DOI : 10.5392/JKCA.2010.10.12.070

# 임베디드시스템을 위한 혼용텍스트 파일의 개선된 LZW 압축 알고리즘 구현

## Development on Improved of LZW Compression Algorithm by Mixed Text File for Embedded System

조미남, 지유강
동신대학교 정보통신공학과

Mi-Nam Cho(mncho1004@dsu.ac.kr), Yoo-Kang Ji(neo@dsu.ac.kr)

**요약**

최근의 스마트폰, 임베디드시스템 등의 정보통신 단말기는 데이터의 송·수신 및 분산처리 등의 업무를 수행하기 위하여 데이터의 크기를 축소시키는 압축률 향상이 매우 크게 대두되어졌다. 일반적으로 텍스트의 압축에는 LZW(Lempel Ziv Welch)알고리즘을 활용하고 있다. 그러나 LZW알고리즘은 1Byte 조합형 텍스트(알파벳 등)의 압축에는 효율적이나 2Byte 완성형 텍스트(한글 등)에 압축률이 현저하게 저하되는 단점을 가지고 있다. 이를 극복하기 위하여 본 논문에서는 2Byte 전위 필드(prefix)와 반복 계수를 위한 1Byte 후위 필드(suffix)를 사용하는 확장된 ELZW(EBCDIC Lempel Ziv Welch)알고리즘을 제안한다. 제안 알고리즘은 압축률 증가를 위해 압축사전을 구성하여, 알파벳, 한글, 포인터에 따라 각각 서로 다른 비트 스트링으로 적절하게 패킹된다. 제안하는 알고리즘의 성능분석을 위하여 각 140,355byte의 영문, 한글, 한영혼용 텍스트를 비교 실험하였고, 실험결과 제안한 ELZW알고리즘의 압축률은 기존의 1Byte 방식의 LZW 알고리즘보다 5.22% 더 우수하고, 2Byte LZW 알고리즘 보다 8.96% 더 우수함을 보였다.

■ 중심어 : | 압축알고리즘 | 패킹 | 스퀴징 | 크런칭 | 스쿼싱 |

**Abstract**

This paper Extended ELZW(EBCDIC Lempel Ziv Welch) algorithm uses 2 byte prefix field for pointer of a table and 1 byte suffix field for repeat counter. where, a prefix field uses a pointer(index) of compression table and a suffix field uses a counter of overlapping or recursion text data in compression table.

To increase compression ratio, after construction of compression table, table data are properly packed as different bit string in accordance with a alphabet, Hangeul, and pointer respectively. Therefore, proposed ELZW algorithm is superior to 1byte LZW algorithm as 5.22 percent and superior to 2byte LZW algorithm as 8.96 percent.

■ keyword : | LZW Algorithm | Packing | Sqeezing | Cruching | Squashing |

## I. 서 론

최근 데이터 통신의 급속한 발달은 서로 다른 컴퓨터에 저장된 데이터를 송·수신하거나 분산 처리를 수행하여 많은 정보를 서로 공유할 수 있도록 하고 있다. 데이터 전송과 보관에 있어 데이터의 크기 즉, 압축률은

접수번호 : #101001-001
접수일자 : 2010년 10월 01일
심사완료일 : 2010년 12월 23일
교신저자 : 지유강, e-mail : neo@dsu.ac.kr

매우 중요한 과제이다. 최근까지 많은 연구 결과가 발표되었으며, 실용화단계의 압축 프로그램도 활성화 되었으나 임베디드시스템, 스마트폰 등의 발달로 더욱 효과적인 압축 기법이 요구되고 있다.

기존에 발표된 데이터 압축 프로그램들은 거의 외국에서 개발된 소프트웨어로서 1바이트로 처리되는 영문 ASCII 코드 체계나 EBCDIC 코드 체계의 텍스트 파일을 압축하기에 적절한 프로그램으로 개발되었다. 이것은 2바이트 한글이나 한글과 영문 혼용 텍스트 파일을 압축하기에는 부적절하여 압축률이 현저히 떨어지는 단점을 가지고 있다.

