는 검색 시스템이고 필요로 하는 문헌이 '사랑의 슬픔'인지 '사랑의 아픔'인지 확신이 서지 않는다면 검색 시스템에서 '사랑 and (슬픔 or 아픔)'으로 검색하면 단번에 결과를 얻을 수 있을 것입니다. 재현률과 정확률 두 가지를 동시에 올릴 수 있는 검색엔진을 개발하는 것이 정보 검색을 연구하는 사람들의 공통된 목표입니다.

압축 라이브러리 zlib와 bzip

zlib는 RFC1950을 따르는 압축 라이브러리입니다. zlib는 압축 속도가 bzip에 비해 빠르지만 압축률은 평균 3 : 1 정도입니다. bzip은 zlib보다 압축 속도는 조금 떨어지지만 4 : 1 정도의 우수한 압축률을 보입니다. 이 때문에 구글에서는 다수의 이용자에게 빠른 속도를 제공할 수 있도록 zlib을 채택하여 사용하고 있습니다.

- zlib 홈페이지 : www.gzip.org/zlib/
- bzip 홈페이지 : sources.redhat.com/bzip2/

있는 상용 DB를 사용하는 경우는 드뭅니다. 성능이 잘 나지 않기 때문입니다. 검색엔진은 검색 품질과 더불어 빠른 속도 역시 생명입니다. 느림보 검색엔진은 타 검색엔진과의 경쟁에서 살아 남을 수 없겠지요?

docID (문서ID)
URL
웹브라우저 상단에 표시되는 제목
자신을 링크한 페이지(parent page)가 사용한 링크의 제목
다운로드 시각
전방 링크 목록
Raw HTML 코드
형태소 분석 결과물

표 1. DB에 저장되는 웹페이지 정보들

구글이나 와이즈넛 같은 검색엔진에서는 미리 보기 기능을 지원합니다. 정보를 가지고 있는 사이트로 이동하지 않고서도 바로 HTML 문서를 볼 수 있는 기능입니다. 이런 기능은 해당 사이트가 어떤 장애로 인해 서비스가 일시적으로 중단되었다거나 접속 속도가 매우 느릴 경우 매우 유용하게 쓸 수 있습니다. 이런 서비스를 제공하기 위해 저장소에 웹페이지의 HTML 코드를 넣어 두어야 하는데, 모든 웹페이지의 HTML 코드를 가지고 있는다는 것은 용량 문제로 매우 부담스러운 일입니다. 그래서 압축을 하게 되는데, 압축 라이브러리로는 zlib나 bzip이 속도가 빠르고 압축률도 좋아 많이 쓰입니다.

4 URL 구성 요소를 세밀하게 분리하는 URL 처리기

URL(Uniform Resource Locator)은 프로토콜 이름, 서버 주소, 파일 경로로 구성되며 다음과 같이 표기합니다.

```
protocol://server_address:port/path/filename
```

예를 들면 다음과 같습니다.

```
http://suny.yonsei.ac.kr/~lihu/index.html
```

웹페이지 검색시 프로토콜은 항상 HTTP(Hypertext Transfer Protocol)로 한정되기 때문에, URL을 처리할 때 'http://' 는 생략해도 무방합니다. 서버 주소는 호스트(host) 이름, 서브 도메인(sub domain), 기관 종류 코드, 국가 코드, 포트(port) 번호로 구성됩니다. 여기서 포트 번호는 대개의 경우 생략되는 것이 보통이며, 생략시에 사용되는 디폴트 값은 80입니다. 예를 들면 'http://suny.yonsei.ac.kr:2002/~lihu/babo.cgi?id=24&value=36' 는 〈표 2〉와 같이 세분됩니다. 이런 식으로 URL의 구성 요소를 세밀하게 분리해 내는 이유는 다음에 소개하는 기능들을 구현하기 위함입니다.

호스트 명	suny
서브넷	yonsei
도메인확장자	ac.kr
포트 번호	2002
디렉토리	~lihu
파일 명	babo.cgi
매개변수 1	id=24
매개변수 2	value=36

표 2. URL 구성 요소 분류의 예

5 링크

링크는 홈페이지 제작자의 편의상 상대 경로(relative path)로 작성되는 경우가 많습니다. 절대 경로(absolute path)에 비해 보다 적은 글자 수로 표현할 수 있고, 문서 집합을 다른 서버에 옮길 경우 별도의 문서 변경 작업이 필요하지 않기 때문이지요. 웹문서 수집기는 모든 페이지에 대하여 정확한 절대 경로를 알고 있어야 하기 때문에 상대 경로를 반드시 절대 경로로 변환해야 합니다.

예를 들어 'suny.yonsei.ac.kr/poem/list.html' 에서 './seosi.html' 을 링크하고 있을 경우 이 링크는 'suny.yonsei.ac.kr/poem/seosi.html' 로 변환하여 인식되어야 합니다.

6 사이트의 안팎 구분

사이트 안과 밖을 구분해야 할 필요가 있습니다. 구분을 하지 않을 경우 사이트 내 검색이 불가능하며, 웹페이지를 가져 올 때에도 계속 여러 사이트를 오가며 가져 오게 되므로 단위 시간당 처리량이 낮아지게 됩니다. 한 사이트에서 집중적으로 가져오고 또 다른 사이트에서 집중적으로 가져 오도록 하는 방식이 여러 모로 효율적입니다. URL을 읽어서 서브넷 이름과 그 이하의 도메인 확장자가 서로 같으면 같은 사이트로 봅니다.

호스트 이름은 웹서비스 내에서 부하 분산 혹은 부서 구분 등의 목적으로 다양하게 나타나기 때문에 어떤 페이지가 현재 수집하고 있는 사이트 내에 속하는 것인지 확인할 때는 이를 고려하지 않습니다. 예를

들면 'www.yonsei.ac.kr', 'mythos.yonsei.ac.kr', 'cs.yonsei.ac.kr', 'coms.yonsei.ac.kr'가 모두 한 사이트로 취급되는 것입니다. 이런 확인을 하기 위해서는 URL에서 hostname 부분을 정확히 알 수 있어야 합니다. 어떤 경우에는 hostname이 없을 수도 있는데, 이를 구분하지 못하면 서브넷 이름을 호스트 이름으로 인식하게 되므로 유의해야 합니다.

도메인 확장자를 정확히 구분할 수 있으면 이 문제는 쉽게 해결될 수 있습니다. 도메인 확장자는 두 가지 타입을 갖는데, 첫 번째는 com, edu, gov 등과 같은 최상위 확장자(top-level extension)이고, 두 번째는 co.kr, ac.kr, go.kr 등과 같은 기관 종류 코드와 국가 코드로 이루어진 것입니다.

