개선된 LZW 사전 관리 기법에 기반한 효과적인 Hyper-Text 문서 압축 방안

신 광 철[†] · 한 상 용[††]

요 약

LZW 알고리즘은 사전(dictionary) 방식의 압축 알고리즘인 LZ78의 변형된 형태로써 높은 압축률을 제공하기 때문에 많은 상업용 압축 프로그램에서 사용되고 있다. LZW의 핵심은 가장 빈번하게 사용되는 문자열을 사전에 저장하면서, 저장된 것과 동일한 입력 데이터의 문자열을 해당 사전 인덱스로 대체하는 것이다. 본 논문에서는 자주 사용되지 않는 사전의 엔트리를 찾기 위해 카운터를 이용하는 새로운 사전 관리 방법을 제안한다. 또한 하이퍼텍스트 문서를 효율적으로 압축하기 위해 태그와 같은 자주 사용되는 용어들을 코드화하여 사전에 저장한 후 압축을 시도하는 방안을 제안한다. 실험결과 기존의 V.42bis와 UNIX의 compress에 비해 상대적으로 일반문서는 3~8%, HTML 문서는 23~24%의 향상된 압축효과를 보여준다.

Hyper-Text Compression Method Based on LZW Dictionary Entry Management

Kwangcheol Shin[†] · Sangyong Han[††]

ABSTRACT

LZW is a popular variant of LZ78 to compress text documents. LZW yields a high compression rate and is widely used by many commercial programs. Its core idea is to assign most probably used character group an entry in a dictionary. If a group of character which is already positioned in a dictionary appears in the streaming data, then an index of a dictionary is replaced in the position of character group. In this paper, we propose a new efficient method to find least used entries in a dictionary using counter. We also achieve higher compression rate by preassigning widely used tags in hyper-text documents. Experimental results show that the proposed method is more effective than V.42bis and Unix compression method. It gives 3~8% better in the standard Calgary Corpus and 23~24% better in HTML documents.

키워드 : LZW, 데이터 압축(Data Compressioin), 사전방식 압축(Dictionary-based Compression), 하이퍼텍스트(Hyper-text), 확장 초기 사전(Expanded Initial Dictionary)

1. 서 론

데이터 압축은 컴퓨팅 분야에서 가장 중요한 연구주제 중의 하나이며 텍스트 문서로부터 시작하여 영상, 소리의 압축까지 다양한 형태의 기법들이 연구되어 왔다. 특히, 네트워크를 통한 데이터의 전송이 보편화되면서 더욱 부각이 되었다[1].

한편, 인터넷이 우리의 삶에 밀접하게 다가옴으로써 인터넷 문서를 효율적으로 검색하기 위한 방법들이 다양하게 모색되고 있다. 이미 많은 검색 엔진들이 방대한 양의 인터넷 문서들을 저장하고 있으면서 사용자들의 검색 질의에 대한 서비스를 제공하고 있으며 여기에 필요한 다양한 기술들이 개발되고 있다. 이때 방대한 인터넷 문서들을 저장하고 검색하기 위해서는 반드시 압축을 통해 많은 문서들을 저장하고 또한 빠르게 재생할 수 있어야만 하며 이를 위해 다양한 압축 기법들이 개발되어 쓰이고 있다[2].

† 준 회 원 : 중앙대학교 대학원 컴퓨터공학과
†† 종신회원 : 중앙대학교 컴퓨터공학과 교수
논문접수 : 2002년 4월 3일, 심사완료 : 2002년 8월 21일

Lempel과 Ziv가 개발한 LZ77[3], LZ78[4]은 사전을 이용하는 압축기법을 사용하는 대표적인 알고리즘이다. 그들이 이 논문을 발표하고 나서부터 압축을 위해 사전을 이용하는 방법이 다각적으로 연구가 되었고 계속적으로 더 나은 성능을 나타내는 수많은 알고리즘이 발표되었다. 그것의 대표적인 예가 LZW 알고리즘[5]이다. 이 방법은 사전에 미리 모든 문자와 기호들을 저장해 놓고 압축을 시작함으로써 사용되는 데이터의 요소를 더욱 압축하는 효과를 보여주고 있다[6].