텍스트파일의 압축을 위한 연구는 활발하게 진행되고 있으나 대부분 LZW(Lempel Ziv Welch) 계열의 압축 알고리즘을 확장한 형태로 두 가지 방향으로 연구되고 있다. 첫째로, 2바이트 한글을 영문처럼 1바이트 단위로 분리하여 압축하는 방법을 사용하는 방법 그리고, 한글의 크기인 2바이트 단위로 압축을 수행하는 방법을 사용하고 있다. 전자는 한글/영문 혼용 파일의 경우 1바이트 단위로 분리하여 압축을 수행하기 때문에 2바이트 한글의 압축에는 상당히 비효율적일 수 밖에 없으며, 후자인 경우는 한글과 영문을 모두 2바이트로 간주하여 압축을 시도하기 때문에 순수 한글 압축에는 상당히 효과적이지만 상대적으로 영문 압축에는 압축률이 떨어지는 단점을 갖고 있다[1][2].

따라서 본 논문에서는 2바이트 한글과 1바이트 영문을 압축하는데 있어서 기존의 LZW계열의 압축 알고리즘인 LZW1과 2바이트 한글과 1바이트 영문을 효과적으로 압축할 수 있는 LZW2 알고리즘에 대하여 분석하고, 한글과 영문 혼용 텍스트의 압축에 적합하도록 새로운 ELZW(EBCDIC Lempel Ziv Welch) 알고리즘을 제안한다.

## II. LZW계열의 한글 압축 알고리즘의 분석

### 1. 1Byte 한글 압축 알고리즘(LZW1)

한글 텍스트 파일을 압축하기 위한 LZW 압축 알고리즘은 입력 데이터를 이진 코드화하여 문자열로 나누기 위해 greedy 파싱 알고리즘을 사용한다. 압축을 수행하기 위한 기억 장소 내의 문자열 테이블의 각 원소는 2바이트로 구성된 상위 포인터(p)와 1바이트로 구성된 하위 문자(c)로 구성된다.

step 1 : 8 비트를 사용하여 발생할 수 있는 이진 부호 256개의 단일 문자열을 초기 압축 테이블에 저장한다.

step 2 : 한 개의 입력 데이터를 읽어 초기화된 압축 테이블로부터 해당 문자에 관한 주소를 구하여 압축 테이블의 상위 포인터(p)에 저장한다.

step 3 : 다음 입력 데이터를 읽어 다음과 같은 2단계 과정을 거쳐 압축을 수행한다.

1) 더 이상 입력이 없으면 Step 2에서 할당한 포인터를 코드화하여 출력한다.
2) pc가 압축 테이블에 등록되어 있으면 pc를 새로운 문자열 p로 하고 Step 3을 수행한다.
3) pc가 압축 테이블에 존재하지 않으면 문자열 p에 할당된 코드를 출력한 다음,
   (1) 압축 테이블에 pc에 대한 문자열(p′)를 생성하여 등록한다.
   (2) 그 다음 문자 c를 다시 새로운 문자열로 취하고 Step 3으로 간다.

[표 1]은 LZW 알고리즘의 Pseudo-code를 나타낸 것이다. 여기서 λ는 빈 문자열을 나타내는 것이며, <<di, ch>>는 문자열 di와 ch를 결합하는 것을 나타낸다[3].

**표 1. LZW 알고리즘의 Pseudo-code[3]**

```
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;
```

"ABC 한글 한글a"이란 문자열을 압축하는 과정을 LZW 알고리즘에 적용하면, 먼저 [표 2]와 같이 초기 압축 테이블이 구성되고 [표 3]과 같이 압축 테이블에 저장하게 된다.

**표 2. 초기압축테이블**

| 테이블 번호 | ASCII 문자 |
|---|---|
| . . . | . . . |
| 32 | 공백 |
| 65 | A |
| 66 | B |
| 67 | C |
| 97 | a |
| 177 | ▒ |
| 199 | ╟ |
| 209 | ╤ |
| 219 | ▌ |
| . . . | . . . |

**표 3. 압축테이블**

| 코드 | 문 자 열 | |
|---|---|---|
| | prefix (2바이트) | suffix (1바이트) |
| 257 | 65(A) | (66)B |
| 258 | 66(B) | 67(C) |
| 259 | 67(C) | 32( ) |
| 260 | 32( ) | 199(╟) |
| 261 | 199(╟) | 177(▒) |
| 262 | 177(▒) | 209(╤) |
| 263 | 209(╤) | 219(▌) |
| 264 | 260 | 177(▒) |
| 265 | 263 | 97(a) |
| 266 | . . . | . . . |

[표 3]의 압축 테이블에서 보는 것처럼 2바이트 한글인 '한'과 '글'은 2바이트 단위로 처리되지 못하고 상위 바이트(prefix)와 하위 바이트(suffix)로 나누어져 확장 ASCII 코드 값에 대응하게 된다.