7 URL 정규화(Normalization)

경우에 따라 같은 페이지가 여러 개의 URL로 표현될 수 있는데, 웹 크롤러는 URL이 다르면 서로 다른 페이지인 것으로 이해하기 때문에 URL을 정규화해야 할 필요가 있습니다. 저장 공간이 낭비되는 문제도 있지만, 그보다 더 심각한 문제점은 추후에 검색 결과 출력시 같은 내용이 페이지가 여러 번 중복되어 나타날 가능성이 높다는 것이지요. 예를 들면 다음과 같은 경우를 들 수 있습니다.

❶ www.yonsei.ac.kr/~user1
❷ www.yonsei.ac.kr/~user1/
❸ www.yonsei.ac.kr/~user1/index.html
❹ www.yonsei.ac.kr/~user1/index.html/

직감적으로 알 수 있듯이 위의 네 개 주소는 실제로는 모두 같은 물리적 위치의 문서를 가리키고 있습니다. 경로의 마지막 부분이 파일 이름이 아니고 디렉토리 이름이라면 웹서버가 자동으로 주소 재지정(Address Redirection)을 수행하기 때문에 ❶이 저절로 ❷의 형태로 변환되며, ❷의 경우 파일명이 지정되지 않았기 때문에 디폴트 파일에 접근하려는 것으로 간주됩니다. 디폴트 파일명은 웹서버의 환경 설정 파일에 정의되어 있으며, 일반적으로 index.html 혹은 index.htm입니다. ❹의 경우 마지막 부분이 파일명이고 슬래쉬(/) 기호 뒷부분이 없기 때문에 가장 마지막의 슬래쉬 기호는 무시되고 맙니다. 따라서 ❶~❹는 모두 같은 주소이며, 이를 어떤 한 가지 형태로 통일할 필요가 있습니다. 필자는 ❷번의 형태로 통일할 것을 추천합니다.

8 기준 위치(Base location) 재지정

HTML 문서는 헤드 부분에 base 태그를 사용하여 상대 경로로 쓰여진 작성된 링크에 대한 기준 위치를 다른 곳으로 지정할 수 있습니다. 즉, 상대 경로를 절대 경로로 변환할 때 현재 파일의 위치를 기준으로 하는 것이 아니라 base 태그에서 지정한 파일의 위치를 기준으로 하는 것입니다. base 태그는 빈번하게 사용되고 있지는 않으나 이를 고려하지 않을 경우 간혹 엉뚱한 주소를 참조하는 경우가 생길 수 있으므로 유의해야 합니다.

리스트 1 · 기준 위치(base location) 재지정문 사용의 예

```
<HTML>
 <HEAD>
   <TITLE>My favorite song</TITLE>
   <BASE href="http://www.home.com/song/list.html">
 </HEAD>
 <BODY>
   I like Matt Monroe's
   <A href="../pop/walk_away.html">Walk Away</A>?
 </BODY>
</HTML>
```

9 페이지 필터링(Page Filtering)

사용자는 검색 결과에 광고 페이지가 들어 가는 것을 원치 않을 것입니다. 예를 들어 호스트 이름이 ad인 경우 온라인 배너(banner) 광고를 위한 서비스인 경우가 거의 대부분입니다. 특정 호스트 이름 혹은 서브도메인 이름을 등록할 경우 이에 해당하는 페이지들을 수집하지 않도록 하는 필터링 기능을 구현한다면 보다 만족스러운 검색 결과를 얻는 데 보탬이 될 것입니다.

10 HTML 문서를 분석하는 색인기

색인기는 여러 가지 역할을 합니다. 저장소에서 문서들에 대한 정보를 받아와 압축을 풀고 HTML 문서를 분석합니다. 자연언어 처리를 가미할 경우에는 본문을 추출하여 형태소 분석 및 품사 부착을 하게 되는데, 결과물의 예는 〈표 3〉과 같습니다.

오늘밤에도 별이 바람에 스치운다					
형태소	품사	확률	형태소	품사	확률
오늘밤	nct	0.002058	에도	jca	0.014641
별	nc	0.001496	이	jc	0.264449
바람	nc	0.000869	에	jca	0.352391
스치우	pv	0.000000	ㄴ다	ef	0.147248

표 3. 형태소 분석 결과의 예

각각의 형태소에 대해 품사와 확률 값이 주어짐을 알 수 있습니다. 보통의 경우에는 이런 정보를 모두 이용하는 것은 아니고 명사만을 추출하여 색인어를 사용합니다. 문서의 모든 색인어는 해당 문서에서의

가중치를 가집니다. 출현 빈도와 문서에서의 위치가 기본적으로 고려되고, 좀더 엄밀하게 가중치를 계산하는 경우에는 해당 색인어가 HTML 문서에서 상대적으로 글씨 크기가 큰지, 굵은 글씨인지, 특정 색으로 강조되어 있는지, 대문자들로만 이루어져 있는지 등의 여부 등을 검사하여 가중치를 조절합니다. 이런 값을 hits라 합니다. 색인기는 hits를 배럴 집합으로 분배시킵니다. 배럴에는 해당 색인어가 어떤 문서에서 출현하는지를 나타내는 정보가 구축됩니다. 이를 역색인 정보라고 합니다.

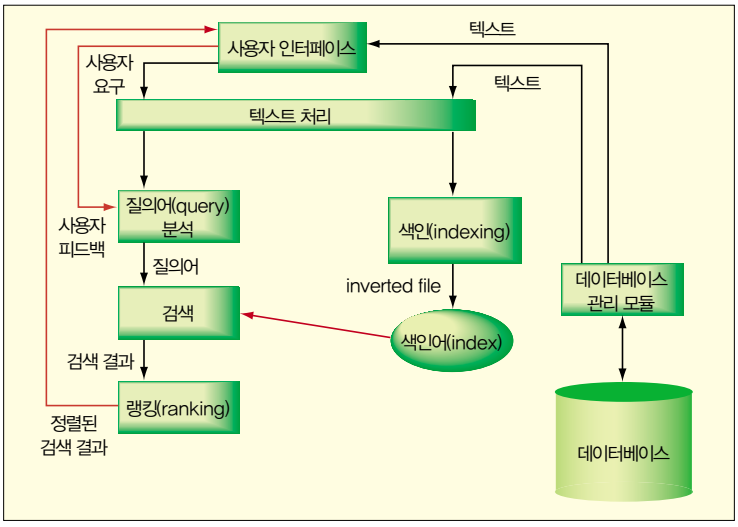

그림 4. 검색 처리 과정

검색 처리 과정

지금까지 검색엔진의 구성 요소에 대해 알아 보았습니다. 이런 구성 요소들이 어떤 역할을 하고 있는지에 대해서도 이해하게 되었을 것입니다. 그렇다면 실제로 검색 요청이 들어왔을 때 어떠한 순서로 처리되는지도 대충 감이 올 것입니다. 잘 모르겠다면 〈그림 4〉를 한번 보기 바랍니다. 화살표를 따라가면 어떤 흐름으로 처리되고 있는지 알 수 있습니다.