그 후로 LZW를 기반으로 한 프로그램들이 개발되었고, 대표적인 예로 UNIX의 compress[7]가 있으며 또한 모뎀 통신 프로토콜인 V.42bis[8]에서도 이용되고 있다[1].

본 논문에서는 카운터를 이용한 적응성이 강화된 사전 관리기법과 HTML 문서와 같이 같은 용어가 반복되는 문서를 위한 초기사전 확장 개념에 기반을 둔 새로운 방법을 제시하고 이를 구현하여 기존의 방법과 비교하였다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 LZW 알고리즘에 대해 설명하고, 3장에서는 LZW를 사용하는 예

로써 UNIX의 compress와 모뎀 통신 프로토콜인 V.42bis에서의 사전관리 기법에 대해 알아본다. 4장에서는 본 논문에서 제시하는 사전관리 기법과 초기 사전 확장 기법에 대해 기술하고, 5장에서는 각각의 성능평가를 다양한 데이터를 통해서 검증해 본다. 마지막으로 6장에서 결론을 기술한다.

2. LZW 알고리즘

LZW 알고리즘[5]은 LZ78의 대표적인 개량으로서 T. Welch에 의해 1984년에 개발되었다. LZW의 기초가 되는 알고리즘인 LZ78은 압축 화일에 사전을 가리키는 포인터와 입력되는 문자로 이루어진 토큰이 기록된다(<표 1>참조). LZW는 기록되는 토큰의 두 번째 필드를 제거한 것으로, LZW의 토큰은 사전을 가리키는 포인터로만 구성되어 있다. LZW는 모든 문자를 사전에 미리 기록해 놓고 압축을 시작한다. 즉, 8bit 문자를 사용할 경우, 사전의 처음 256개는 데이터가 입력되기 전에 미리 채워져 있는 것이다.

〈표 1〉 LZ78의 토큰

사 전	토 큰	사 전	토 큰
0 null		8 'a'	(0, 'a')
1 's'	(0, 's')	9 'st'	(1, 't')
2 'i'	(0, 'i')	10 'm'	(0, 'm')
3 'r'	(0, 'r')	11 'an'	(8, 'n')
4 '_'	(0, '_')	12 '_ea'	(7, 'a')
5 'si'	(1, 'i')	13 'sil'	(5, 'l')
6 'd'	(0, 'd')	14 'y'	(0, 'y')
7 '_e'	(4, 'e')	15 '_t'	(4, 't')

LZW의 원리는 다음과 같다. 부호기(encoder)는 문자를 하나씩 받아들이고 문자열 변수 I에 저장한다. I에 저장된 값이 사전에서 발견되면 계속해서 다음 문자를 읽고 I에 결합시킨다. 다음 문자 x가 입력되면 I에 붙인 후 사전에서 I값이 존재하지 않게 되면, (즉, 사전에 I는 존재하는데 Ix가 존재하진 않는다.) 그 때 부호기는 (1) 문자열 I를 가리키는 사전 인덱스를 출력하고 (2) Ix를 사전의 빈 공간에 저장한 후 (3) I를 x로 초기화한다.

"sir sid eastman easily
teases sea sick seals"

예를 들어 위와 같은 문장이 있을 때 압축하는 과정은 다음과 같다.

1) 0-255까지의 사전 엔트리를 초기화한다.
2) 첫 문자 's'를 입력으로 받아서 사전에 있는지 검사한다. (아스키코드의 경우 115번째 엔트리에 s가 저장되어 있다.) 다음 문자 'i'를 받아서 'si'가 사전에 있는지 검사한다. 사전에 존재하지 않으므로 (1) 115를 출력하고, (2) 문자열 'si'를 사전의 빈 엔트리에 저장한다. (이 경우 비어있는 256번째 엔트리에 저장한다.) (3) I를 문자 'i'로 초기화한다.
3) 'sir'의 'r'이 입력으로 받은 후에 'ir'이 사전에 있는지 검사한다. 사전에 'ir'이 없기 때문에 부호기는 (1) 'i'에 해당하는 아스키코드 105를 출력하고 (2) 'ir'를 사전의 빈 곳(257번)에 저장한다. (3) I를 'r'로 초기화한다.