따라서, 2바이트 한글은 분할되어 압축 테이블에 등록되기 때문에 한글을 인식하지 못하며, 같은 입력 데이터 크기를 처리할 때 영문에 비하여 압축 테이블을 많이 차지하게 되므로 압축률이 낮을 수 밖에 없다. 이러한 문제점을 해결하기 위해서 압축 테이블의 초기 영역에 ASCII 부호와 한글 부호를 모두 저장하여 2바이트 한글의 압축 효율을 증가시키도록 하는 방법이 제안되었다.

## 2. 2Byte 한글 압축 알고리즘(LZW2)

이 알고리즘은 2바이트 한글을 효과적으로 압축하기 위해서 초기 압축 테이블의 상위 영역에 2바이트 한글의 부호를 초기화시키는 방법을 사용하는 것이다.

"ABC 한글 한글a한글과"이란 문자열을 위한 초기 압축 테이블이 다음의 [표 4]와 같이 구성되어 있다고 할 때, 2바이트로 구성된 압축 테이블에 "ABC 한글 한글a한글과"란 문자열을 압축 테이블에 저장하면 [표 5]와 같이 저장하게 된다.

**표 4. 초기압축테이블**

| 테이블 번호 (2바이트) | ASCII/ 한글문자 (2바이트) |
|---|---|
| . . . | . . . |
| 32 | 공백 |
| 65 | A |
| 66 | B |
| 67 | C |
| 97 | a |
| . . . | . . . |
| 1280 | 과 |
| 1300 | 글 |
| 2209 | 한 |

**표5. 압축테이블**

| 코드 | 문 자 열 | |
|---|---|---|
| | prefix (2바이트) | suffix (1바이트) |
| 2606 | A | B |
| 2607 | B | C |
| 2608 | C | 〈32〉 |
| 2609 | 〈32〉 | 한 |
| 2610 | 한 | 글 |
| 2611 | 글 | 〈32〉 |
| 2612 | 2609[〈32〉한] | 글 |
| 2613 | 글 | a |
| 2614 | a | 한 |
| 2615 | 2610[한글] | 과 |
| 2616 | . . . | . . . |

[표 5]의 압축테이블에서 보는 것처럼 한글의 압축에는 압축 효율이 LZW1에 비하여 상당히 증가함을 알 수 있다. 그러나 영문인 경우 압축 테이블의 suffix를 위하여 1바이트 크기를 할당해도 충분하지만 한글 초기 테이블로 인하여 2바이트를 할당하여 사용하기 때문에 상대적으로 압축 테이블 크기가 커져 압축 파일의 크기가 증가할 수 밖에 없다[2].

그리고, LZW1과 LZW2를 압축 효율면에서 서로 비교하면 한글을 처리하는 단위와 문자열 길이에 있어서 trade off 관계가 성립한다. 특히, 한글과 영문 혼용 텍스트 문자열을 압축 테이블에 저장할 때 LZW1에서는 한글을 1바이트 단위로 분할하게 되고, LZW2에서는 하나의 영문과 한글을 분할하여 압축 테이블을 구성하게 된다([표 5] 압축 테이블의 2610, 2611 번지 참조). 따라서 한글과 영문 혼용 텍스트 문자열의 압축에 있어서 그 효율을 감소할 수 밖에 없다.

따라서 본 논문에서는 한글과 영문을 효과적으로 압축하기 위해서 LZW1과 LZW2의 장점만을 취하였으며, 압축 테이블의 prefix는 한글과 영문을 위한 포인터로서 2바이트 할당하고, suffix는 1바이트로 할당하여 압축 테이블의 각 원소의 바이트 크기를 줄였다.

suffix는 문자를 할당하는 공간으로 사용하는 것이 아니라 입력 파일로부터 읽어 들인 문자가 압축 테이블의 prefix에 등록되어 있을 때 연속적으로 일치하는 문자들의 수를 세는 계수 부분으로 사용하여 압축 파일의 크기를 줄일 수 있도록 하고 있다.

## Ⅲ. 제안한 압축 알고리즘(ELZW)

제안한 ELZW 알고리즘에서는 2바이트 한글과 ASCII 문자를 효과적으로 압축할 수 있도록 앞서 기술된 LZW1처럼 압축 테이블의 prefix인 상위 2바이트는 ASCII 문자와 2바이트 한글의 각 주소를 표현할 수 있도록 2바이트 포인터를 사용하며, 압축 테이블의 suffix인 하위 1바이트 계수로 사용한다. suffix는 입력 파일로부터 읽어 들인 문자가 이미 등록되어 있을 때 읽어 들인 문자열과 압축 테이블 내부에 등록되어 있는 문자와 연속적으로 일치하는 문자들의 수를 세는 계수 부분으로 사용한다.