이런 검색 과정을 간략하게 의사코드(pseudocode)로 나타내면 다음과 같습니다.

❶ 질의어를 파싱한다.
❷ 모든 키워드를 wordID로 변환한다.
❸ 배럴에서 각 단어에 해당하는 역색인 정보를 읽는다.
❹ 검색 조건을 모두 만족하는 문서가 나타날 때까지 스캔한다.
❺ 사용자의 질의어에 대한 해당 문서의 랭크를 계산한다.
❻ 문서 목록의 마지막에 도달하지 않았다면 다시 ❹번으로 간다.
❼ 랭크를 기준으로 문서들을 정렬한 후 결과를 반환한다.

지금까지 검색엔진의 구조를 해부하면서 검색이 이루어지는 과정까지 알아 보았습니다. 간단하기는 하지만 검색엔진의 원리를 이해했을 것입니다. 2부에서는 위의 내용들을 바탕으로 실제 검색엔진을 제작해 보겠습니다. PW

정리 • 김상연 기자 sykim@pserang.co.kr

PART 2 도전! C로 초짜도 할 수 있는 검색엔진 만들기

1부에서 검색엔진의 구조를 통해 검색이 이루어 지는 과정을 이해한 데 이어 2부에서는 리눅스 환경에서 C 언어를 사용한 검색엔진을 직접 만들어 보겠습니다. 검색엔진 제작에 필요한 핵심 라이브러리 사용법을 간단하게 소개한 후 사이트의 웹페이지를 모두 방문하는 탐색 알고리즘부터 필요한 문서 내용을 검색할 수 있는 색인기를 구현합니다.

요즘 배포되는 리눅스에는 BerkeleyDB, MySQL, zlib, wget이 모두 기본적으로 갖추어져 있으므로 여러분은 그냥 하려는 의지만 있으면 됩니다. 솔라리스 같은 유닉스 환경에서 작업하는 이라면 준비물을 먼저 갖춰야 합니다(오른쪽 박스 참조). 자, 준비가 끝났으면 이제 필요한 라이브러리의 사용법부터 차근차근 익혀 보겠습니다.

검색엔진 제작을 위한 준비물 챙기기

- 리눅스나 유닉스 머신
- Berkeley DB(www.sleepycat.com)
- MySQL(www.mysql.com)
- zlib(www.gzip.org/zlib/)
- wget 유틸리티(www.wget.org)

검색엔진 제작을 위한 라이브러리 사용법 익히기

정보 구축과 검색을 위해 오라클이나 MySQL, MS-SQL 등의 DBMS를 사용하는 것이 편리하지만 정보 검색엔진에서 이런 상용 DBMS를 사용하는 경우는 드뭅니다. 대규모 검색엔진에서 이러한 DBMS로는 좋은 성능을 내기 어렵기 때문입니다. DB를 거치면서 매우 큰 오버헤드가 발생하여 반응 속도가 급격히 떨어지게 되고, 심지어는 과부하로 인해 서비스가 마비될 수도 있기 때문입니다. 그래서 보다 저수준(low-level)의 단순한 DB를 사용하게 됩니다. 따라서 업계에서는 꼭 필요한 기능만을 갖춘 단순한 DB 엔진을 자체 개발하기도 합니다.

정보 구축과 검색을 위한 데이터베이스, 버클리 DB

소스가 공개되어 있는 초고속 경량 DB 엔진으로는 버클리(Berkeley) DB가 유명합니다. 버클리 DB는 키워드와 데이터의 쌍으로 자료를 처리하는 단순 내장형 데이터베이스 엔진으로 매우 빠르게 동작합니다. C/C++ 에서 사용할 수 있으며 'www.sleepycat.com' 사이트에서 다운로드 할 수 있습니다. 또한 사용법이 쉽고 온라인 매뉴얼도 잘 작성되어 있어 편리합니다. 우리가 주로 사용하게 될 함수들의 사용법을 간단히 살펴 보겠습니다.

```
int db_create(DB **dbp, DB_ENV *dbenv, u_int32_t flags);
```

db_create는 버클리 DB 데이터베이스에 접근하기 위한 핸들(handle)을 생성하는 함수입니다. 특별한 경우가 아니라면 'db_create(&dbp, NULL, 0);' 을 사용하면 됩니다.

```
int DB->open(DB *db, const char *file, const char *database, DBTYPE type, u_int32_t flags, int mode);
```

현재 버클리 DB에서 지원하는 DB의 자료구조로는 B-tree, Hash, Queue 등이 있습니다. 정보 검색엔진을 위한 자료구조로는 B-Tree가 시간 및 공간 사용에 있어 가장 효율적입니다. type 변수의 값으로는 DB_BTREE, DB_HASH, DB_QUEUE, DB_RECNO, DB_UNKNOWN을 사용할 수 있습니다. type 값이 DB_UNKNOWN이고 데이터베이스 파일이 이미 존재할 경우, 'DB → open' 은 데이터베이스 타입을 자동으로 인식합니다.

flags와 mode는 파일이 아직 존재하지 않을 때 어떻게 생성하고 열 것이지 지정하는 인수입니다. flags로 사용할 수 있는 값으로는 DB_CREATE, DB_RDONLY, DB_TRUNCATE 등이 있습니다. 직감적으로 알 수 있듯이 DB_CREATE는 DB를 생성합니다. 이미 파일이 존재하고 있으면 기존의 파일을 열어 작업합니다. 파일이 존재하지 않는데도 DB_CREATE를 지정하지 않으면 에러가 발생합니다.