〈표 2〉 LZW 사전

0	NULL	107	k	256	si	272	ly
1	SOH	108	l	257	ir	273	y_
.		109	m	258	r_	274	_t
.		110	n	259	_s	275	te
.		.		260	sid	276	eas
32	SP	.		261	d_	277	se
.		.		262	_e	278	es
.		115	s	263	ea	279	s_
.		116	t	264	as	280	_se
97	a	.		265	st	281	_si
98	b	.		266	tm	282	ic
99	c	.		267	ma	283	ck
100	d	121	y	268	an	284	k_
101	e	.		269	n_	285	_sea
.		.		270	_es	286	al
.		.		271	sil	287	ls
.		255					

<표 2>는 LZW 사전을 초기화한 후 위의 데이터를 가지고 처리하였을 경우의 사전 내용을 보여주고 있다. 완전한 출력 데이터는 다음과 같다.(숫자만이 출력되는 것으로 괄호 안의 문자는 이해를 돕기 위한 것이다.)

115(s), 105(i), 114(r), 32(_), 256(si), 100(d), 32(_), 101(e), 97(a), 115(s), 116(t), 109(m), 97(a), 110(n), 262(_e), 256(si), 108(l), 121(y), 32(_), 116(t), 263(ea), 115(s), 101(e), 115(s), 259(_s), 263(ea), 259(_s), 105(i), 99(c), 107(k), 280(_se), 97(a), 108(l), 115(s), eof

〈표 3〉 LZW 알고리즘

```
for i := 0 to 255 do
    append as a 1-symbol string to the dictionary ;
append λ to the dictionary ;
di := dictionary index of λ ;
repeat
    read(ch) ;
    If << di, ch >> is in the dictionary then
        di := dictionary index of << di, ch >> ;
    else
        output (di) ;
        append << di, ch >> to the dictionary ;
        di := dictionary index of ch ;
    endif ;
until end-of-input ;
```

<표 3>은 알고리즘의 의사 코드를 나타낸 것이다. 여기서 λ는 빈 문자열을 나타내는 것이며, <<a, b>>는 문자열

a와 b를 결합하는 것을 나타낸다.

사전의 처음 256개의 엔트리는 시작할 때 채워지기 때문에 사전을 가리키는 포인터는 8bit보다 길어야 한다. 가장 간단한 구현은 16bit 포인터를 사용하는 것이며 이것은 64K 엔트리를 가지는 사전을 포인트 할 수 있다. 그러나 어떤 경우든 LZW의 사전은 매우 빨리 가득차게 되며 이에 대한 해결책을 다음과 같이 생각해 볼 수 있다[1].

1) 사전이 가득찬 시점에서 사전을 동결(freeze)시킨다. 사전은 정적인 상태가 되나 여전히 입력을 부호화(encode)할 수 있다.
2) 사전이 가득차게 되면 사전의 모든 엔트리를 비우고 처음부터 다시 시작한다. 이 방법은 문서가 다양한 형태의 블록으로 나뉘어 있을 경우 좋은 효과를 볼 수 있다.
3) UNIX의 compress 프로그램이 사용하는 방법이 있다.
4) 사전이 가득차게 되었을 때 가장 사용되지 않는 엔트리를 삭제하여 새로운 문자들을 받아들일 수 있도록 한다.

이 중 3)번과 4)번이 상용화된 방법에 쓰이고 있으며 다음 장에서는 이들 방법에 대해 자세히 살펴본다.