이와 같은 방법을 사용하면 나중에 압축 테이블에 저장된 데이터로부터 압축 파일을 생성할 때 압축 테이블의 가변적 주소를 출력하는 것보다 더 적은 크기의 주소를 출력할 수 있어 압축 효율을 훨씬 높일 수 있다. 압축 테이블에 입력 문자를 등록하는 알고리즘은 [표 6]과 같다.

**표 6. 제안한 압축 알고리즘(ELZW)**

```
while((ch = getc(input_file)) != EOF)
{
 search ch in TABLE;
 if(ch is not exist in current_index of TABLE) {
  Set address of ch in prefix of TABLE[current_index];
  Set 0 in suffix of TABLE[current_index];
 }

else {
  move position to existed address of ch in TABLE;
  while((ch = getc(input_file)) != EOF) {
    if( ch is equal to TABLE[position] ) {
      increment position;
      increment count;
    } else {
    if(count 〉 1) {
      Set address of ch in prefix of
TABLE[current_index];
      Set count in suffix of TABLE;
    } else {
      Set address of ch in prefix of
TABLE[current_index];
      Set 0 in suffix of TABLE;
    }
   }
  }
 }
 increment current_index;
}
```

[표 7]에서 보는 것처럼 압축 테이블의 상위 바이트는 압축 테이블의 주소를 저장하고 하위 바이트는 압축 테이블 내의 반복되는 문자의 수를 저장하게 된다. “ABC 한글 한글a한글과 ABC 한글 한글a” 같은 문자열을 압축 테이블에 저장하면 [표 7]과 같은 압축 테이블을 구성할 수 있다.

**표 7. 압축 테이블**

| 코 드 | 문 자 열 | |
|---|---|---|
| | prefix (2바이트) | suffix (1바이트) |
| 2606 | 65[A] | 0 |
| 2607 | 67[B] | 0 |
| 2608 | 68[C] | 0 |
| 2609 | 32 | 0 |
| 2610 | C3D1[한] | 0 |
| 2611 | B1DB[글] | 0 |
| 2612 | 2609 | 3 |
| 2613 | 97[a] | 0 |
| 2614 | 2610 | 2 |
| 2615 | B0FA[과] | 0 |
| 2616 | 32 | 0 |
| 2617 | 2606 | 8 |
| 2618 | . . . | . . . |

[표 7]의 압축 테이블에 문자열을 등록하는 알고리즘은 먼저 입력 파일로부터 파일의 끝에 도달할 때까지 문자를 순서적으로 읽어 압축 테이블에 등록하게 된다. 만약 읽어 들인 문자가 압축 테이블에 등록되어 있지 않다면 해당 문자의 주소를 압축 테이블의 prefix에 저장하고 계수 필드인 suffix는 0으로 설정한다.

그러나 입력 테이블 내의 어떤 위치에 해당 문자가 이미 등록되어 있다면 입력 파일로부터 다음 문자를 읽어 압축 테이블의 다음 위치에 있는 문자의 주소와 비교하는 과정을 일치하지 않을 때까지 계속 반복하게 된다. 만약 일치하지 않게 되면 처음 일치한 곳의 주소와 그 때까지 일치 계수를 테이블의 prefix와 suffix에 각각 저장하면 된다.

보다 효과적인 압축률을 얻기 위해서는 압축 테이블에 존재하는 불필요한 비트를 제거하여 패킹하는 것이 좋다. 예를 들면, LZW2에서 suffix에 저장된 문자가 영

문인 경우 하위 바이트만 사용하고 상위 바이트는 불필요하지만 압축 파일을 생성할 때 그대로 출력되므로 압축된 파일 내부에는 불필요한 1바이트를 갖게 된다. 이와 같은 이유는 압축 테이블에 등록된 데이터를 전혀 패킹하지 않고 그대로 출력하기 때문에 발생하는 문제점이다. 따라서 이러한 불필요한 비트열을 압축 테이블에서 압축 파일로 데이터를 출력할 때 제거하는 것이 바람직하다. 따라서 본 논문에서는 압축 테이블에서 불필요한 비트를 삭제하는 프로그램을 추가하여 우수한 압축률을 갖도록 하였다.