DB_RDONLY는 DB를 읽기 전용으로 여는 것이고, DB_TRUNCATE는 기존의 내용을 전부 삭제하고 시작하도록 하는 것입니다. 여러 가지를 동시에 지정하고 싶으면 'bitwise OR(|)' 을 사용하면 됩니다. mode 값은 유닉스 계열에서 권한을 설정할 때 사용하는 값과 같습

리스트 1 · 버클리 DB에 자료 입력하기

```
#include <sys/types.h>
#include <stdio.h>
#include <db.h>

#define DATABASE "access.db"

main()
{
        DB *dbp;                /* DB 핸들 */
        DBT key, data;          /* 키 값과 실제 데이터 저장을 위한 변수 */
        int ret;                /* 반환값을 받기 위한 임시 변수 */
        char *name = "김현우";
        char *email = "hwkim@yonsei.ac.kr";

        ret = db_create(&dbp, NULL, 0);
        if (ret)
        {
              fprintf(stderr, "db_create: %s\n", db_strerror(ret));
                exit(1);
        }
        ret = dbp->open(dbp, DATABASE, NULL, DB_BTREE, DB_CREATE, 0664);
        if (ret)
        {
                dbp->err(dbp, ret, "%s", DATABASE);
                exit(1);
        }

        /* key와 data를 먼저 초기화 하는 것이 매우 중요합니다 */
        memset(&key, 0, sizeof(key));
        memset(&data, 0, sizeof(data));

        /* key와 data를 입력 */
        key.data = name;
        key.size = strlen(name)+1;
        data.data = email;
        data.size = strlen(email)+1;

        ret = dbp->put(dbp, NULL, &key, &data, 0);
        if (ret)
        {
                dbp->err(dbp, ret, "DB->put");
                exit(1);
        }
        printf("db: %s: 키값이 저장되었습니다.\n", (char*)key.data);
        dbp->close(dbp, 0);
}
```

리스트 2 · 버클리 DB에서 자료 검색하기

```
#include <sys/types.h>
#include <stdio.h>
#include <db.h>
#define DATABASE "access.db"
main()
{
        DB *dbp;
        DBT key, data;
        int ret;

        ret = db_create(&dbp, NULL, 0);
        if (ret)
        {
              fprintf(stderr, "db_create: %s\n", db_strerror(ret));
                exit(1);
        }

        ret = dbp->open(dbp, DATABASE, NULL, DB_BTREE, DB_CREATE, 0664);
        if (ret)
        {
                dbp->err(dbp, ret, "%s", DATABASE);
                exit(1);
        }

        memset(&key, 0, sizeof(key));
        memset(&data, 0, sizeof(data));
        key.data = "김현우";
        key.size = sizeof("김현우");

        ret = dbp->get(dbp, NULL, &key, &data, 0);
        if (ret)
        {
                dbp->err(dbp, ret, "DB->get");
                exit(1);
        }
        printf("key값 '%s'에 대한 data는 '%s' 입니다.\n",
                        (char*)key.data, (char*)data.data);
        dbp->close(dbp, 0);
}
```

니다. 보통의 경우 'DB → open(dbp, DATABASE, NULL,DB_BTREE, DB_CREATE, 0664);' 로 사용하면 무리가 없습니다.

```
int DB->put(DB *db, DB_TXN *txnid, DBT *key, DBT *data, u_int32_t flags);
```

'DB → put' 함수는 키워드와 데이터 쌍을 데이터베이스에 저장하는 기능을 합니다. db는 데이터베이스 핸들이며, txnid는 트랜젝션 처리를 위한 것인데 보통은 NULL을 사용합니다. key와 data가 바로 키워드와 데이터 쌍을 나타냅니다.

flags는 새로 삽입하려는 자료가 이미 데이터베이스 내에 존재할 경우 어떻게 처리할 것인지를 지정하는 값으로 DB_APPEND, DB_NODUPDATA, DB_NOOVERWRITE 중 하나를 사용할 수 있습니다. DB_APPEND를 사용하면 키/데이터 중복이 허용되지만, DB_NODUPDATA를 지정하면 완전히 동일한 키/데이터가 이미 존재할 경우 에러 값을 반환합니다. DB_NOOVERWRITE는 동일한 key 값만 존재해도 에러 값을 반환하도록 합니다.

```
int DB->get(DB *db, DB_TXN *txnid, DBT *key, DBT *data, u_int32_t flags);
```

'DB → get' 함수는 데이터베이스로 부터 원하는 자료를 가져 오는 기능을 합니다. 즉, key 값을 기준으로 검색을 수행하여 데이터를 넘겨줍니다.

〈리스트 1〉과 〈리스트 2〉에서 사용한 'DB → get' 은 사실 하나의 자료만 가져올 수 있습니다. 동일한 key 값으로 여러 데이터가 저장되어 있을 경우에는 가장 처음 발견되는 데이터만을 가져 옵니다. 여러 데이터를 가져오기 위해서는 커서를 이용해야 합니다. 지면 관계상 자세한 내용을 다루기 어려우므로 이에 관해서는 생략하겠습니다.

자료 축적과 관리를 위한 MySQL 연동법

실제 검색이 이루어질 때는 빠른 속도를 내기 위해 버클리 DB를 이용하지만 자료 축적 및 관리를 위해서 MySQL에 원본 데이터를 보관합니다(1부 참조). MySQL 연동을 보다 쉽게 하기 위한 함수를 몇 개 소개합니다.

다음은 MySQL DB접속을 위한 함수로 사용자 ID와 비밀번호를 인수로 넘겨 주기만 하면 됩니다.

```
MYSQL mysql;        /* MySQL 핸들 (전역 변수) */
int DBconnect(const char* userid, const char* password)
{
        int ret;
        mysql_init(&mysql);
        if ( mysql_real_connect(&mysql, "localhost", userid, password, "robot",0,NULL,0) ) {
                printf("Connected to MySQL!\n");
                ret = 0;
        } else {
                fprintf(stderr,"Connection failed!\n");
                fprintf(stderr,"=> %s\n",mysql_error(&mysql));
                ret = -1;
        }
        return ret;
}
```

SQL 질의 문장을 MySQL로 보내는 함수입니다. 예를 들어 'DB query("select * from pginfo");' 를 수행하면 pginfo 테이블에 있

는 모든 자료가 출력될 준비가 끝납니다.

```
int DBquery(const char* query)
{
        int ret;
        ret = mysql_query(&mysql,query);
        if (ret) {
                fprintf(stderr,"ERROR: %s\n",query);
                fprintf(stderr,"    => %s\n",mysql_error(&mysql));
        }
        return ret;
}
```

DB 연결을 해제하기 위해서는 DBdisconnect를 사용하면 됩니다.

```
void DBdisconnect()
{
        mysql_close(&mysql);
        printf("Disconnected from MySQL!\n");
}
```