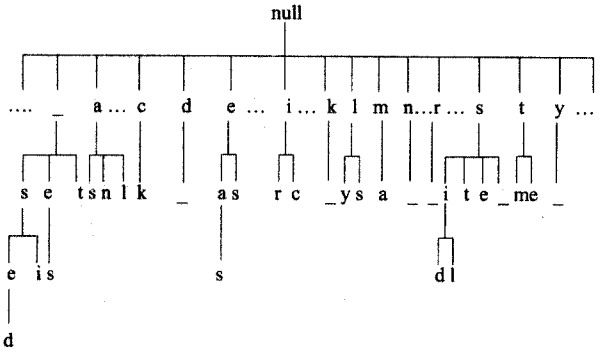
3. 사전 관리 기법

3.1 UNIX compress

compress[7]는 데이터를 압축하기 위해 사용되는 UNIX 프로그램으로서 사전의 크기를 변경할 수 있도록 만든 LZW 알고리즘에 기반을 두고 있다. 처음에 2^9(512)개의 엔트리를 가지는 사전으로 시작하고, 이때의 포인터의 크기는 9bit가 된다. 사전이 가득 차게 되면 사전의 크기를 두 배로 늘린다. 즉, 1024개의 엔트리를 갖도록 하며 포인터 또한 늘어나 10bit가 된다. 이런 방식으로 사전을 늘려가다가 사용자가 정한 크기(기본 값은 16bit)에 이르면 사전의 크기를 더 이상 늘리지 않고 압축 비율을 감시하고 있다가 미리 정한 일정한 값 이하로 압축률이 떨어지면 사전을 삭제하고 다시 512개의 엔트리를 가지는 새로운 사전으로 압축을 시작한다. 이와 같이 함으로서 사전이 '너무 낡지'(too out of date) 않게 되도록 유지한다.

3.2 V.42bis 프로토콜

V.42bis 프로토콜[8]은 빠른 모뎀(fast modem)을 사용하기 위해 ITU-T에서 제정한 표준이다. V.42bis에서는 일반(transparent)모드와 압축(compressed)모드를 명시하고 있다. 일반 모드는 주로 이미 압축된 문서를 전송하는데 쓰이는 것이며 압축모드는 사전의 크기를 증가시키며 압축을 수행하고 전송을 하는 것이다. 사전이 최대크기가 되고, 가득 차게 되면 V.42bis는 재사용 처리를 시작한다. 즉, 최근에 가장 사용되지 않았던 엔트리를 삭제하여 새로운 문구를 받을 수 있도록 한다. 이것은 256번째 엔트리부터 시작하여 다른 문구의 접두어가 아닌 것들을 삭제함으로 이뤄진다. 예를 들어 "abcd"라는 문구가 발견되었을 때 어떤 x에 대해 "abcdx"라는 문구가 없다면 이것은 "abcd"가 생성된 후 사용되지 않았다는 것을 의미하는 것이며 이것은 오래된 문구임을 보여주는 것이다. 따라서 이러한 문구를 제거함을 통해 새로운 패턴을 받아들일 수 있게 된다.



(그림 1) LZW의 사전을 Tree로 표현

즉, <표 2>의 사전을 (그림 1)과 같이 트리 형태로 표현했을 때 V.42bis 프로토콜의 사전 관리 기법은 모든 리프 노드(leaf node)들을 삭제한 후 공간을 확보하고 다시 압축을 수행하는 것이다.

4. 진보된 사전 관리 기법

우리는 압축 효율을 개선할 수 있는 방법 두 가지를 제시한다. 첫 번째는 V.42bis의 사전 관리 방식의 개선을 통한 것이며, 두 번째는 태그와 같이 동일한 단어가 문서 내에서 계속해서 반복되는 HTML문서 압축에 관한 것이다.

4.1 적응성이 강화된 사전 관리 기법

앞장에서 살펴본 바와 같이 사전을 관리하는 것은 압축 효율을 결정하는 중요한 요인이 된다.

본 장에서는 적응성이 강화된 새로운 사전 관리 방안을 제안한다. 먼저 새로운 방법과 관련된 개념과 원리를 설명한다.

정의 1 : i번째로 생성되는 리프 노드를 N_i이라고 하면, N_i은 일정한 카운트 값 t_i을 갖는다.

