LZW 알고리즘의 초기 사전 확장을 통한 효과적인 Hyper-Text 문서 압축

신광철°, 한상용
중앙대학교 컴퓨터 공학과
kcshin@archi.cse.cau.ac.kr°, hansy@cau.ac.kr

Expanding LZW initial dictionary for the effective compression of hyper-text

Kwangcheol Shin°, Sangyong Han
Dept. of Computer Science & Engineering, Chung-Ang Univ.

요 약

LZW 알고리즘은 사전방식의 압축 알고리즘인 LZ78의 변형된 형태로써 높은 압축률을 제공하기 때문에 많은 압축 방법에서 사용되고 있다. LZW에서 사전을 이용하는 방식은 LZ78과는 달리 미리 문서에 나오는 문자들을(즉, ASCII 코드) 미리 저장해 놓고 압축을 한다. 본 논문은 Hyper-Text에서는 같은 단어가 자주 반복되는 것에 착안하여 그러한 단어를 코드형태로 미리 저장해 놓고 LZW알고리즘을 적용하였다. 본 논문에서 제시하는 방법의 성능을 테스트하기 위해 기존의 V.42bis에 이러한 확장된 초기사전을 적용한 후 기존의 V.42bis 및 UNIX의 compress에 비교하였다. 실험결과를 통해 기존의 V.42bis와 compress에 비해 각각 17.9%와 17.5%의 향상된 성능을 확인할 수 있다.

1. 서론

데이터 압축은 컴퓨팅 분야에서 가장 중요한 것 중 하나이며 Text로부터 시작하여 이미지, 소리의 압축까지 다양한 형태의 기법들이 연구되어 왔다. 특히, 네트웍을 통한 데이터의 전송이 보편화되면서부터 더욱 부각이 되었다.

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. 이 때 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.

Lempel과 Ziv가 개발한 LZ77[2], LZ78[3]은 사전을 이용하는 압축기법을 사용하는 대표적인 알고리즘이다. 그들이 알고리즘을 개발한 이후로 압축을 위해 사전을 이용하는 방법이 다각적으로 연구가 되었고 계속적으로 더 나은 성능을 나타내는 수많은 변종 알고리즘이 개발되었다. 그것의 대표적인 예가 LZW알고리즘[4]이다. 이 방법은 사전에 모든 문자와 기호들을 미리 저장해 놓고 압축을 시작함으로써 인코드되는 데이터의 요소를 하나로 줄이는 효과를 보여 주었다. 그 후로 LZW를 기반으로 한 프로그램들이 개발되었고, 대표적인 예로 UNIX의 compress[7]가 있고 또한 Modem 통신 프로토콜인 V.42bis[6]에서도 이용되고 있다.

본 논문에서는 HTML 문서와 같이 같은 용어가 반복되는 문서를 위한 초기사전 확장개념에 기반을 둔 새로운 방법을 제시하고 이를 구현하여 기존의 방법과 비교하였다.

본 논문은 다음과 같이 구성되어있다. 2장에서는 LZW 알고리즘에 대해 알아보고, 3장에서는 LZW알고리즘을 기반으로 한 통신 프로토콜 V.42bis에서의 LZW 적용 예를 살펴본다. 4장에서는 본 논문에서 Hyper-Text문서 압축을 위한 LZW 초기 확장 사전을 제시하며, 5장에서는 실험결과를 보여준다. 마지막 6장에서는 결론을 기술한다.

2. LZW 알고리즘

LZW알고리즘[4]은 LZ78의 대표적인 변종으로서 T. Welch에 의해 1984년에 개발되었다. LZW의 기조가 되는 알고리즘인 LZ78[3]은 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 구성되어있다. LZW는 모든 문자를 사전에 미리 기록해 놓고 압축을 시작한다. 즉, 8bit 문자를 사용할 경우, 사전의 처음 256개는 데이터가 입력되기 전에 미리 채워져 있게 되는 것이다.

표 1. LZ78의 토큰

Dictionary	Token	Dictionary	Token
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"	(0,"i")	13 "sil"	(5,"l")
6 "d"	(0,"d")	14 "y"	(0,"y")
7 "_e"	(0,"e")	15 "_t"	(4,"t")

LZW의 원리는 다음과 같다. 인코더는 문자를 하나씩 받아들이고 String 변수 I에 저장한다. I에 저장된 값이 사전에서 발견되면 계속해서 다음 문자를 읽고 I에 concatenate시킨다. 다음 문자 x가 받아 드려져서 I에 붙인 후 사전에서 I값이 존재하지 않게 되면, (즉, 사전에 I는 존재하는데 Ix가 존재하진 않는다.) 그 때 인코더는 (1) String I를 가리키는 사전 포인터를 출력하고 (2)Ix를 사전의 빈 공간에 저장한 후 (3) I를 x로 초기화한다.