DB 연동 사용 예는 다음과 같습니다.

```
        MYSQL_RES* result;
        MYSQL_ROW  row;
        int max = 0;
          DBconnect("id","password");
        DBquery("select max(id) from pginfo");
        result = mysql_use_result(&mysql);
        row = mysql_fetch_row(result);
        if (row) max = atoi(row[0]) + 10;
        mysql_free_result(result);
          DBdisconnect();
        return max;
```

pginfo에서 id의 최대값을 알고자 'select max(id) from pginfo' 를 MySQL로 보냈습니다. 그러면 MySQL은 실제로 검색을 하여 검색 결과를 출력할 준비를 하게 됩니다. mysql_use_result를 이용하여 결과를 받을 준비를 하고, mysql_fetch_row를 통해 검색 결과를 한 행(row)씩 수신하게 됩니다. 행의 첫 번째 필드는 row[0], 두 번째 필드는 row[1], … 이런 식으로 배열에 값이 들어 가 있습니다. 검색 결과를 모두 받은 다음에는 mysql_free_result를 수행하여 초기화를 해 주어야 합니다.

zlib으로 압축 및 압축해제하기

zlib는 압축 및 압축해제 속도가 빠르고 압축률 또한 우수한 라이브러리 입니다. 하지만 프로그래머 입장에서는 사용하기가 약간 불편합니다. 압축된 데이터에 크기 정보가 들어가지 않아 프로그래머가 따로 처리해 주어야 하기 때문입니다. 그래서 필자는 압축된 데이터에 헤더를 붙여서 압축 해제에 필요한 정보를 포함하도록 하는 wrapping 함수를 만들어 보았습니다. 실행 화면은 다음과 같습니다.

```
$ gcc compress.c -o compress -lz
$ compress
source: [216] 죽는 날까지 하늘을 우러러 한 점 부끄럼이 없기를
              잎새에 이는 바람에도 나는 괴로워했다
              별을 노래하는 마음으로 모든 죽어가는 것을 사랑해야지
              그리고 나한테 주어진 길을 걸어가야겠다
              오늘밤에도 별이 바람에 스치운다
compressed: 195
uncompressed: [216] 죽는 날까지 하늘을 우러러 한 점 부끄럼이 없기를
              잎새에 이는 바람에도 나는 괴로워했다
              별을 노래하는 마음으로 모든 죽어가는 것을 사랑해야지
              그리고 나한테 주어진 길을 걸어가야겠다
              오늘밤에도 별이 바람에 스치운다
```

처음에 216바이트였던 문자열이 압축 후 195바이트로 줄었습니다. 다시 압축을 해제하자 216바이트로 돌아 왔으며 자료가 손상 없이 그대로 복원되었음을 알 수 있습니다. 위의 예에서는 문장의 길이가 그다지 길지 않아 좋은 압축률이 나타나지 않았지만 실제 HTML 문서를 압축해 보면 평균 3 : 1의 비율이 나타나는 것을 볼 수 있습니다.

문서를 파일로 다운로드 해주는 wget 유틸리티

웹 문서를 다운로드 하기 위해 HTTP 규약을 모두 공부하기에는 너무 부담스러울 것입니다. 우리가 또 하나의 인터넷 익스플로러를 만들 필요는 없겠지요? 다행히 매우 편리하게 사용할 수 있는 유틸리티가 있습니다. 주소만 알려 주면 해당 문서를 척척 파일로 다운로드 해주는 프로그램입니다. http 뿐만 아니라 ftp도 지원이 됩니다. 소스가 공개되어 있어 리눅스, 유닉스는 물론 윈도우에서도 사용 가능합니다. 리눅스에는 기본적으로 설치가 되는 경우가 많으므로 지금 바로 확인해 보기 바랍니다.

- **사용법 :** wget [옵션]… [URL(홈페이지주소)]…
- **사용 예 :** wget -O tmp www.pserang.co.kr

사용법	설명
-o, —output-file=FILE	출력되는 메시지를 파일에 저장한다.
-a, —append-output=FILE	출력되는 메시지를 파일에 저장하되 기존의 파일을 지우지 않고 계속 추가한다.
-q, —quiet	메시지를 출력하지 않는다.
-nv, —non-verbose	중요한 메시지만 출력하도록 한다.
-O —output-document=파일명	다운로드 받은 문서를 다른 이름으로 저장한다. -O를 사용하면 받은 문서를 stdout으로 출력한다.
-T, —timeout=숫자	읽기 시간 제한을 설정한다(초 단위).

표 1. wget 유틸리티 사용법

리스트 3 · zlib를 보다 편리하게 사용하기 위한 warpping 함수

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <zlib.h>

static char hex2int(char hex)
{
        if (hex >= '0' && hex <= '9') return hex - '0';
        if (hex >= 'a' && hex <= 'f') return hex - 'a' + 10;
        if (hex >= 'A' && hex <= 'F') return hex - 'A' + 10;
        return 0;
}

char* Compress(char* source, int sourceLen, int* retLength)
{
        uLongf i,destLen;
        char  size1h[sizeof(uLongf)+1],  size2h[sizeof(int)+1],  lenSizes,
        *buff, header[sizeof(uLongf)+sizeof(int)+2], headerLen;

       if (!source) {*retLength = -1; return (char*)NULL;}
       destLen = (sourceLen + 12) * 1.1 + 1;
       buff = (char*)malloc(destLen);

       switch(compress(buff,&destLen,source,sourceLen))
       {
             case Z_OK: break;
             case Z_MEM_ERROR:
                  fprintf(stderr,"compress: not enough memory to uncompress!\n");
                       free(buff); *retLength = -1;
                       return (char*)NULL;
             case Z_BUF_ERROR:
                  fprintf(stderr,"compress: not enough room in the output buffer!\n");
                  free(buff); *retLength = -1;
                  return (char*)NULL;
             default:
                  fprintf(stderr,"compress: an unknown error occured!\n");
                  free(buff); *retLength = -1;
                  return (char*)NULL;
        }
        sprintf(size1h,"%x",destLen);
        sprintf(size2h,"%x",sourceLen);
        lenSizes = (strlen(size1h) << 4) + strlen(size2h);
        sprintf(header,"%c%s%s", lenSizes,size1h,size2h);
        headerLen = strlen(header);
        *retLength = headerLen + destLen;
        buff = (char*)realloc(buff,*retLength);
        i = destLen;
        do { i--; buff[headerLen+i] = buff[i]; } while(i>0);
        for ( i = 0 ; i < headerLen ; i++ ) buff[i] = header[i];
        return buff;
}

char* Uncompress(char* source, int sourceLen, int* retLength)
{
        uLongf i = 0, j = 0, compressedLen = 0;
        uLong stringLen = 0;
        size_t buffsize;
        char *buff, lenSizes;

        if (!source) {*retLength = -1; return (char*)NULL;}
        lenSizes = source[i++];

        for (j = i + ((lenSizes & 0xf0) >> 4) ; i < j ; i++)
             compressedLen = (compressedLen << 4) + hex2int(source[i]);