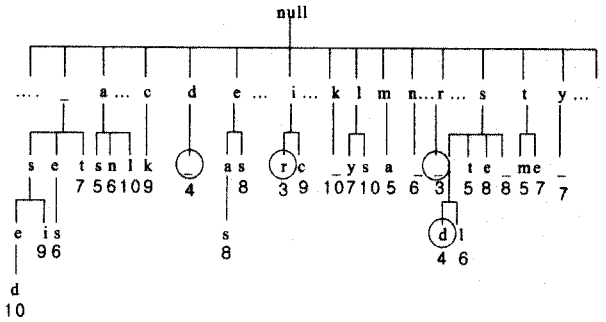
추론 1 : 일정 시간(t_p)마다 모든 리프 노드의 카운트 값을 1만큼 낮춘다면 t_p시간 후 생성되는 k번째 노드 N_k은 항상 기존의 리프 노드보다 더 높은 카운트 값을 갖는다.

추론 2 : 두 리프 노드 N_i와 N_j에 대해 t_i가 t_j보다 크다면 N_j에 비해 N_i가 최근에 사용된 것을 나타낸다.

이러한 사실을 바탕으로 새로운 알고리즘을 제안한다. 제시하는 알고리즘은 각각의 엔트리 중 가장 최근에 사용되

지 않았던 엔트리를 찾아 제거하는 방법이다.

이를 위해 각각의 엔트리에 카운터를 두어서 엔트리가 생성될 때 카운터를 어떤 일정한 값으로 정한 후 사전이 채워져 감에 따라 주기적으로 리프 엔트리의 카운터 값을 모두 1만큼 떨어뜨리고 사전이 가득 차게 되었을 때 모든 리프 엔트리 중 카운터 값이 가장 작은 것부터 순서적으로 제거해서 일정량의 공간을 확보한 후 계속해서 새로운 패턴을 저장한다.



(그림 2) 제안하는 방법의 사전 관리 방식

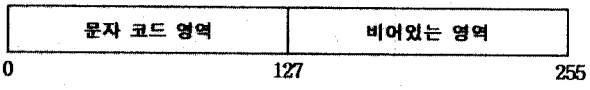
(그림 2)는 제시하는 알고리즘에 의해 <표 2>의 사전을 관리하는 경우이다. 각 리프 노드 아래의 숫자는 해당 노드의 카운터 값을 나타낸 것으로, 그림의 값은 초기값으로 10을 할당하고 4개의 리프 노드가 생성될 때마다 모든 리프 노드 값을 1만큼 떨어뜨렸을 때의 값을 보여주는 것이다. 그림에서 삭제 순서를 정하게 되면 가장 작은 카운트 값을 갖는 'r'과 '_'(공백)이 첫 순위가 되고 다음으로 작은 값을 갖는 '_'와 'd'가 다음 순위가 되는 것이다.

4.2 태그의 코드화를 통한 하이퍼텍스트 문서의 압축

본 논문에서 두 번째로 제안하는 방법의 아이디어는 기존의 LZW알고리즘이 모든 문자 코드를 저장하고 있으면서 압축을 수행하는 것과 하이퍼텍스트(HTML) 문서는 태그와 같은 동일한 단어가 자주 등장하는 것에 착안한 것이다. 즉, 처음부터 <표 4>와 같이 아스키코드가 끝나는 시점인 128번째부터 255번째까지 128개의 자주 나타나는 단어를 코드형태로 미리 저장해 놓는다. 실제로 LZW에서 128번째부터 255번째까지의 엔트리는 사용되지 않고 비어있기 때문에 그 영역에 다른 데이터를 넣고 압축을 수행해도 전체 사전의 크기에는 전혀 영향을 주지 않는다(그림 3).