다음의 예를 들어 위의 과정을 설명한다.

"sir sid eastman easily teases sea sick seals"

위와 같은 Text가 있을 때 과정은 다음과 같다.

1. 0-255까지의 사전 엔트리를 초기화한다.
2. 첫 문자 "s"를 입력으로 받아서 사전에 있는지 검사한다. (ASCII code의 경우 115번째 엔트리에 s가 저장되어 있다.) 다음 문자 "i"를 받아서 "si"가 사전에 있는지 검사한다. 사전에 존재하지 않으므로 (1) 115를 출력하고, (2)string "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 인코더는 (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	255				

표 2.은 LZW 사전을 초기화한 후 위의 데이터를 가지고 처리하였을 경우의 사전 내용을 보여주고 있다.

위의 예에 대한 완전한 출력 스트림은 다음과 같다.(괄호 안의 문자는 이해를 돕기 위한 것으로 실제로는 숫자만이 출력된다.)

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.은 알고리즘의 pseudo-code를 나타낸 것이다. 여기서 λ는 빈 문자열을 나타내는 것이며, <<a,b>>는 문자열 a와 b를 결합하는 것을 나타낸다.

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;
```

LZW 알고리즘에서 사전의 처음 256 엔트리가 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 사전은 매우 빨리 가득 차게 된다. 이에 대한 가능한 해결책을 다음과 같이 생각해 볼 수 있다.

1. 사전이 가득 찬 시점에서 사전을 동결(freeze)시킨다. 사전은 정적인 상태가 되나 여전히 입력을 인코드할 수 있다.
2. 사전이 가득 차게 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 유틸리티가 사용하는 방법이 있다. 이 방법은 사전의 크기를 동적으로 유지하는 것으로서 압축을 시작할 때 사전의 크기를 2^9인 512로 시작하여 사전이 가득차게 되면 2^{10}인 1024로 늘려서 결국 216까지 확장한다. 2^{16}의 엔트리가 가득 차게 되면 압축률을 감시하고 있다가 일정 비율 이하로 압축률이 떨어지면 사전의 모든 엔트리를 삭제하고 다시 2^9인 512부터 압축을 시작하는 방법이다.
4. 사전이 가득 차게 되었을 때 가장 사용되지 않는 엔트리를 삭제하여 새로운 문자들을 받아들일 수 있게 한다. 이 방법은 대표적으로 모뎀 통신 프로토콜인 V.42bis에서 사용되는 것으로 다음 장에서 살펴본다.

3. V.42bis 프로토콜에서의 사전 관리 방식

V.42bis 프로토콜[6]은 빠른 모뎀을 사용하기 위해 ITU-T에서 제정한 표준이다. 이것은 존재하는 V32bis 프로토콜에 기반하고 57.6K baud에 이르는 빠른 전송률을 위해 제안된 것이다. ITU-T 표준은 권고(recommendations)사항이다. 그러나 이것들은 일반적으로 모든 주요한 모뎀 제조사에서 따르고 있다. 표준안은 데이터 압축과 에러 감지에 대해 명시하고 있으나 여기서는 압축에 대해서만 언급한다.

V.42bis에서는 압축이 사용되지 않는 transparent와 LZW변형을 사용하는 compressed의 두 가지 모드를 명시하고 있다. 전자는 압축이 제대로 되지 않거나 혹은 이미 압축한 데이터를 전송할 때 주로 이용된다.

compress 모드는 증가하는 크기를 가진 사전을 이용하며 초기 크기는 모뎀이 초기연결을 하면서 모뎀간에 협상을 통해 결정한다. V.42bis는 사전의 크기를 최대 2048엔트리, 최소 512엔트리를 사용하도록 권고하고 있다. 처음의 3개의 엔트리(즉, 0, 1, 2를 가리키는 포인터)는 어떤 문구도 포함하지 않고 단지 특별코드로 사용된다. 코드 0(enter transparent mode-ETM)은 인코더가 낮은 압축률을 사용할 때 보내지며 인코더는 압축되지 않은 데이터를 보내기로 결정한 것을 나타낸다. 코드 1 (FLUSH)은 데이터를 flush하기 위해 사용된다. 코드 2 (STEPUP)는 사전이 거의 차게 되었을 때 전송되는 것으로서 인코더는 사전을 두 배로 늘이게 된다. (사전의 크기가 특정 값(threshold)을 넘어가게 되면 사전은 거의 차게 된 것으로 인식하게 된다.) 사전이 가득 차게 되면 V.42bis는 재사용 처리를 권고한다. 즉, 새로운 문구를 받아들이기 위해 최근에 가장 쓰이지 않은 문구를 찾아 삭제하는 것이다. 이것은 256번째 엔트리부터 시작하여 다른 문구의 접두어가 되지 않는 것을 찾아 제거하는 것이다.

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 있으며, 이로써 사전은 항상 최근의 입력 패턴을 반영할 수 있게된다.

4. 초기 사전 확장을 통한 Hyper-Text 문서의 압축

본 논문에서 제시하는 방법은 기존의 LZW알고리즘이 모든 문자 set를 저장하고 있으면서 압축을 시행하는 것과 Hyper-Text 문서에서는 동일한 단어가 아주 많이 등장한다는 것에 착안하여 처음부터 표 4.에서 점선으로 표시한 영역과 같이 ASCII 코드가 끝나는 시점인 128번째부터 256번째까지 128개의 자주 등장하는 단어를 코드형태로 미리 저장해 놓고 압축을 시작한다. (실제로 그림 1.과 같이 LZW에서 128번째부터 256번째까지의 엔트리는 사용되지 않고 비어있기 때문에 그 영역에 다른 데이터를 넣고 압축을 수행해도 전체 사전의 크기에는 전혀 영향을 주지 않는다.) 이와 같은 초기 사전 확장을 이용하면 Hyper-Text문서 매우 효과적으로 압축할 수 있다.

압축과정에 대해 다음의 예를 통해 설명하겠다. 표 5.와 같은 HTML 문서가 있다고 가정하면 문서에서 자주 반복되는 용어는 "<a href=","<b>",""등과 같은 것들로 이것은 확장된 초기사전에 각각 134, 136, 135의 코드에 대응하는 것이다. 따라서 출력되는 데이터는 134(<a href=), 115(s), 47(/),

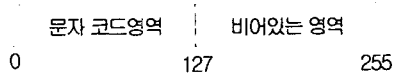

그림 1. 사전의 사용양식

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	254	.gif
.		134	<a href=	151	left	255	.html
97	a	135		152	right	256	face

50(2), 48(0), 49(1), 50(2), 62(>), 136(), 65(A),...의 형태가 되는 것이다. 이처럼 확장된 사전을 이용하면 반복되는 단어들에 대해 단 하나의 코드 값을 이용해 압축할 수 있으므로 기존의 LZW의 사전에 비해 효과적인 압축을 수행하게 된다.

다음 장에서는 본 논문이 제시하는 방법에 대한 다양한 결과들을 명시하였다.

5. 실험결과

우리는 본 논문에서 제시하는 방법을 검증하기 위해 V.42bis 프로토콜의 압축 방법을 구현하였고 V.42bis에 본 논문의 처리과정을 추가하였다.

우리는 실험 결과를 얻기 위해 웹 상에 있는 다양한 문서들을 이용하였다(표 6). 압축효과를 비교하기 위해 압축되기 전의 파일크기를 명시하였고, 각각의 방법으로 압축했을 때의 파일크기를 명시하였다. 실험결과 본 논문에서 제시하는 방법을 사용한 V.42bis가 기존의 V.42bis에 비해서는 17.9%, UNIX의 compress에 비해서는 17.5%의 성능향상이 있었다.

6. 결론

본 논문에서는 최근의 방대한 인터넷 문서의 저장과 전송 문

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></tabl
e></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>
 