        for (j = i + (lenSizes & 0x0f) ; i < j ; i++)
             stringLen = (stringLen << 4) + hex2int(source[i]);

        if (sourceLen != i + compressedLen) {
             fprintf(stderr,"str_uncompress: broken header!\n");
             *retLength = -1;
        return (char*)NULL;
        }

        buff = (char*)malloc(stringLen+1);
        switch(uncompress(buff, &stringLen, &source[i], compressedLen))
        {
             case Z_OK: break;
             case Z_MEM_ERROR:
                  fprintf(stderr,"uncompress: not enough memory to uncompress!\n");
                       free(buff); *retLength = -1;
                       return (char*)NULL;
             case Z_BUF_ERROR:
                  fprintf(stderr,"uncompress: not enough room in the output buffer!\n");
                       free(buff); *retLength = -1;
                       return (char*)NULL;
             case Z_DATA_ERROR:
                  fprintf(stderr,"uncompress: input data corrupted!\n");
                       free(buff); *retLength = -1;
                       return (char*)NULL;
             default:
                  fprintf(stderr,"compress: an unknown error occured!\n");
                       free(buff); *retLength = -1;
                       return (char*)NULL;
        }
        buff[stringLen] = '\0';
        *retLength = stringLen;
        return buff;
}

main()
{
```

```
        char *a, *b, *c;
        int len,len2;

        a = "죽는 날까지 하늘을 우러러 한 점 부끄럼이 없기를
             잎새에 이는 바람에도 나는 괴로워했다
             별을 노래하는 마음으로 모든 죽어가는 것을 사랑해야지
             그리고 나한테 주어진 길을 걸어가야겠다
             오늘밤에도 별이 바람에 스치운다";

        printf("source: [%d] %s\n",strlen(a),a);

        b = Compress(a,strlen(a),&len);
        printf("compressed: %d\n",len);

        c = Uncompress(b,len,&len2);
        printf("uncompressed: [%d] %s\n",len2,c);
}
```

맨 윗 부분의 o는 소문자이고 볼드체로 되어 있는 부분은 대문자 O임에 주의하세요. 대문자 O옵션을 왜 굵은 글씨로 해 놓았냐구요? 제일 중요한 부분이기 때문입니다. 파일명을 지정해 주지 않으면 제각기 다른 이름으로 받아지기 때문에 관리하기가 어렵습니다. 어차피 검색엔진 내에서는 DB에 저장이 되기 때문에 파일 이름이 필요 없습니다. 따라서 일관된 이름으로 이를테면 tmp라는 이름으로 다운로드하고, 처리 과정을 거친 후에는 그 파일을 삭제하는 방식을 택하는 것이 편리합니다. 사용 예와 같이 입력을 하면 프로그램 세계 홈페이지 첫 화면의 HTML코드가 tmp라는 파일로 저장이 됩니다.

실전! 검색엔진 만들기

지금까지 검색엔진의 이론적 배경과 실제 개발에 필요한 도구들에 대해서도 알아 보았습니다. 이제 실제 검색엔진 소스를 분석하며 이해해 보겠습니다.

Step 1 웹페이지를 모두 방문하는 탐색 알고리즘 구현

탐색 알고리즘(Traversing algorithm)이란 방향성 그래프(directed graph) 구조로 되어 있는 사이트의 웹페이지들을 모두 방문하기 위한 방법을 말합니다. 크게 두 가지 방법이 있습니다. 하나는 재귀호출을 이용하는 방법이고 또 하나는 큐(Queue) 자료구조를 이용하는 방법입니다. 재귀적 호출 방법을 사용할 경우에는 주의를 많이 기울여야 합니다. 깊이(depth)가 깊은 사이트를 탐색할 경우 자칫 잘못하면 스택 오버플로(stack overflow) 오류가 생길 수 있기 때문입니다. 따라서 깊이의 최대 값을 지정해 주어야 합니다. 구현하기가 간단한 반면 시스템에 많은 부담을 주는 단점이 있습니다. 〈그림 1〉과 같은 구조를 가진 사이트가 있다고 가정할 때, 노드(node)의 방문 순서는 'main → sub1 → sub4 → sub5 → sub7 → sub2 → sub3 → sub6' 입니다. 한번 방문한 곳은 다시 가지 않도록 처리해야 합니다.

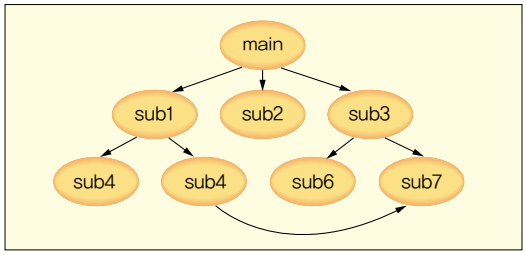

그림 1. 웹페이지들 간의 상호 연결 관계

큐 자료구조를 이용한 방법은 재귀적 호출 방법의 단점을 보완하기 위해 고안한 것으로, 웹문서들을 탐색하면서 발견되는 모든 전방링크(forward links)를 URL 큐에 삽입하는 방법입니다. 〈그림 1〉에 이 방법에 적용할 경우 방문 순서는 'main → sub1 → sub2 → sub3 → sub4 → sub5 → sub6' 이 됩니다.

대량의 페이지를 가진 사이트를 탐색할 경우 큐가 매우 커지게 되고 그럴 경우 메모리가 부족하게 되어 URL 서버는 물론이고 같은 서버에서 돌아가고 있는 다른 응용 프로그램들의 수행 능력도 현저히 떨어뜨릴 수 있습니다. 따라서 하드디스크를 적절히 사용해야 합니다. 그렇다고 하드디스크에만 저장을 하면 I/O 병목 현상이 생기게 되므로 메모리와 하드디스크를 적절히 사용하는 지혜가 필요합니다.

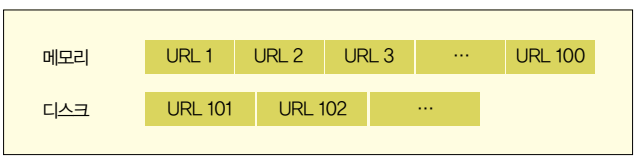

그림 2. URL 큐의 개념

Step 2 URL을 정밀 분석하는 URL 처리기 만들기

URL 처리기(URL Resolver)는 상대 주소를 절대 주소로 변환해 주고, URL을 정밀 분석하여 호스트 이름과 서브넷 이름, 도메인 확장자, 포트 번호 등의 정보를 분리해 냅니다. 자세한 내용은 1부에서 이미 다루었으니 소스를 분석하면 이해하는데 그다지 어렵지 않을 것입니다. 원래 URL 처리기는 스트링 처리에 많은 정성을 들여야 하는 부분입니다. 그만큼 에러가 발생하기 쉬운 부분이기도 합니다.

지면 관계상 URL 처리기에 관한 소스는 프세 홈페이지를 통해 제공하겠습니다. 이 소개하는 소스는 수년간 사용된 것으로 어느 정도 안정성이 보장되어 있다 할 수 있습니다. 유용하게 사용하기 바랍니다.

리스트 4 · URL 큐 소스