〈표 4〉 확장된 LZW 초기 사전

0	NULL	98	b	136		153	*center*
1	SOH	99	c	137		154	*<script*
.		100	d	138	**	155	*type*
.		.		139	**	156	*link*
.		.		140	*<body>*	157	*rowspan =*
32	SP	.		141	*</body>*	158	*colspan =*
.		125	}	142	*<form>*	159	*height*
.		126	~	143	*</form>*	160	*width*
.		127	del	144	* *	161	*<table*
65	A	128	*<html>*	145	* *	162	*<tr>*
66	B	129	*</html>*	146	*<pre>*	163	*<td>*
67	C	130	*<head>*	147	*</pre>*	.	
68	D	131	*</head>*	148	*<meta*	.	
.		132	*<title>*	149	*name*	.	
.		133	*</title>*	150	*content*	253	*.gif*
.		134	*<a href=*	151	*left*	254	*.html*
97	a	135	**	152	*right*	255	*face*



(그림 3) 초기 사전의 사용 내역

또한 확장된 초기 사전은 아스키코드와 같이 미리 약속해서 이용할 수 있기 때문에 압축을 해제할 때도 추가적인 부담(overhead)은 없다.

압축과정에 대해 다음의 예를 통해 설명한다. <표 5>와 같은 HTML 문서가 있다고 가정하면 문서에서 자주 반복되는 용어는 "<a href=", "",""등과 같은 것이며, 이것은 확장된 초기 사전에 의하면 각각 134, 136, 135의 코드에 대응하는 것이다. 따라서 실제 출력되는 데이터는 134(<a href=), 115(s), 47(/), 50(2), 48(0), 49(1), 50(2), 62(>), 136(), 65(A), …가 된다. 이와 같이 확장된 초기 사전을 이용하여 HTML 문서를 효과적으로 압축할 수 있다.

〈표 5〉 Hyper-Text의 예

```
<a href = s/2012 > <b > Auctions </b > </a > - buy/sell anything -
<a href = s/2809 > Harry Potter </a >,
<a href = s/2757 > GameCube </a >,
<a href = s/2742 > Xbox </a >,
<a href = s/2758> PS2 </a >,
<a href = s/2782> Longaberger </a >,
<a href = s/2743> Dept 56 </a >,
<a href = s/2744> Barbie </a > </td > </tr > </table > </td >
    </tr > <tr > <td nowrap > <small > <b > Shop </b >   ;
<a href = r/a2 > Auctions </a > &#183 ;
<a href = r/cr > <b > Autos </b > </a > &#183 ;
<a href = r/cf > Classifieds </a > &#183 ;
<a href = r/sh > Shopping </a > &#183 ;
<a href = r/ta > Travel </a > &#183 ;
<a href = r/yp > Yellow Pgs </a > &#183 ;
<a href = r/mp > Maps </a >
  ;
```

<표 6>은 제안하는 두 가지 방법에 대한 알고리즘의 의사코드이다.

〈표 6〉 제안하는 알고리즘

```
for i := 0 to 127 do
  append as a 1-symbol string to the dictionary ;
for i := 128 to 255 do
  append as a 'predefined well used string' to the dictionary ;
append λ to the dictionary ;
di := dictionary index of λ ;
δ := t_p ; α := 0 ;
repeat
  store the current file pointer and read a word(wd) ;
  if wd is in the dictionary then
        wd's code number is stored in ch ;
```

```
else retrieve the stored file pointer and read a character(ch) ;
if << di, ch >> is in the dictionary then
    di := dictionary index of << di, ch >> ;
else
        output (di) ;
        if the dictionary space is full then
                remove leaf nodes with the smallest counter values
                while dictionary space is available ;
        append << di, ch >> to the dictionary ;
        di := dictionary index of ch ;
        α := α + 1 ;
        if α is bigger than δ then
                reduce every leaf node's counter value by 1 ;
                α := 0 ;
        endif ;
endif ;
until end-of-input ;
```

다음 장은 본 논문이 제안하는 방법과 기존의 방법의 실험 결과를 보여준다.