일반적으로 압축 프로그램은 압축 테이블에 저장된 모든 내용을 그대로 압축 파일로 출력되는 방법을 사용하고 있다. 그러나 이는 부분적으로 불필요한 비트들마저도 그대로 출력되기 때문에 압축률을 저하시키는 원인이 된다.

따라서 본 논문에서는 [표 8]과 같은 알고리즘을 통해 압축 테이블에 저장된 데이터 중에서 실제 불필요한 데이터를 제거하여 압축 파일에 출력할 수 있도록 하는 방법을 사용하고 있다.

**표 8. 압축 테이블로부터 불필요한 비트열을 제거하는 알고리즘**

```
index = 2606;
. . .
packing( )
{
 while( TABLE_SIZE > index && index <= LIMIT)
 {
  read an element from TABLE[index];
  switch( prefix )
  {
   case 'table_address'   :
        write prefix to output_file;
        write suffix to output_file;
   case 'english_address' :
        write high byte of prefix to output_file;
   case 'hangeul_address':
        write prefix to out_file;
  }
 }
}
```

## IV. 실험 결과 및 분석

본 논문에서 제시한 압축 테이블로부터 불필요한 비트열을 제거하는 알고리즘을 문자열 “ABC 한글 한글a 한글과 ABC 한글 한글a”를 사용하여 생성된 압축 테이블에 적용하여 압축 파일을 출력하면 다음 [표 9]와 같은 결과를 얻을 수 있다.

**표 9. 압축 파일**

| A | B | C | 〈32〉 | 한 | 글 | 2609 | 1 | 2610 | 2 | a | 2610 | 2 | 과 | 〈32〉 | 2606 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

기존의 한글 압축 알고리즘은 LZW 알고리즘을 확장하여 한글 처리에 활용하고 있으나 앞에서 지적한 바와 같이 많은 문제점을 가지고 있다.

따라서, 본 논문에서는 기존의 LZW1과 LZW2에서 갖는 단점을 해결하기 위해서 압축 테이블의 suffix에 다음 문자를 입력하는 것이 아니라 압축 테이블 내의 반복되는 문자의 수를 계산하는 계수를 사용함으로써 다음과 같은 결과를 얻을 수 있다.

(1) 압축 테이블의 상위 prefix에만 영문 또는 한글의 주소를 입력함으로써 LZW2에서 한글이 상위 바이트와 하위 바이트로 나누어지는 문제점을 해결하여 압축 효율을 증가시켰다.

(2) 압축 테이블의 하위 suffix는 압축 테이블 내의 반복되는 문자의 계수로 사용함으로써 기존의 LZW1 이나 LZW2 에 비하여 전체적인 압축 파일의 크기를 증가시키는 문제점을 제거하였다.

(3) 초기화 테이블 이후의 등록된 문자가 오직 하나만 일치한 경우 해당 주소를 사용하지 않고 초기화된 주소를 압축 테이블에 등록함으로써 압축 파일의 생성시 주소의 크기를 줄였다.

(4) 압축 테이블에 저장된 데이터 중에서 불필요한 비트열을 제거하는 알고리즘을 추가하므로써 최종적인 압축률을 증가시켰다.

표 10. 기존 알고리즘과 제안 알고리즘 비교 분석

| 표본 | 원문 | LZW1 | | LZW2 | | 제안된 ELZW | |
|---|---|---|---|---|---|---|---|
| | | 압축문 | 압축률 | 압축문 | 압축률 | 압축문 | 압축률 |
| 영문 | 140,355 | 47,094 | 66.44 | 51,350 | 57.33 | 46,356 | 66.97 |
| 한글 | 140,355 | 76,356 | 45.59 | 68,878 | 50.92 | 70,945 | 49.45 |
| 한영혼용 | 140,355 | 77,227 | 44.97 | 82,485 | 41.23 | 69,910 | 50.19 |