```
// URL QUEUE (CIRCULAR QUEUE)
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include "queue.h"
#include "hstring.h"
#define CQ_CELLSIZEMAX 602
#define CQ_EXTFILE "_url_cq.extended.tmp"
void CircularQueueClass::print_queue()
{ // 큐의 사용 현황을 보여준다(테스트용).
        FILE* f_cq;
        char buff[1000];
        for (int i=0 ; i< cq_max; i++) {
                if (cq[i]==NULL) printf("NULL  ");
                else printf("%4s  ",cq[i]);
        }
        printf("\n");
        f_cq = fopen(CQ_EXTFILE,"r");
        if (f_cq) {
                printf("[DISK]-------------------------\n");
                while(fgets(buff,1000,f_cq))
                        fputs(buff,stdout);
                printf("------------------------------\n");
                fclose(f_cq);
        }
        printf("\n");
}

CircularQueueClass::CircularQueueClass(int mem_maxcells)
{ // 클래스 생성자 : 초기화를 수행한다.
        cq_max = mem_maxcells;
        cq = (char**)malloc( sizeof(char*) * cq_max );
        memset(cq,(char)NULL,sizeof(char*) * cq_max );
        FILE* f_cq = fopen(CQ_EXTFILE,"w");
        fclose(f_cq);
        remove(CQ_EXTFILE);
        cq_front = cq_rear = 0;


}
CircularQueueClass::~CircularQueueClass()
{ // 클래스 소멸자
      int i;
        for (i=0 ; i< cq_max; i++)
                if (cq[i]) free(cq[i]);

        free(cq);
        FILE* f_cq = fopen(CQ_EXTFILE,"w");
        fclose(f_cq);
        remove(CQ_EXTFILE);
}

int CircularQueueClass::enqueueDisk(char* s)
{ // 메모리에 있던 URL들을 디스크로 옮긴다.
        // FILE SIZE LIMIT : 2GB (1024^3 * 2 = 2,147,483,648)
        long filesize;
        FILE* f_cq = fopen(CQ_EXTFILE,"r");
        if (f_cq) {
                fseek(f_cq,0,SEEK_END);
                filesize = ftell(f_cq);
                // 만약 파일 크기가 2G가까이 되면 더 이상 기록하지 않습니다.
                if (filesize > 2000000000) { fclose(f_cq); return 0; }
                fclose(f_cq);
        }
        //printf("enqueueing(disk) %s ...\n",s);
        f_cq = fopen(CQ_EXTFILE,"a");
        fprintf(f_cq,"%s\n",s);
        fclose(f_cq);
        return 1;
}

int CircularQueueClass::loadDisk()
{ // 디스크에 있는 URL을 메모리로 읽어 들인다.
        long filesize, backoffset;
        FILE* f_cq = fopen(CQ_EXTFILE,"r");
        if (!f_cq) { cq_diskmode = 0; return 0; }
        fseek(f_cq,0,SEEK_END);
        filesize = ftell(f_cq);
        if (filesize == 0) { fclose(f_cq); remove(CQ_EXTFILE); cq_diskmode = 0; return 0; }

        char* buff = (char*)malloc(500);
        fseek(f_cq,0,SEEK_SET);
        cq_diskmode = 2;
        for (int i=0; i < cq_max; i++) {
                backoffset = ftell(f_cq);
                if (fgets(buff,CQ_CELLSIZEMAX,f_cq)==NULL) break;
                buff[strlen(buff)-1] = '\0';
                //printf("read from disk(to mem): %s\n",buff);
                if (!enqueue(buff)) {
                        //printf("back to disk: %s\n",buff);
                        fseek(f_cq,backoffset,SEEK_SET);
                        break;
                }

        }
        cq_diskmode = 1;
        if (!feof(f_cq)) {
                FILE* f_rcq = fopen("REMAINED_CQ.TMP","w");
                while(fgets(buff,CQ_CELLSIZEMAX,f_cq)) {
                    //printf("read from disk(to disk): %s\n",buff);
                        fputs(buff,f_rcq);
                }
                fclose(f_rcq);
```

```
                fclose(f_cq);
                remove(CQ_EXTFILE);
                rename("REMAINED_CQ.TMP",CQ_EXTFILE);
        }
        else {
                fclose(f_cq);
                remove(CQ_EXTFILE);
                cq_diskmode = 0;      // disable the disk queue mode
        }
        free(buff);
        return 1;
}

int CircularQueueClass::enqueue(char* s)
{
        if (cq_diskmode == 1) {
                if (!enqueueDisk(s)) return 0;
                return 1;
        }

        // FULL CONDITION
        if ( cq_rear + 1 < cq_max ) {
                if ( cq_rear + 1 == cq_front ) {
                        if (cq_diskmode == 2) return 0;
                        cq_diskmode = 1;
                        if (!enqueueDisk(s)) return 0;
                        return 1;
                }
        }
        else {
                if ( cq_front == 0 ) {
                        if (cq_diskmode == 2) return 0;
                        cq_diskmode = 1;
                        if (!enqueueDisk(s)) return 0;
                        return 1;
                }
        }

        // STRING LENGTH LIMIT
        if ( CQ_CELLSIZEMAX != 0 && strlen(s) > CQ_CELLSIZEMAX ) return 0;

        if ( ++cq_rear >= cq_max ) cq_rear = 0;
        //printf("enqueueing %s...\n",s);
        cq[cq_rear] = (char*)malloc(strlen(s)+1);
        if ( cq[cq_rear] == NULL ) {
                perror("QUEUE 메모리 부족!");
                if ( cq_rear ) cq_rear--; else cq_rear = cq_max - 1;
                return 0;
        }
        strcpy(cq[cq_rear],s);
        /*
        printf("%s has now been enqueued! at %d\n",s,cq_rear);
        printf("front = %d , rear = %d\n",cq_front,cq_rear);
        */
        return 1;
}

int CircularQueueClass::dequeue(char* ret, int maxlen)
{
        if ( cq_rear == cq_front && cq_diskmode == 1 ) {
                if (!loadDisk()) return 0;
        }
        if ( cq_rear == cq_front) return 0;
        if ( ++cq_front >= cq_max ) cq_front = 0;

        strcpy(ret, cq[cq_front],maxlen);
        free(cq[cq_front]);
        cq[cq_front] = NULL;
        /*
        printf("%s has now been dequeued! at %d\n",ret,cq_front);
        printf("front = %d , rear = %d\n",cq_front,cq_rear);
        */
        return 1;
}