5. 실험 결과

본 논문에서 제시하는 방법들을 검증하기 위해 두 가지 종류의 데이터를 사용했다. 첫째로 일반 문서의 검증을 위해서 표준 Calgary Corpus[9]를 대상으로 실험했다. 실험을 위해 제시하는 방법의 사전의 크기를 2^{10}(1024)으로 하였으며, 1024개의 엔트리 중 900개가 사용되면 그 때부터 20개의 엔트리가 사용될 때마다 모든 리프 노드의 카운트 값을 떨어뜨리기 시작하고 1024개 모두 사용되면 카운트 값이 작은것 순서대로 삭제하여 124개의 엔트리 공간을 확보하면 다시 압축을 시작하도록 하였다.

일반문서에 대한 실험결과는 <표 7>에서 보는 바와 같이 제안하는 방법이 기존의 V.42bis에 비해서 3.5%, UNIX의 compress에 비해서는 8.5%의 향상된 결과를 나타냈다.

또한, HTML 문서의 압축효과를 검증하기 위해서 현재

〈표 7〉 일반문서의 실험 결과(단위 KB)

파일 명	파일 크기	V.42bis		UNIX compress		제안하는 방법	
		압축파일크기	압축비율	압축파일크기	압축비율	압축파일크기	압축비율
paper1	51.9	22.34	56.9%	24.40	53.0%	22.00	57.6%
paper2	80.2	33.13	58.7%	35.30	56.0%	32.50	59.5%
paper3	45.4	19.22	57.7%	21.60	52.4%	19.13	57.9%
paper4	12.9	5.38	58.3%	6.79	47.4%	5.56	56.9%
paper5	11.6	5.13	55.8%	6.42	44.7%	5.28	54.5%
paper6	37.2	16.03	56.9%	18.20	51.1%	15.88	57.3%
book1	750.0	310.94	58.5%	327.00	56.4%	289.06	61.5%
book2	596.0	255.31	57.2%	251.00	57.9%	235.63	60.5%
progc	38.6	17.88	53.7%	18.60	51.8%	16.31	57.7%
progl	69.9	29.41	57.9%	26.50	62.1%	25.53	63.5%
progp	48.2	19.41	59.7%	18.70	61.2%	17.81	63.0%
news	368.0	178.13	51.6%	180.00	51.1%	161.25	56.2%
trans	91.4	40.94	55.2%	37.30	59.2%	36.88	59.7%
bib	108.0	43.13	60.1%	45.40	58.0%	42.19	60.9%
합 계	**2309.30**	**996.34**		**1017.21**		**925.00**	
평 균			**57.0%**		**54.4%**		**59.0%**

〈표 8〉 HTML 문서의 실험 결과(단위 KB)

웹 사이트 명	파일 크기	V.42bis		UNIX compress		V.42bis(확장사전)		제안하는 방법	
		압축파일크기	압축비율	압축파일크기	압축비율	압축파일크기	압축비율	압축파일크기	압축비율
www.yahoo.com	18.7	7.34	60.8%	8.50	54.6%	5.66	69.8%	5.59	70.1%
www.amazon.com	48.5	20.41	58.0%	18.20	62.5%	16.69	65.6%	15.06	69.0%
www.whitehouse.org	23.0	12.84	44.2%	11.00	52.2%	8.91	61.3%	7.84	65.9%
www.time.com	42.4	22.56	46.8%	18.60	56.1%	17.38	59.0%	14.34	66.2%
www.nato.int	16.6	6.06	63.5%	6.94	58.2%	4.66	72.0%	4.06	75.5%
www.unicef.org	14.7	7.03	52.0%	7.07	51.7%	6.22	57.6%	5.00	65.9%
www.abc.com	49.3	24.81	49.7%	21.90	55.6%	19.13	61.2%	16.53	66.5%
www.latimes.com	72.2	31.88	55.9%	26.30	63.6%	24.31	66.3%	21.31	70.5%
www.ox.ac.uk/research	5.4	2.36	56.4%	3.17	41.4%	1.97	63.6%	2.03	62.5%
www.harvard.edu	9.7	3.84	60.4%	4.57	52.9%	3.00	69.0%	3.04	68.7%
합 계	**300.53**	**139.14**		**126.25**		**107.91**		**94.82**	
평 균			**54.7%**		**54.9%**		**64.5%**		**68.0%**

인터넷상의 대표적인 몇 개의 사이트를 임의로 선정하여 실험하였다. 이때의 실험결과는 <표 8>과 같이 제안하는 방법이 기존의 V.42bis에 비해서 24.3%, UNIX의 compress에 비해서는 23.9%의 향상된 결과를 나타냈다. 또한 V.42bis에 초기 확장 사전을 적용하였을 경우에는 초기 확장 사전을 적용하지 않은 V.42bis에 비해 17.9%, UNIX의 compress에 비해서는 17.5%의 향상된 결과를 나타냈다.