* 데이터 크기는 바이트 단위
* 압축률={1-(압축후의 파일 크기/압축전의 파일 크기)}×100

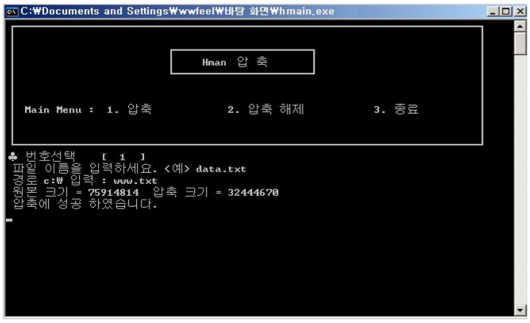


그림 1. 제안한 혼용 텍스트 파일 압축 실행

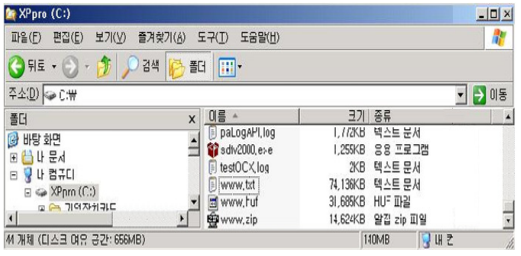


그림 2. 제안한 혼용 텍스트 파일 압축 결과

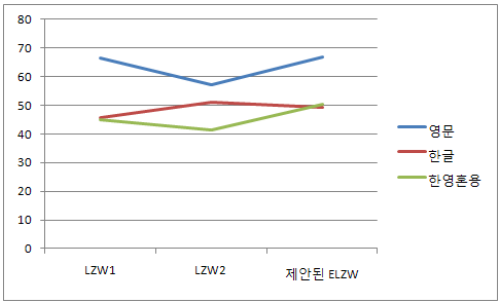


그림 3. 압축률 시뮬레이션 결과

## V. 결 론

본 논문에서 제안한 ELZW 알고리즘은 기존의 파일 압축에 비하여 압축률이 더 우수하다는 것을 얻을 수 있었다. 이는 제안한 알고리즘의 압축 테이블의 suffix는 1바이트만 사용하기 때문에 컴퓨터 내부의 주기억 장소를 절약할 수 있다. 또한 보다 효과적인 압축률을 얻기 위해 압축 파일 생성 시 압축 테이블에서 압축 파일로 데이터를 출력할 때 불필요한 비트를 제거하여 패킹하는 것이 좋다. 실제로 한글/영문 혼용 파일의 경우 상용화되어 있는 LZW1 압축 프로그램보다 5.22% 정도, LZW2의 압축 프로그램보다는 8.96% 정도의 압축률이 증가함을 보였다.

앞으로의 연구 과제는 한글/영문 혼용 텍스트 파일뿐만 아니라 영문, 한글 등 단일 문자 파일을 압축하는데 더욱 향상된 압축률을 갖을 수 있는 알고리즘으로의 개발이 요구된다.

## 참 고 문 헌

[1] 정진욱, “리얼 타임 환경에서 압축 코딩과 암호 코딩의 결합”, 한국정보과학회논문지, 제17권, 제6호, 1990.

[2] 한승조, “완성형 한글을 위한 압축 코딩과 암호화 방법”, 한국정보과학회논문지, 제20권, 제9호, 1993.

[3] 신광철, 한상용, “LZW 알고리즘의 초기 사전 확장을 통한 효과적인 Hyper-Text 문서 압축”, 한국정보과학회논문지, 제29권, 제1호, p.689, 2002.

[4] H. K. Reghbati, “An Overview of Data Copression Techiques,” Computer, Vol.14, No.4, pp.71-76, 1981(4),

[5] W. H. Lenug, “Inducing Codes Form Exaples,” DCC '91, Data Compression Conference, pp.267-276, 1991(4).

[6] M. Pechura, “File Archival Techiques Using Data Compression,” Comm.ACM, Vol.25, No.9,

pp.605-609, 1982(9).

[7] M. B. Lin, “A Hardware Architecture for the LZW Compression and Decompression Algorithms Based on Parallel Dictionaries,” Journal of VLSI signal processing systems for signal, image, and video technology, Vol.26, No.3 pp.369-381, 2004.

[8] 김대영, 이성열, 이혜영, “모바일 시스템을 위한 연결 데이터 압축 알고리즘”, 컴퓨터그래픽스학회논문지, 제14권, 제1호, pp.27-33, 2008.

## 저 자 소 개

**조 미 남(Mi-Nam Cho)** 정회원

- 2002년 2월 : 동신대학교 정보통신공학과(공학석사)
- 2009년 2월 : 동신대학교 정보통신공학과(공학박사)

<관심분야> : RFID시스템, 데이터베이스, 영상처리

**지 유 강(Yoo-Kang Ji)** 정회원

- 2002년 2월 : 동신대학교 정보통신공학과(공학석사)
- 2006년 2월 : 동신대학교 정보통신공학과(공학박사)
- 2006년 3월 ~ 2009년 8월 : 동신대학교 정보통신공학과 전임강사

<관심분야> : 영상통신, 디지털신호처리, 임베디드시스템, 컴퓨터네트워크