/* 테스트용 메인 함수
main()
{
        class CircularQueueClass CQ(4);
        char buff[100];
        while(1) {
                printf("INPUT>> ");
                if (fgets(buff,100,stdin)==NULL) break;
                buff[strlen(buff)-1] = '\0';
                if (!strcmp(buff,"deq")) {
                if ( CQ.dequeue(buff,99) == 0 ) printf("CANNOT DEQUEUE! (EMPTY)\n");
                        else printf("DEQUEUED: %s\n",buff);
                        CQ.print_queue();
                        continue;
                }
                if (!CQ.enqueue(buff)) printf("CANNOT ENQUEUE! (FULL)\n");
                CQ.print_queue();
        }
}
*/
```

Step 3 문서를 검색할 수 있는 색인기 만들기

색인기 동작 과정의 이해를 돕기 위해 단순한 예를 들어보겠습니다. 문서가 두 개 있다고 가정합시다. 그리고 이 문서를 검색할 수 있도록 하기 위해 색인을 하려고 합니다. 그냥 문자열 비교를 하면 되지 왜 색인을 하냐고 반문할지 모르지만 이것은 그저 단순한 예에 불과할 뿐입니다. 수십 내지 수백 기가에 달하는 문서를 순차적으로 문자열 비교를

하자면 어마어마한 시간이 걸립니다. 많은 사용자가 몰려 와도 0.1초 이내에 결과를 내야만 하는 상황임을 염두에 두기 바랍니다.

[문서 1] 처음인걸요 분명한 느낌 놓치고 싶지 않죠 사랑이 오려나 봐요
[문서 2] 사랑해 널 잊을 순 없을 거야 미안해 널 지키지 못 한 것을

❶ **바이그램(bigram) 생성 :** 형태소 분석기를 이용하여 문장에서 추출된 명사를 색인어로 사용하는 것이 검색 품질을 높이는데 도움이 되지만 형태소 분석기는 수 천 만원에 달하는 고가의 소프트웨어입니다. 형태소 분석기를 사용하지 않고도 바이그램을 이용하여 색인하는 방법에 대해 알아 보기로 하겠습니다. 다음은 두 문서를 바이그램으로 변환한 것입니다.

[문서 1] #처 처음 음인 인걸 걸요 요분 분명 명한 한느 느낌 낌놓 놓치 치고 고싶 싶지 지않 않죠 죠사 사랑 랑이 이오 오려 려나 나봐 봐요 요#

[문서 2] #사 사랑 랑해 해널 널잊 잊을 을순 순없 없을 을거 거야 야미 미안 안해 해널 널지 지키 키지 지못 못한 한것 것을 을#

원래는 문서에서의 출현 빈도를 계산해야 하지만 이 예에서는 거의 동일한 출현 빈도가 나타나므로 생략했습니다. 출현 빈도란 문서에서 해당 단어가 문서에서 얼마나 빈번하게 나타나는지를 계산한 것입니다.

❷ **역색인(inverted index) 생성 :** 색인어가 어떤 문서에서 나타나고 있는지의 정보를 구축해야 합니다. 바이그램 분석시에는 문서에 어떤 색인어가 있는지를 알아냈었는데 이와는 정 반대의 작업이라 할 수 있지요. 아래의 결과물을 봅시다.

```
#처 1
처음 1
음인 1
...
사랑 1 2
랑이 1
...
(생략)
```

색인어가 어느 문서에 등장하고 있는지 나타내고 있다는 것을 알 수 있습니다. '사랑' 이라는 색인어는 문서 1과 문서 2에서 모두 등장하고 있습니다.

색인기에 대해서는 간단한 예만을 들어 보았는데 사실 실제 정보 검색 시스템에서는 이렇게 간단하지 않습니다. 용량 절약을 위해 어휘 사전을 구축하여 이를 참조하도록 합니다. 또한 색인어가 문서에서 얼마나 빈번하게 나타나는지를 계산하여 가중치를 갖도록 합니다. 지면 관계상 소스 코드를 싣지 못하는 점을 양해 바라며 프세 홈페이지를 통해 제공되는 소스 코드를 참조하기 바랍니다.

Step 4 웹문서 수집기 관리자 만들기

한 개의 웹문서 수집기 프로세스만을 가동하는 것은 비효율적입니다. 어떤 사이트의 경우 전송 속도가 매우 느리거나 일시적으로 지연 시간이 길어질 수 있는데, 이럴 경우 시간당 페이지 수집양이 급격히 떨어지게 되기 때문입니다. 네트워크 자원은 일정한데 사용량이 적으므로 결국 많은 자원이 쓰이지 않게 되는 것입니다.

이를 보완하기 위해 작업 관리자를 만들 필요가 있습니다. 멀티스레드 라이브러리(Multi-thread library)를 사용해 웹문서 수집기를 10~30개 정도 생성하여 각기 다른 사이트를 동시에 수집하도록 하는 것이 좋습니다. 프로세스 대신 스레드를 생성하는 것은 시스템 자원을 절약하기 위해서입니다. 자식 프로세스(child process)를 생성할 때는 부모 프로세스(parent process)의 데이터 메모리 영역이 자식 프로세스에게 그대로 복사되는데 이는 고비용의 연산입니다. 스레드는 경량의 프로세스(lightweight process)라고도 불리며, 스레드의 생성은 일반 프로세스 보다 적게는 10배에서 많게는 100배까지 빠릅니다. 게다가 많은 정보를 서로 공유하게 되므로 메모리를 효율적으로 사용할 수 있습니다.

지금까지 간단하게 검색엔진 소스를 분석해 보았습니다. 이 분야에 관심이 많은 독자라면 의문도 많이 생겼을 것입니다. 필자에게 메일로 문의하면 성의껏 답변해 드리겠습니다. 그럼 여러분도 멋진 검색엔진 제작에 성공하기를 기원하면서 이상으로 글을 마칩니다. PW

소스 제공 IR.zjp

참고 자료

1. ARANES 정보검색 엔진을 위한 Web Crawler 설계, 김현우, 한영석, 한국인터넷정보학회 2001년 춘계학술발표대회 논문집
2. Modern Information Retrieval, ACM Press
3. Advanced Programming in the UNIX Environment, Addison-Wesley
4. UNIX Network Programming. Volume 1. 2nd ed, Prentice Hall

정리 • 김상연 기자 sykim@pserang.co.kr