우리는 본 실험을 위해 IBM PC(Pentium IV, 256M RAM)를 사용하였으며 C++를 이용해 프로그램을 구현하였다.

6. 결 론

본 논문은 대표적인 사전 방식 압축 알고리즘인 LZW의 사전 관리 방안을 중심으로 먼저 실제적으로 LZW를 이용하는 UNIX의 compress와 모뎀 통신 프로토콜인 V.42bis에 대해 살펴보았다. 그리고 사전의 재사용성과 연관하여 V.42bis를 개선한 보다 적응성이 강화된 새로운 알고리즘을 제시하였다. 이와 함께 HTML 문서의 효과적인 압축을 위해 태그와 같이 자주 사용되는 단어를 미리 LZW사전의 빈자리에 채워 넣고 압축을 수행하는 방안을 제시하였다. 우리는 실험을 통해 제안하는 두 가지 방법이 일반문서와 하이퍼텍스트 문서에 대해 기존의 방법들보다 우수한 압축 효과를 나타냄을 알 수 있었다. 그러나 제안하는 첫 번째 방법의 경우 카운터의 유지관리를 위해서 부가적인 연산시간이 소요되는데 이 점은 앞으로 연구해야할 부분이다.

참 고 문 헌

[1] Salomon D., "Data Compression-the complete reference," Springer, 1997.

[2] Baeza-Yates R. and Riebeiro-Neto B., "Modern Information Retrieval," Addison Wesley, 1999.

[3] Ziv, J. and Lempel A., "A Universal Algorithm for Sequential Data Compression," IEEE Transaction on Information Theory IT-23(3) : pp.337-343, 1977.

[4] Ziv, J. and Lempel A., "Compression of Individual Sequences via Variable-Rate Coding," IEEE Transaction on Information Theory IT-24(5) : pp.530-536, 1978.

[5] Welch T., "A Technique for High Performance Data Compression," IEEE Computer, Vol.17, No.6, pp.8-19, 1984.

[6] Phillips and Dwayne, "LZW Data Compression," The Computer Application Journal Circuit Cellar Inc., 27 : pp.36-48, June/July.

[7] Horspool, N. R., "Improving LZW," in Proceedings of the 1991 Data Compression Conference, J. Storer Ed., Los Alamitos, CA, IEEE Computer Society Press, pp.332-341, 1991.

[8] Thomborson and Clark "The V.42bis Standard for Data-Compressing Modems," IEEE Micro, pp.41-53, October, 1992.

[9] Available at ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus.

신 광 철

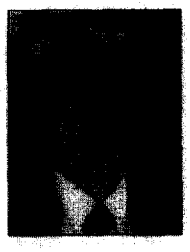
e-mail : kcshin@archi.cse.cau.ac.kr

1996년 중앙대학교 전자계산학과 졸업 (공학사)

1998년 중앙대학교 컴퓨터공학과 공학석사

2001년~현재 중앙대학교 컴퓨터공학과 박사과정

관심분야 : 인터넷 응용, 정보 검색, 웹 마이닝



한 상 용

e-mail : hansy@cau.ac.kr

1975년 서울대학교 공과대학 졸업(공학사)

1977년~1978년 KIST 연구원

1984년 미네소타 대학 컴퓨터공학과 공학박사

1984년~1995년 미국 IBM 연구소 책임연구원

1995년~현재 중앙대학교 컴퓨터공학과 교수

관심분야 : 전자 상거래, 인터넷 응용