```

가 다두되고 있는 상황에서 Hyper-Text를 효율적으로 저장하기 위해 LZW의 초기 사전 확장을 통한 압축 방안에 대해 살펴보았고 다양한 데이터를 통해 실험하였다. 실험결과 본 논문에서 제안하는 방법을 사용하여 기존의 LZW계열의 압축 프로그램에 적용할 경우 대략 17%의 추가적인 성능향상 결과를 얻었다.

- 참고 문헌 -

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

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

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

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

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

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

[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.

표 6. HTML 문서의 실험 결과 (단위 KB)

Web site name	Raw size	V.42bis		UNIX compress		V.42bis(초기확장사전)	
		size	ratio	size	ratio	size	ratio
www.yahoo.com	18.7	7.34	60.8%	8.50	54.6%	5.66	69.8%
www.amazon.com	48.5	20.41	58.0%	18.20	62.5%	16.69	65.6%
www.whitehouse.org	23.0	12.84	44.2%	11.00	52.2%	8.91	61.3%
www.time.com	42.4	22.56	46.8%	18.60	56.1%	17.38	59.0%
www.nato.int	16.6	6.06	63.5%	6.94	58.2%	4.66	72.0%
www.unicef.org	14.7	7.03	52.0%	7.07	51.7%	6.22	57.6%
www.abc.com	49.3	24.81	49.7%	21.90	55.6%	19.13	61.2%
www.latimes.com	72.2	31.88	55.9%	26.30	63.6%	24.31	66.3%
www.ox.ac.uk/research	5.4	2.36	56.4%	3.17	41.4%	1.97	63.6%
www.harvard.edu	9.7	3.84	60.4%	4.57	52.9%	3.00	69.0%
total	300.53	139.14		126.25		107.91	
average			54.7%		54.9%		64.5%